Decentralised Functional Signatures

Mobile Networks and Applications (2019) 24:934–946
https://doi.org/10.1007/s11036-018-1149-1

Decentralised Functional Signatures

Bei Liang[1] · **Aikaterini Mitrokotsa**[1]

Published online: 11 October 2018

Abstract

With the rapid development of the Internet of Things (IoT) a lot of critical information is shared however without having guarantees about the origin and integrity of the information. Digital signatures can provide important integrity guarantees to prevent illegal users from getting access to private and sensitive data in various IoT applications. Functional signatures, introduced by Boyle, Goldwasser and Ivan (PKC 2014) as signatures with a finegrained access control, allow an authority to generate signing keys corresponding to various functions such that a user with a signing key for a function f, can sign the image of the function f on a message m *i.e.,* can sign $f(m)$. Okamoto and Takashima (PKC 2013) firstly proposed the notion of a decentralized multi-authority functional signature (DMA-FS) scheme, which supports non-monotone access structures combined with inner-product relations. In this paper, we generalise the definition of DMA-FS proposed by Okamoto et al. (PKC13) for even more general policy functions, which support any polynomial-size boolean predicates other than the inner product relation and allow modifications of the original message. In our multi-authority functional signature (MAFS), there are multiple authorities and each one is able to certify a specific function and issue a corresponding functional signing key for each individual with some property, rendering them very useful in application settings such smart homes, smart cities, smart health care etc. We also provide a general transformation from a standard signature scheme to a MAFS scheme. Moreover, we present a way to build a function private MAFS from a FS without function privacy together with SNARKs.

Keywords IoT · Functional signatures · Attribute-based signature · Decentralised multi-authority functional signatures

1 Introduction

With the rapid development of the Internet of Things (IoT) in various application settings (e.g., smart homes [6, 7], smart cities [4, 8], and smart health-care [1]), ensuring the communication integrity and authenticity of information shared between IoT devices becomes a major concern, since millions of devices sense and communicate large volumes of private and sensitive data. Digital signatures is one of simplest and more reliable approaches, that can be employed to achieve the integrity and authentication properties and thus, prevent illegal users from getting access to private and sensitive data.

A digital signature on a message, originally introduced by Diffie and Hellman [5], produces information that allows the receiver to verify that the original message was indeed signed by the claimed signer (sender). Boyle, Goldwasser and Ivan [2] have recently introduced a new type of signatures, called *functional signatures*. In a functional signature scheme, a trusted authority holds a master secret key known only to the authority. Given a description of a function f, the authority using the master secret key can generate a functional signing key sk_f associated with the function f. Anyone that has access to the functional signing key sk_f and a message m can compute $f(m)$ as well as a functional signature σ of $f(m)$. Such fine-grained generation of signatures is extremely useful in multiple applications such as authentication and trust-negotiation. For instance, we can consider the case where a document (message m) has some fields to be filled in, *i.e.,* the initial document m needs to be modified by applying a function on it $f(m)$ and then subsequently signed. Thus, it is required to produce a signature σ that corresponds to $f(m)$. In that case, a function f could be associated with the fields of the document that need to be filled in, as well as the type of

✉ Bei Liang
lbei@chalmers.se

Aikaterini Mitrokotsa
aikmitr@chalmers.se

[1] Chalmers University of Technology, Gothenburg, Sweden

the information (*e.g.,* calendar date). Then, by generating a signing key sk_f for this specific function f, we can guarantee which documents have been filled in and signed by an individual.

In this paper, we go beyond standard functional signatures and we re-explore the existing concept: *decentralised or multi-authority functional signatures* for general policy functions, which aids for smart projects such for smart home, smart city, smart health care etc.

Decentralised functional signatures Although functional signatures (FS) are a very powerful primitive, the basic concept of FS has a serious challenge since it requires that a single authority issues for all users their secret signing keys associated with different functions. This definitely contradicts the distributed public key infrastructure that allows having multiple certification authorities with different levels of trust. When considering realistic applications of functional signatures, it is hard to imagine that a single central authority will be trusted by everyone and will manage all credentials and generate functional signing keys for different functions for all individuals. Inspired by the practical demand for the distributed trust, we explore the model for *multi-authority functional signatures* (MAFS).

Okamoto et al. [12] firstly proposed the concept of a decentralised multi-authority functional signature (DMA-FS) scheme, which supports non-monotone access structures combined with inner-product relations [11]. Intuitively, the non-monotone access structures combined with inner-product relations [11] supported by the DMA-FS scheme of Okamoto et al. are: for properties $\mathbf{u} := (u_1, \ldots, u_N) \in \mathbb{F}_q^{n_1+\ldots+n_N}$, a policy function $F := (\hat{M}, (v_1, \ldots, v_N) \in \mathbb{F}_q^{n_1+\ldots+n_N})$ is the componentwise inner-product relations for property vector components, e.g., $\{u_i \cdot v_i = 0 \text{ or not}\}_{i\in\{1,\ldots,N\}}$ that are taken as input to a span program $\hat{M}$, and the property vector $\mathbf{u}$ satisfies the policy F iff the truth-value vector of $((u_1 \cdot v_1 = 0), \ldots, (u_N \cdot v_N = 0))$ is accepted by the span program $\hat{M}$. In their DMA-FS setting, there are multiple authorities and each authority is able to generate a secret key associated with a sort of attributes u_i, *i.e.,* a user obtains several secret keys w.r.t. the attributes it has, $\mathbf{u} := (u_1, \ldots, u_N)$, each of which is issued by corresponding authority. A user with secret keys w.r.t. the attributes $\mathbf{u} := (u_1, \ldots, u_N)$ is able to sign the message iff $\mathbf{u}$ s.t. $((u_1 \cdot v_1 = 0), \ldots, (u_N \cdot v_N = 0))$ is accepted by the span program $\hat{M}$.

Contrary to Okamoto et al. [12] work, in this paper we consider whether it is possible to extend the DMA-FS with respect to even more general policy functions, which are not limited to inner-product predicates [11] but support any polynomial sized boolean predicate. Furthermore, we allow one to sign a message that is in the range of a function f, which can be considered as to make some operations on the original message first and then to sign the new resulting message.

A motivating example of using MAFS in the IoT setting Before we describe our generalized multi-authority FS (MAFS) scheme in more detail, let us consider a motivating example in the IoT setting. In order to allow both businesses and customers greater visibility and tracking of their products and parcels than ever before, every shipment or parcel can be equipped with an inexpensive small device or sensor which would continuously monitor and record every information from the position of a parcel to the temperature and light conditions– meaning that those sending and receiving the parcels can see both where their items are and how they are being handled on route and whether they have been opened or not. For instance, a headphone made by a company in Germany is ordered to be delivered to China by a logistics company. A rather challenging task is keeping and updating the records of all transactions during the delivery where multiple authorities (such as country of origin, logistics company, insurance company) might be involved, while ensuring that the users are able to verify all the transactions that occurred. The concept of *multi-authority functional signature* properly addresses this type of problems. For example, a headphone from Country A might be ordered to be delivered to Country B by a specific logistics company X with a insurance company. The transaction messages may be updated with info about the luminous intensity and the current location of the parcel should be signed through a MAFS with a policy such that ((Original country = Country A OR B) AND (Logistics = Company X OR Y) AND (Luminous Intensity $\leq$ 0.2lx)). In a MAFS scheme, a user obtains several secret keys associated with multiple properties, each of which is issued by a different authority. A sensor embedded in the parcel receives a secret key (for the country of origin A) from the producer S, a secret key for the logistics company X, and a secret key for a luminous intensity from the insurance company T, where the country of origin A, the logistics company X and the insurance company T are all individual authorities.. The sensor while the parcel in on route, updates the properties information of the parcel and signs the updated messages. A user can then verify that the signed transactions (depicted in Fig. 1) indeed occurred.

Our results Our main results and contributions can be summarised as follows: *(i)* we introduce for the definition of *multi-authority functional signatures* (MAFSs) for general policy functions; *(ii)* we provide a general transformation from a standard signature scheme to a MAFS scheme and a corresponding construction; *(iii)* given a non-function-

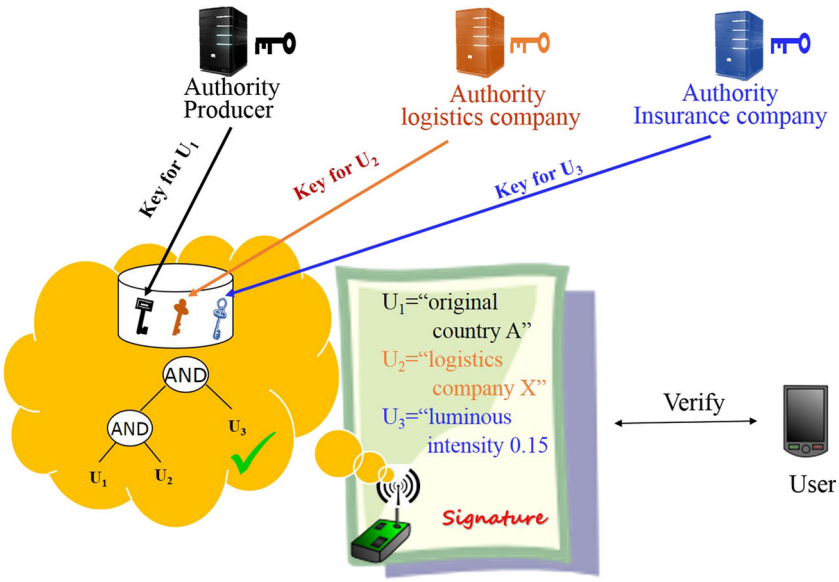

Fig. 1 An example of using the multi-authority functional signature scheme in the IoT setting

private FS scheme and a SNARK we provide a way to build a function private MAFS scheme.

In a MAFS scheme, multiple authorities can independently generate their master secret and master public verification keys. Any authority i should be able to generate a functional signing key for a user with identity UID and property U_i along with a policy F over various authorities. A user UID, with signing keys for a policy function F and different property U_i from the authority $i \in [N]$, is allowed to generate a signature for the new message $F(U_1, \ldots, U_n, m)$ only if his property set $\{U_i\}_{i\in[N]}$ satisfies the policy, while anyone using the N authorities' master verification keys should be able to verify the validity of the message-signature pair. An important requirement for MAFS is the security (unforgeability) against collusion attacks, which states that colluding users *e.g.,* whose identities are UID_1 with property U_1 and UID_2 with property U_2 (even if they collude with some corrupted authorities) should not be able to forge a signature endorsed by a user UID_1 with properties U_1 and U_2. Another desirable property of MAFS is *function privacy*, which implies that the signature should reveal neither the function F that the functional signing key used in the signing process corresponds to, nor the message m that F was applied to. In particular, the notion of MAFS generalizes that of multi-authority attribute based signature [12].

In this paper, we first give a construction of MAFS that is not function private, based on any standard signature scheme. Then, assuming the existence of succinct non-interactive arguments of knowledge (SNARKS) for NP languages, we propose a function-private MAFS scheme from a non-function-private FS scheme. The resulting function-private MAFS scheme is defined in the *common reference string model*, which means that we allow all algorithms (and the adversary) to get as input also a public uniformly distributed common reference string [9, 10]. Below we describe in an informal way, highlights of our results and the approaches we employ.

Theorem 1 (Informal) *Assuming the existence of an existentially unforgeable signature scheme, then there exists a multi-authority functional signature scheme satisfying the unforgeability requirement but not function privacy.*

Overview of the approach The master signing and verification key for each authority $i \in [N]$ will correspond to a key pair, $(\mathsf{msk}_i, \mathsf{mvk}_i)$, in an underlying (standard) signature scheme. To generate a signing key for a user with UID and property U for a function F, the authority i does the following. First, it samples a fresh signing and verification key pair $(\mathsf{sk}_i, \mathsf{vk}_i)$ in the underlying signature scheme, and then signs the concatenation $\mathsf{UID}\|U\|F\|\mathsf{vk}_i$ using msk_i. The signing key consists of this certificate (signature) together with sk_i. Given N signing keys, a user can sign any message $m^* = F(\{(i, U_i)\}_{i\in[N]}, m)$ by signing m using each sk_i, and outputting these N signatures, together with N certificates of $\mathsf{UID}\|U\|F\|\mathsf{vk}_i$.

Theorem 2 (Informal) *Assuming the existence of SNARKs and a functional signature scheme that is unforgeable, then there exists a multi-authority functional signature scheme in the common reference string model satisfying the unforgeability requirement and function privacy.*

Overview of the approach The setup algorithm generates a common reference string crs. The authority setup and

key generation algorithms are the same as those of the construction in Theorem 2. Given N signing keys, a user does the following. First, it signs m using each sk_i^f, and then it generates a zero-knowledge SNARK for the following statement: $\forall i \in [N]$, $\exists \sigma_i$ such that σ_i is a valid signature of $m^* = f(m)$ under mvk_i in the functional signature scheme. The final signature consists of the proof together with N certificates of $\mathsf{UID} \| U \| F \| \mathsf{vk}_i$.

2 Related work

Multi-authority functional encryption Inspired by the distributed trust offered by public-key encryption, Chandran et al. [3] introduced a new primitive that is called Multi-Authority Functional Encryption (MAFE) as a generalization of both Functional Encryption and Multi-Authority Attribute-Based Encryption (MA-ABE). Based on subexponentially secure indistinguishability obfuscation and injective one-way functions they show how to obtain MAFE for arbitrary polynomial-size circuits, namely $F(\{U_{id}\}_{id \in \mathcal{S}}, m)$ which is a $|\mathcal{S}| + 1$ variate policy function (each component is in $\{0, 1\}^k$) in MAFE that takes as input up to $|\mathcal{S}|$ properties and a message and outputs a value. In this paper, we generalise the definition of DMA-FS in [12] for even more general policy functions, which supports any polynomial sized boolean predicates other than the inner product relation and allows modifications of the original message. Such a generalisation is not straightforward. To achieve that we take advantage of the method that Chandran et al. [3] used to define the function policy in a multi-authority setting.

Multi-authority attribute based signature The concept of multi-authority attribute based signature (MA-ABS) was introduced by Okamoto et al. [9, 10]. In a MA-ABS model, there are multiple authorities and each authority is responsible for issuing a secret key associated with a category of attributes. But a central trustee is required in addition to multiple authorities. Okamoto et al. [12] showed that if the central authority is corrupted in MA-ABS, the security (unforgeability) of the system will be totally broken. Thus, in [12] they proposed the first MA-ABS scheme with no central authority, which supports more general predicates, non-monotone access structures in which no central authority exists and no global coordination is required except for the setting of a parameter for a prime order bilinear group and hash functions under a standard assumption, the DLIN assumption in the random oracle model. They also proposed a more general signature scheme, DMA-FS scheme, which supports non-monotone access structures combined with inner-product relations [11], and prove that the proposed DMA-FS scheme is fully secure (adaptive-predicate unforgeable and perfect private in the DMA security model) under the DLIN assumption in the random oracle model.

From the point of view of the class of functions that multi-authority FS supports, the DMA-FS scheme proposed by Okamoto et al. [12] is a special case of our generalized MAFS scheme, where the underlying predicate is specialized to be the inner product relation. Okamoto et al.'s DMA-FS scheme is fully secure (adaptive-predicate unforgeable and perfect private in the DMA security model) under the DLIN assumption in the random oracle model, while our MAFS scheme is secure in the common reference string model satisfying the unforgeability requirement and function privacy. Moreover, the notion of MA-ABS proposed by Okamoto et al. [12] is a specific case of our generalized MAFS for general policy functions since the policy function in MA-ABS can be considered as the special case of our MAFS where the policy function F always outputs the message m if the properties satisfy F.

3 Preliminaries

Definition 1 (Functional signature [2]) A functional signature scheme for a message space $\mathcal{M}$, and function family $\mathcal{F} = \{f : \mathcal{D}_f \to \mathcal{M}\}$ consists of the PPT algorithms $\mathsf{FS} = (\mathsf{FS.Setup}, \mathsf{FS.KeyGen}, \mathsf{FS.Sign}, \mathsf{FS.Verify})$:

- $\mathsf{FS.Setup}(1^\lambda) \to (\mathsf{msk}, \mathsf{mvk})$: on input the security parameter 1^λ, the setup algorithm outputs the master signing key and the master verification key.
- $\mathsf{FS.KeyGen}(\mathsf{msk}, f) \to \mathsf{sk}_f$: on input the master signing key and a function $f \in \mathcal{F}$, the key generation algorithm outputs a signing key for f.
- $\mathsf{FS.Sign}(f, \mathsf{sk}_f, m) \to (f(m), \sigma)$: on input the signing key for function $f \in \mathcal{F}$ and a message $m \in \mathcal{D}_f$, the signing algorithm outputs $f(m)$ and a signature of $f(m)$.
- $\mathsf{FS.Verify}(\mathsf{mvk}, m^*, \sigma) \to \{0, 1\}$: on input the master verification key mvk, a message m^* and a signature σ, the verification algorithm outputs 1 if the signature is valid.

We require it to satisfy the following conditions:

Correctness $\forall f \in \mathcal{F}$, $\forall m \in \mathcal{D}_f$, $(\mathsf{msk}, \mathsf{mvk}) \leftarrow \mathsf{FS.Setup}(1^\lambda)$, $(m^*, \sigma) \leftarrow \mathsf{FS.Sign}(f, \mathsf{sk}_f, m)$, it holds that $\mathsf{FS.Verify}(\mathsf{mvk}, m^*, \sigma) = 1$.

Unforgeability The scheme is unforgeable if the advantage of any PPT algorithm $\mathcal{A}$ in the following game is negligible:

- The challenger generates $(\mathsf{msk}, \mathsf{mvk}) \leftarrow \mathsf{FS.Setup}(1^\lambda)$, and gives mvk to $\mathcal{A}$;

- The adversary is allowed to query a key generation oracle $\mathsf{O}_{\mathsf{key}}$, and a signing oracle $\mathsf{O}_{\mathsf{sign}}$, that share a dictionary indexed by tuples $(f, i) \in \mathcal{F} \times \mathbb{N}$, whose entries are signing keys: $\mathsf{sk}_i^f \leftarrow \mathsf{FS.KeyGen}(\mathsf{msk}, f)$. This dictionary keeps track of the keys that have been previously generated during the unforgeability game. The oracles are defined as follows:
  - $\mathsf{O}_{\mathsf{key}}(f, i)$:
    - If there exists an entry for the key (f, i) in the dictionary, then output the corresponding value, sk_f^i.
    - Otherwise, sample a fresh key $\mathsf{sk}_f^i \leftarrow \mathsf{FS.KeyGen}(\mathsf{msk}, f)$, add an entry $(f, i) \rightarrow \mathsf{sk}_f^i$ to the dictionary, and output sk_f^i.
  - $\mathsf{O}_{\mathsf{sign}}(f, i, m)$:
    - If there exists an entry for the key (f, i) in the dictionary, then generate a signature on $f(m)$ using this key: $\sigma \leftarrow \mathsf{FS.Sign}(f, \mathsf{sk}_f^i, m)$.
    - Otherwise, sample a fresh key $\mathsf{sk}_f^i \leftarrow \mathsf{FS.KeyGen}(\mathsf{msk}, f)$, add an entry $(f, i) \rightarrow \mathsf{sk}_f^i$ to the dictionary, and generate a signature on $f(m)$ using this key: $\sigma \leftarrow \mathsf{FS.Sign}(f, \mathsf{sk}_f^i, m)$.
- The adversary wins if it can produce (m^*, σ) such that:
  - $\mathsf{FS.Verify}(\mathsf{mvk}, m^*, \sigma) = 1$.
  - There exists no m such that $m^* = f(m)$ for any f which was sent as a query to the $\mathsf{O}_{\mathsf{key}}$ oracle.
  - There exists no (f, m) pair such that (f, m) was a query to the $\mathsf{O}_{\mathsf{sign}}$ oracle and $m^* = f(m)$.

Function privacy The scheme is function private if the advantage of any PPT algorithm $\mathcal{A}$ in the following game is negligible:

- The challenger honestly generates $(\mathsf{mvk}, \mathsf{msk}) \leftarrow \mathsf{FS.Setup}(1^\lambda)$ and gives both values to the adversary.
- The adversary $\mathcal{A}$ chooses a function f_0 and receives an (honestly generated) secret key $\mathsf{sk}_{f_0} \leftarrow \mathsf{FS.KeyGen}(\mathsf{msk}, f_1)$.
- The adversary $\mathcal{A}$ chooses a second function f_1 for which $|f_0| = |f_1|$ and receives an (honestly generated) secret key $\mathsf{sk}_{f_1} \leftarrow \mathsf{FS.KeyGen}(\mathsf{msk}, f_1)$.
- The adversary chooses a pair of messages m_0, m_1 for which $|m_0| = |m_1|$ and $f_0(m_0) = f_1(m_1)$.
- The challenger selects a random bit $b \leftarrow \{0, 1\}$ and generates a signature on the image $f_0(m_0) = f_1(m_1)$ using secret key sk_{f_b}, and gives the resulting signature $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_{f_b}, m_b)$ to the adversary.
- The adversary outputs a bit b', and wins the game if $b' = b$.

4 Multi-authority functional signature

We describe syntax and security notions of a multi-authority functional signature scheme.

Definition 2 (Multi-Authority FS (MAFS)) A decentralized multi-authority FS scheme is composed of the following algorithms:

- $\mathsf{ASetup}(id, 1^\lambda) \rightarrow (\mathsf{MVK}_{id}, \mathsf{MSK}_{id})$: Each authority $id \in \mathcal{I}$ runs the authority setup algorithm to generate its own public verification key and secret key pair, $(\mathsf{MVK}_{id}, \mathsf{MSK}_{id})$. The authority id publishes MVK_{id} and stores MSK_{id}.
- $\mathsf{KeyGen}(\mathsf{MSK}_{id}, \mathsf{UID}, U, F) \rightarrow K_{id,F}^{\mathsf{UID},U}$: When an authority id who wishes to issue a user, whose identity is UID, a secret key associated with a property U and a policy function F, it runs the key generation algorithm that outputs a secret key $K_{id,F}^{\mathsf{UID},U}$. The authority gives $K_{id,F}^{\mathsf{UID},U}$ to the user.
- $\mathsf{Sign}(\{K_{id,F}^{\mathsf{UID},U_{id}}\}_{(id,U_{id})\in\Gamma}, m) \rightarrow (F(\{(id, U_{id})\}_{(id,U_{id})\in\Gamma}, m), \sigma)$: A user signs a message m with a $|\Gamma| + 1 < O(poly(\lambda))$ variate policy function F (each component is in $\{0, 1\}^\lambda$), only if the user has obtained a set of secret keys $\{K_{id,F}^{\mathsf{UID},U_{id}} | (id, U_{id}) \in \Gamma\}$ from the authorities such that all properties included in Γ are satisfied by the policy function. Then, the user outputs the value $F(\{(id, U_{id})\}_{(id,U_{id})\in\Gamma}, m)$ and a signature of $F(\{(id, U_{id})\}_{(id,U_{id})\in\Gamma}, m)$. F takes as input up to $|\Gamma|$ properties and a message and outputs a value.
- $\mathsf{Verify}(\{\mathsf{MVK}_{id}\}_{id\in\Gamma}, m^*, \sigma) \rightarrow \{0, 1\}$: To verify a signature σ on a message m^*, using a set of public keys for relevant authorities $\{\mathsf{MVK}_{id}\}_{id\in\Gamma}$, a user runs the verification algorithm which outputs a boolean value $\mathsf{accept} := 1$ or $\mathsf{reject} := 0$.

Definition 3 (Correctness of MAFS) A MAFS scheme is said to be correct, if, for all $F \in \mathcal{F}$, $m \in \mathcal{D}_f$, authority $id \in \mathcal{I}$, $(\mathsf{MVK}_{id}, \mathsf{MSK}_{id}) \leftarrow \mathsf{ASetup}(id, 1^\lambda)$, $K_{id,F}^{\mathsf{UID},U_{id}} \leftarrow \mathsf{KeyGen}(\mathsf{MSK}_{id}, \mathsf{UID}, U_{id}, F)$ assigned by the authority id for the user UID with property U_{id} and policy function F, and $(F(\{(id, U_{id})\}_{(id,U_{id})\in\Gamma}, m), \sigma) \leftarrow$

$\mathsf{Sign}(\{K_{id,F}^{\mathsf{UID},U_{id}}\}_{(id,U_{id})\in\Gamma}, m)$ for a set of authorities $\Gamma \subset \mathcal{I}$, if it satisfies the following condition:

$$\mathsf{Verify}(\{\mathsf{MVK}_{id}\}_{id\in\Gamma}, F(\{(id, U_{id})\}_{(id,U_{id})\in\Gamma}, m), \sigma) = 1.$$

Remark 1 1. We assume that each user is allowed to have only one particular property corresponding to one authority (category) id.

2. For $|\Gamma| + 1 < O(poly(\lambda))$ variate policy function F, any different component of property is associated with a different authority. In fact, a policy function F consists of a predicate Predic which is used to indicate the set of properties that are accepted or not, and a deterministic function f which is used to modify the input message. The functionality of $F(\{(id, U_{id})\}_{(id,U_{id})\in\Gamma}, m)$ first checks whether $\mathsf{Predic}(\{(id, U_{id})\}_{(id,U_{id})\in\Gamma}) = 1$, if it is true then it outputs $f(m)$; else it outputs $\perp$.
3. For a set of properties $\{(id, U_{id})\}_{id}$ which is accepted by F, we denote it as Γ. Since a different property component in F corresponds to different authorities, we misuse the notation $id \in \Gamma$ to represent the authority that belongs to the accepted set.

Definition 4 (Unforgeability of MAFS) For any PPT adversary, we define $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{MAFS,UF}}(\lambda)$ to be the success probability in the following experiment for any security parameter λ. For ease of notation let us assume the chosen set of authorities by the adversary is $[N]$ for a polynomial N. A MAFS scheme is existentially unforgeable if the success probability of any polynomial-time adversary is negligible:

- The adversary $\mathcal{A}$ outputs $[N]$. The challenger runs the ASetup for the authorities labelled in $[N]$ and hands over the verification keys of all the authorities to $\mathcal{A}$. Adversary $\mathcal{A}$ specifies a set $S \subset [N]$ of corrupted authorities, and gets $\{\mathsf{MSK}_i\}_{i\in S}$. Let's denote $\bar{S} := [N]\backslash S$.
- The adversary is allowed to query a key generation oracle $\mathsf{O_{key}}$ and a signing oracle $\mathsf{O_{sign}}$, that share a dictionary indexed by tuples $(\mathsf{UID}, i, U_i, F)$, whose entries are keys: $K_{i,F}^{\mathsf{UID},U_i} \leftarrow \mathsf{KeyGen}(\mathsf{MSK}_i, \mathsf{UID}, U_i, F)$. This dictionary keeps track of the keys that have been previously generated during the unforgeability game. The oracles are defined as follows :
  - $\mathsf{O_{key}}(\mathsf{UID}, i, U_i, F)$: The attacker $\mathcal{A}$ submits tuples of the form $(\mathsf{UID}, i, U_i, F)$ to the challenger, where UID is a user's identity and U_i is the user's property w.r.t a non corrupted authority i. The challenger responds by giving $\mathcal{A}$ the corresponding key $K_{i,F}^{\mathsf{UID},U_i}$.
    - If there exists an entry for $(\mathsf{UID}, i, U_i, F)$ in the dictionary, then output the corresponding value $K_{i,F}^{\mathsf{UID},U_i}$.
    - Else, generate $K_{i,F}^{\mathsf{UID},U_i} \leftarrow \mathsf{KeyGen}(\mathsf{MSK}_i, \mathsf{UID}, U_i, F)$, add an entry $(\mathsf{UID}, i, U_i, F) \rightarrow K_{i,F}^{\mathsf{UID},U_i}$ to the dictionary, and output $K_{i,F}^{\mathsf{UID},U_i}$.
  - $\mathsf{O_{sign}}(\{\mathsf{UID}, i, U_i, F\}_{i\in\bar{S}}, \{K_{i,F}^{\mathsf{UID},U_i}\}_{i\in S}, m)$: The attacker selects a policy F and wishes to get the user UID's signature.
    - For any $i \in \bar{S}$, generate $K_{i,F}^{\mathsf{UID},U_i} \leftarrow \mathsf{KeyGen}(\mathsf{MSK}_i, \mathsf{UID}, U_i, F)$ and add the entry $(\mathsf{UID}, i, U_i, F) \rightarrow K_{i,F}^{\mathsf{UID},U_i}$ to the dictionary. Run $\mathsf{Sign}(\{K_{i,F}^{\mathsf{UID},U_i}\}_{i\in[N]}, m)$ to generate a tuple $(F(\{(i, U_i)\}_{i\in[N]}, m), \sigma)$ and return it back.
- At the end, $\mathcal{A}$ outputs (m^*, σ^*). We say the adversary succeeds, if
  - $\mathsf{Verify}(\{\mathsf{MVK}_i\}_{i\in[N]}, m^*, \sigma^*) = 1$.
  - $(\{\mathsf{UID}, i, U_i, F^*\}_{i\in\bar{S}}, \{K_{i,F^*}^{\mathsf{UID},U_i}\}_{i\in S}, \tilde{m})$ has never been sent as a query to the oracle $\mathsf{O_{sign}}$ for any F^* and $\tilde{m}$ such that $m^* = F^*(\{(i, U_i)\}_{i\in[N]}, \tilde{m})$.
  - There doesn't exist m such that $m^* = F^*(\{(i, U_i\ )\}_{i\in[N]}, m)$ where for any $i \in \bar{S}$, $(\mathsf{UID}, i, U_i, F^*)$ has been sent as a query to the oracle $\mathsf{O_{key}}$ and the user UID's property set $\{(i, U_i)\}_{i\in[N]}$ is accepted by F^*.

Remark 2 Since the predicate circuit Predic contained in the policy function F^* is specified over N properties (i, U_i), we say that F^* does not accept Γ_{UID} which means that when the properties corresponding to any authority $i \in \bar{S}$ are taken from Γ_{UID} no matter what the leftover $|S|$ properties of the user UID are, F^* never accepts it.

Definition 5 (Function privacy of MAFS) For any PPT adversary, we define $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{MAFS,priv}}(\lambda)$ to be the success probability in the following experiment for any security parameter λ. A MAFS scheme is function private if the success probability of any polynomial-time adversary is negligible:

- The adversary $\mathcal{A}$ outputs $[N]$. The challenger runs the setup for all N authorities labelled in $[N]$, namely generating a key pair $(\mathsf{MVK}_i, \mathsf{MSK}_i) \leftarrow \mathsf{ASetup}(i, 1^\lambda)$ and hands over the verification keys of all the authorities to $\mathcal{A}$. The adversary $\mathcal{A}$ specifies a set $S \subset [N]$ of corrupted authorities, and gets their master secret keys $\{\mathsf{MSK}_i\}_{i\in S}$. Let us denote this set of corrupted authorities as $\bar{S} := [N]\backslash S$.

- The adversary $\mathcal{A}$ chooses a policy function F_0 as well as a tuple $(\mathsf{UID}, i, U_i, F_0)$ for one specific user UID and his property U_i according to the category i, and receives an (honestly generated) key $K_{i,F_0}^{\mathsf{UID},U_i} \leftarrow \mathsf{KeyGen}(\mathsf{MSK}_i, \mathsf{UID}, U_i, F_0)$ corresponding to an uncorrupted authority $i \in \bar{S}$.
- The adversary $\mathcal{A}$ chooses a second policy function F_1 for which $|F_0| = |F_1|$, as well as a tuple $(\mathsf{UID}, i, U_i, F_1)$ for the same user UID and his property U_i according to the category i, and receives an (honestly generated) secret key $K_{i,F_1}^{\mathsf{UID},U_i} \leftarrow \mathsf{KeyGen}(\mathsf{MSK}_i, \mathsf{UID}, U_i, F_1)$ corresponding to the uncorrupted authority $i \in \bar{S}$.
- The adversary chooses a pair of messages m_0, m_1 for which $|m_0| = |m_1|$ and $F_0\left(\{(i, U_i)\}_{i\in[N]}, m_0\right) = F_1\left(\{(i, U_i)\}_{i\in[N]}, m_1\right)$.
- The challenger selects a bit $b \leftarrow \{0, 1\}$ and generates a signature on the image $m' = F_0\left(\{(i, U_i)\}_{i\in[N]}, m_0\right) = F_1\left(\{(i, U_i)\}_{i\in[N]}, m_1\right)$ using secret key $K_{i,F_b}^{\mathsf{UID},U_i}$, and gives the resulting signature $\sigma \leftarrow \mathsf{Sign}(\{K_{i,F_b}^{\mathsf{UID},U_i}\}_{i\in[N]}, m_b)$ to the adversary.
- The adversary outputs a bit b', and wins the game if $b' = b$.

Remark 3 The policy functions F_0 and F_1 should contain the same predicates, which means $F_0 = \mathsf{Predic}\|f_0$ and $F_1 = \mathsf{Predic}\|f_1$.

Definition 6 (MAFS with CRS) We say that (ASetup, KeyGen, Sign, Verify) is a MAFS scheme with CRS if Definition 2 is satisfied except that we allow all algorithm (and the adversary) to get as input also a public uniformly distributed common random string.

5 Our construction for MAFS

5.1 MAFS from standard signatures

In this section, we provide a construction of a MAFS scheme based on any standard signature scheme (*i.e.,* existentially unforgeable under chosen-message attacks). Our resulted MAFS scheme is proved to be unforgeable as the Definition 4, but not function private.

The main ideas of our construction are as follows. The master signing and public keys for each authority i $(\mathsf{MSK}_i, \mathsf{MVK}_i)$ will simply be a standard key pair for the underlying signature scheme. The signing key generated by the authority i for a user's identity UID, a property U and a function F consists of a fresh key pair $(\mathsf{sk}, \mathsf{vk})$ for the underlying signature scheme, and a signature (with respect to MVK_i) on the user's identity UID, a property U and a function F together with vk. We can regard this signature as a certificate authenticating that the owner of key vk is allowed to sign values resulted from the policy function F.

Let λ denote the security parameter and N denote the bound on the number of authorities used, while signing the message. Let Sig=(Sig.Setup, Sig.Sign, Sig.V- erify) be a signature scheme that is existentially unforgeable. We propose a MAFS scheme as follows:

- $\mathsf{ASetup}(i, 1^\lambda)$: Each authority $i \in [N]$ runs $\mathsf{Sig.Setup}$ $(1^\lambda) \rightarrow (\mathsf{msk}_i, \mathsf{mvk}_i)$ and sets $\mathsf{MSK}_i = \mathsf{msk}_i$ and $\mathsf{MVK}_i = \mathsf{mvk}_i$.
- $\mathsf{KeyGen}(\mathsf{MSK}_i, \mathsf{UID}, U, F)$:
  - Sample a signing and verification key pair for signature scheme $(\mathsf{sk}_i, \mathsf{vk}_i) \leftarrow \mathsf{Sig.Setup}(1^\lambda)$.
  - Parse $\mathsf{MSK}_i = \mathsf{msk}_i$. Run $\sigma_{\mathsf{vk}_i} \leftarrow \mathsf{Sig.Sign}$ $(\mathsf{msk}_i, \mathsf{UID}\|U\|F\|\mathsf{vk}_i)$.
  - Create a certificate $c_{i,F}^{\mathsf{UID},U} = (\mathsf{vk}_i, \sigma_{\mathsf{vk}_i}, \mathsf{UID}, U, F)$.
  - Set $K_{i,F}^{\mathsf{UID},U} = (\mathsf{sk}_i, c_{i,F}^{\mathsf{UID},U})$ and output key $K_{i,F}^{\mathsf{UID},U}$.
- $\mathsf{Sign}(\{K_{i,F}^{\mathsf{UID},U_i}\}_{i\in[N]}, m)$:
  - Parse $K_{i,F}^{\mathsf{UID},U_i} = (\mathsf{sk}_i, c_{i,F}^{\mathsf{UID},U_i})$ where $c_{i,F}^{\mathsf{UID},U_i} = (\mathsf{vk}_i, \sigma_{\mathsf{vk}_i}, \mathsf{UID}, U_i, F)$, and run $\sigma_i \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}_i, m)$ for all $i \in [N]$.
  - Compute $m^* = F(\{(i, U_i)\}_{i\in[N]}, m)$ and set $\sigma = (m, \{c_{i,F}^{\mathsf{UID},U_i}, \sigma_i\}_{i\in[N]})$.
  - Output (m^*, σ).

  Remark 4 When $\{(i, U_i)\}_{i\in[N]}$ is rejected by Predic the output of the policy function F will be $\perp$, and the corresponding signature should also be $\perp$.
- $\mathsf{Verify}(\{\mathsf{MVK}_i\}_{i\in[N]}, m^*, \sigma)$:
  - Parse $\sigma = (m, \{c_{i,F}^{\mathsf{UID},U_i}, \sigma_i\}_{i\in[N]})$ and $c_{i,F}^{\mathsf{UID},U_i} = (\mathsf{vk}_i, \sigma_{\mathsf{vk}_i}, \mathsf{UID}, U_i, F)$, and perform the following checks. If all of the checks pass output 1; otherwise output 0.
    1. $m^* = F(\{(i, U_i)\}_{i\in[N]}, m)$;
    2. $\mathsf{Sig.Verify}(\mathsf{vk}_i, m, \sigma_i) = 1$ for all $i \in [N]$;
    3. $\mathsf{Sig.Verify}(\mathsf{mvk}_i, \mathsf{UID}\|U_i\|F\|\mathsf{vk}_i, \sigma_{\mathsf{vk}_i}) = 1$ for all $i \in [N]$.

Theorem 3 *If the signature scheme Sig is existentially unforgeable under chosen message attacks, then our multi-authority functional signature scheme as specified above satisfies the unforgeability requirement defined in* Section 4.

Proof Let $\mathcal{A}_{\mathsf{MA}}$ be a PPT adversary in the unforgeability game for multi-authority functional signatures that is allowed to make query to the oracles $\mathsf{O}_{\mathsf{key}}$ and $\mathsf{O}_{\mathsf{sign}}$.

Assume $\mathcal{A}_{\mathsf{MA}}$ made polynomial numbers of queries, $Q(\lambda)$ in total to the oracles $\mathsf{O}_{\mathsf{key}}$ and $\mathsf{O}_{\mathsf{sign}}$. We construct an adversary $\mathcal{B}_{\mathsf{Sig}}$ with $\mathcal{A}_{\mathsf{MA}}$ as a subroutine such that, if $\mathcal{A}_{\mathsf{MA}}$ wins in the unforgeability game for MAFS with non-negligible probability, then $\mathcal{B}_{\mathsf{Sig}}$ breaks the unforgeability game of underlying signature scheme, which is assumed to be unforgeable.

For $\mathcal{A}_{\mathsf{MA}}$ to win unforgeability game of MAFS, it should output a message signature pair $(m^*, \sigma^* = (\tilde{m}, \{c_{i,F^*}^{\mathsf{UID},U_i}\}_{i\in[N]}, \{\sigma_i^*\}_{i\in[N]}))$ where $c_{i,F^*}^{\mathsf{UID},U_i} = (\mathsf{vk}_i, \sigma_{\mathsf{vk}_i}, \mathsf{UID}, U_i, F^*)$ such that:

- For each $i \in \bar{S}$, σ_i^* is a valid signature of $\tilde{m}$ under the verification key vk_i;
- For each $i \in \bar{S}$, σ_{vk_i} is a valid signature of $\mathsf{UID}\|U_i\|F\|\mathsf{vk}_i$ under mvk_i;
- $m^* = F^*(\{(i, U_i)\}_{i\in[N]}, \tilde{m})$;
- $\mathcal{A}_{\mathsf{MA}}$ has not sent a query of form $(\{\mathsf{UID}, i, U_i, F^*\}_{i\in\bar{S}}, \{K_{i,F^*}^{\mathsf{UID},U_i}\}_{i\in S}, \tilde{m})$ to the signing oracle $\mathsf{O}_{\mathsf{sign}}$.
- $\mathcal{A}_{\mathsf{MA}}$ has not sent queries $(\mathsf{UID}, i, U_i, F^*)$ for all $i \in \bar{S}$, to the oracle $\mathsf{O}_{\mathsf{key}}$ such that the user's UID property set $\{(i, U_i)\}_{i\in[N]}$ is accepted by F^* and m^* is in the range of the function F^*.

Assume there are $N-1$ authorities that are corrupted, while the uncorrupted one's identity is i^*. There are two cases for such a forgery $\left(m^*, \sigma^* = (\tilde{m}, \{c_{i^*,F^*}^{\mathsf{UID},U_{i^*}}, \sigma_{i^*}^*\}, \{c_{i,F^*}^{\mathsf{UID},U_i}, \sigma_i^*\}_{i\in[N]\setminus i^*})\right)$ where $c_{i,F^*}^{\mathsf{UID},U_i} = (\mathsf{vk}_i, \sigma_{\mathsf{vk}_i}, \mathsf{UID}, U_i, F^*)$ for all $i \in [N]$:

- **Type I forgery:** The tuples $(\mathsf{vk}_{i^*}, \mathsf{UID}, U_{i^*}, F^*)$ satisfy that $\mathsf{UID}\|U_{i^*}\|F^*\|\mathsf{vk}_{i^*}$ has *not* been signed under mvk_{i^*} for the queries of $\mathcal{A}_{\mathsf{MA}}$ to both the oracle $\mathsf{O}_{\mathsf{sign}}$ and $\mathsf{O}_{\mathsf{key}}$.
- **Type II forgery:** The tuples $(\mathsf{vk}_{i^*}, \mathsf{UID}, U_{i^*}, F^*)$ satisfy that $\mathsf{UID}\|U_{i^*}\|F^*\|\mathsf{vk}_{i^*}$ *has* been signed under mvk_{i^*} during the queries of $\mathcal{A}_{\mathsf{MA}}$ to both the oracle $\mathsf{O}_{\mathsf{sign}}$ and $\mathsf{O}_{\mathsf{key}}$.

Here we assume that all the queries of $\mathcal{A}_{\mathsf{MA}}$ send to the oracles $\mathsf{O}_{\mathsf{sign}}$ are proceeded by the oracle $\mathsf{O}_{\mathsf{key}}$ to generate a signing key as intermediate steps.

We now describe the signature adversary $\mathcal{B}_{\mathsf{Sig}}$. In the unforgeability game for the standard signature scheme, given the verification key $\mathsf{vk}_{\mathsf{Sig}}$, and access to a signing oracle $\mathsf{O}_{\mathsf{Reg}_{\mathsf{sig}}}$, $\mathcal{B}_{\mathsf{Sig}}$ wins the unforgeability game if he successfully outputs a forgery, *i.e.,* a signature for a message that was not queried to $\mathsf{O}_{\mathsf{Reg}_{\mathsf{sig}}}$. In order to play the role of the challenger interacting with $\mathcal{A}_{\mathsf{MA}}$ in the security game for MAFS, $\mathcal{B}_{\mathsf{Sig}}$ must simulate the $\mathsf{O}_{\mathsf{key}}$ and $\mathsf{O}_{\mathsf{sign}}$ oracles. $\mathcal{B}_{\mathsf{Sig}}$ flips a coin b, and proceeds as following.

Case 1: $b = 1$ $\mathcal{B}_{\mathsf{Sig}}$ guesses that $\mathcal{A}_{\mathsf{MA}}$ will produce a **Type I** forgery:

First $\mathcal{B}_{\mathsf{Sig}}$ sets $\mathsf{vk}_{\mathsf{Sig}}$ as the master verification key for the authority i^* in the MAFS security game, namely $\mathsf{mvk}_{i^*} := \mathsf{vk}_{\mathsf{Sig}}$ and sends it to $\mathcal{A}_{\mathsf{MA}}$. For all of the other corrupted authorities, $\mathcal{B}_{\mathsf{Sig}}$ behaves honestly. To simulate the $\mathsf{O}_{\mathsf{key}}$ and $\mathsf{O}_{\mathsf{sign}}$ oracles, $\mathcal{B}_{\mathsf{Sig}}$ maintains a dictionary indexed by the tuples $(\mathsf{UID}, i^*, U_{i^*}^j, F)$, whose entries are signing keys that are generated by the authority i^* for the user UID with property $U_{i^*}^j$ and policy function F. $\mathcal{B}_{\mathsf{Sig}}$ answers the queries issued by $\mathcal{A}_{\mathsf{MA}}$ as follows:

- $\mathsf{O}_{\mathsf{key}}(\mathsf{UID}, i^*, U_{i^*}^j, F)$:
  - If there exists an entry for the tuple $(\mathsf{UID}, i^*, U_{i^*}^j, F)$ in the dictionary, then output the corresponding value $K_{i^*,F}^{\mathsf{UID},U_{i^*}^j}$.
  - Otherwise, run $(\mathsf{sk}_j, \mathsf{vk}_j) \leftarrow \mathsf{Sig.Setup}(1^\lambda)$ and send $\mathsf{UID}\|U_{i^*}^j\|F\|\mathsf{vk}_j$ to its own signing oracle to get $\sigma_{\mathsf{vk}_{i^*}^j} \leftarrow \mathsf{O}_{\mathsf{Reg}_{\mathsf{sig}}}(\mathsf{UID}\|U_{i^*}^j\|F\|\mathsf{vk}_j)$. Set $K_{i^*,F}^{\mathsf{UID},U_{i^*}^j} = (\mathsf{sk}_j, c_{i^*,F}^{\mathsf{UID},U_{i^*}^j})$ where $c_{i^*,F}^{\mathsf{UID},U_{i^*}^j} = (\mathsf{vk}_j, \sigma_{\mathsf{vk}_{i^*}^j}, \mathsf{UID}, U_{i^*}^j, F)$, add it to the dictionary, and output it.
- $\mathsf{O}_{\mathsf{sign}}(\{\mathsf{UID}, i^*, U_{i^*}, F\}, \{K_{i,F}^{\mathsf{UID},U_i}\}_{i\in[N]\setminus i^*}, m)$:
  - If there exists an entry for the tuple $\{\mathsf{UID}, i^*, U_{i^*}, F\}$ in the dictionary, namely the key $K_{i^*,F}^{\mathsf{UID},U_{i^*}}$, then it generates $(F(\{(i, U_i)\}_{i\in[N]}, m), \sigma) \leftarrow \mathsf{Sign}(\{K_{i,F}^{\mathsf{UID},U_i}\}_{i\in[N]}, m)$ and outputs the result.
  - Otherwise, $\mathcal{B}_{\mathsf{Sig}}$ samples a new pair of key for the signature scheme, $(\mathsf{sk}_{i^*}, \mathsf{vk}_{i^*}) \leftarrow \mathsf{Sig.Setup}(1^\lambda)$, and obtains $\sigma_{\mathsf{vk}_{i^*}} \leftarrow \mathsf{O}_{\mathsf{Reg}_{\mathsf{sig}}}(\mathsf{UID}\|U_{i^*}\|F\|\mathsf{vk}_{i^*})$ fr- om its own signing oracle. It sets $K_{i^*,F}^{\mathsf{UID},U_{i^*}} := (\mathsf{sk}_{i^*}, c_{i^*,F}^{\mathsf{UID},U_{i^*}})$ where $c_{i^*,F}^{\mathsf{UID},U_{i^*}} = (\mathsf{vk}_{i^*}, \sigma_{\mathsf{vk}_{i^*}}, \mathsf{UID}, U_{i^*}, F)$, and adds it to the dictionary. For each $i \in [N]$, it then computes $\sigma_i \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}_i, m)$ and $m^* = F(\{(i, U_i)\}_{i\in[N]}, m)$. It sets $\sigma := (m, \{c_{i,F}^{\mathsf{UID},U_i}\}_{i\in[N]}, \{\sigma_i\}_{i\in[N]})$, then outputs (m^*, σ).

Eventually, $\mathcal{A}_{\mathsf{MA}}$ outputs a message signature forgery $\left(m^*, \sigma^* = (\tilde{m}, \{c_{i^*,F^*}^{\mathsf{UID},U_{i^*}}, \sigma_{i^*}^*\}, \{c_{i,F^*}^{\mathsf{UID},U_i}, \sigma_i^*\}_{i\in[N]\setminus i^*})\right)$ where $c_{i,F^*}^{\mathsf{UID},U_i} = (\mathsf{vk}_i, \sigma_{\mathsf{vk}_i}, \mathsf{UID}, U_i, F^*)$ for each $i \in [N]$. Since the forgery satisfies $\mathsf{Sig.Verify}(\mathsf{mvk}_{i^*}, \mathsf{UID}\|U_{i^*}\|F^*\|\mathsf{vk}_{i^*}, \sigma_{\mathsf{vk}_{i^*}}^*) = 1$, $\mathcal{B}_{\mathsf{Sig}}$ outputs $(\mathsf{UID}\|U_{i^*}\|F^*\|\mathsf{vk}_{i^*}, \sigma_{\mathsf{vk}_{i^*}}^*)$ as its message-forgery pair in the security game for the standard signature scheme.

Case 2: $b = 0$ $\mathcal{B}_{\mathsf{Sig}}$ guesses that $\mathcal{A}_{\mathsf{MA}}$ will produce a **Type II** forgery:

For each authority $i \in [N]$, $\mathcal{B}_{\mathsf{Sig}}$ generates $\mathsf{Sig.Setup}(1^\lambda) \rightarrow (\mathsf{msk}_i, \mathsf{mvk}_i)$ and sets $\mathsf{MSK}_i = \mathsf{msk}_i$ and $\mathsf{MVK}_i = \mathsf{mvk}_i$. $\mathcal{B}_{\mathsf{Sig}}$ forwards mvk_i of all the authorities together with $\{\mathsf{msk}_i\}$ for a specified set $S = \{i : i \in [N] \backslash i^*\}$ of corrupted authorities to $\mathcal{A}_{\mathsf{MA}}$. $\mathcal{B}_{\mathsf{Sig}}$ chooses a random value q^* between 1 and $Q(\lambda)$ as the index of MAFS's signing queries which the challenge verification key will be positioned on. We use $\mathsf{numkeys}$ to denote the number of signing keys that has been generated. We initialize $\mathsf{numkeys} = 0$. As before, $\mathcal{B}_{\mathsf{Sig}}$ maintains a dictionary indexed by the tuples $(\mathsf{UID}, i^*, U_{i^*}^j, F)$, whose entries are signing keys that are generated by the authority i^* for the user UID with property $U_{i^*}^j$ and policy function F. $\mathcal{B}_{\mathsf{Sig}}$ answers the queries issued by $\mathcal{A}_{\mathsf{MA}}$ as follows:

- $\mathsf{O_{key}}(\mathsf{UID}, i^*, U_{i^*}^j, F)$:
  - If there exists an entry for tuple $(\mathsf{UID}, i^*, U_{i^*}^j, F)$ in the dictionary, with value Chal, abort;
  - If there exists an entry for tuple $(\mathsf{UID}, i^*, U_{i^*}^j, F)$ in the dictionary and its value is not Chal, then output the corresponding value $K_{i^*,F}^{\mathsf{UID},U_{i^*}^j}$.
  - Otherwise, generate $(\mathsf{sk}_j, \mathsf{vk}_j) \leftarrow \mathsf{Sig.Setup}(1^\lambda)$ and $\sigma_{\mathsf{vk}_{i^*}^j} \leftarrow \mathsf{Sig.Sign}(\mathsf{msk}_{i^*}, \mathsf{UID} \| U_{i^*}{}^j \| F \| \mathsf{vk}_j)$. Set $K_{i^*,F}^{\mathsf{UID},U_{i^*}^j} = (\mathsf{sk}_j, c_{i^*,F}^{\mathsf{UID},U_{i^*}^j})$ where $c_{i^*,F}^{\mathsf{UID},U_{i^*}^j} = (\mathsf{vk}_j, \sigma_{\mathsf{vk}_{i^*}^j}, \mathsf{UID}, U_{i^*}^j, F)$, add it to the dictionary, and output it.
- $\mathsf{O_{sign}}(\{\mathsf{UID}, i^*, U_{i^*}, F\}, \{K_{i,F}^{\mathsf{UID},U_i}\}_{i \in [N] \backslash i^*}, m)$:
  - If there exists an entry for tuple $\{\mathsf{UID}, i^*, U_{i^*}, F\}$ in the dictionary, namely the key $K_{i^*,F}^{\mathsf{UID},U_{i^*}}$, then it runs $(F(\{(i, U_i)\}_{i \in [N]}, m), \sigma) \leftarrow \mathsf{Sign}(\{K_{i,F}^{\mathsf{UID},U_i}\}_{i \in [N]}, m)$ and outputs the result.
  - If there is no $\{\mathsf{UID}, i^*, U_{i^*}, F\}$ entry in the dictionary, and $\mathsf{numkeys} \neq q^*$, then $\mathcal{B}_{\mathsf{Sig}}$ generates a new key pair of signature scheme, $(\mathsf{sk}_{i^*}, \mathsf{vk}_{i^*}) \leftarrow \mathsf{Sig.Setup}(1^\lambda)$, signs $\mathsf{UID} \| U_{i^*} \| F \| \mathsf{vk}_{i^*}$ under msk_{i^*}: $\sigma_{\mathsf{vk}_{i^*}} \leftarrow \mathsf{Sig.Sign}(\mathsf{msk}_{i^*}, \mathsf{UID} \| U_{i^*} \| F \| \mathsf{vk}_{i^*})$. $\mathcal{B}_{\mathsf{Sig}}$ sets $K_{i^*,F}^{\mathsf{UID},U_{i^*}} := (\mathsf{sk}_{i^*}, c_{i^*,F}^{\mathsf{UID},U_{i^*}})$ where $c_{i^*,F}^{\mathsf{UID},U_{i^*}} = (\mathsf{vk}_{i^*}, \sigma_{\mathsf{vk}_{i^*}}, \mathsf{UID}, U_{i^*}, F)$, and adds it to the dictionary. For each $i \in [N]$, it computes $\sigma_i \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}_i, m)$ and $m^* = F(\{(i, U_i)\}_{i \in [N]}, m)$. Set $\sigma = (m, \{c_{i,F}^{\mathsf{UID},U_i}\}_{i \in [N]}, \{\sigma_i\}_{i \in [N]})$, then output (m^*, σ). $\mathsf{numkeys}$ is then incremented.
  - If there is no $\{\mathsf{UID}, i^*, U_{i^*}, F\}$ entry in the dictionary and $\mathsf{numkeys} = q^*$, or if the $\{\mathsf{UID}, i^*, U_{i^*}, F\}$ entry in the dictionary is set to Chal, then $\mathcal{B}_{\mathsf{Sig}}$ sets $\mathsf{vk}_{i^*} := \mathsf{vk}_{\mathsf{Sig}}$, signs $\mathsf{UID} \| U_{i^*} \| F \| \mathsf{vk}_{\mathsf{Sig}}$ under msk_{i^*}: $\sigma_{\mathsf{vk}_{i^*}} \leftarrow \mathsf{Sig.Sign}(\mathsf{msk}_{i^*}, \mathsf{UID} \| U_{i^*} \| F \| \mathsf{vk}_{\mathsf{Sig}})$, and queries its oracle for a signature of m under $\mathsf{vk}_{\mathsf{Sig}}$, $\sigma_{i^*} \leftarrow \mathsf{O}_{\mathsf{Reg}_{\mathsf{sig}}}(m)$. Set $c_{i^*,F}^{\mathsf{UID},U_{i^*}} = (\mathsf{vk}_{i^*}, \sigma_{\mathsf{vk}_{i^*}}, \mathsf{UID}, U_{i^*}, F)$. For each $i \in [N] \backslash i^*$ it then runs $\sigma_i \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}_i, m)$ and $m^* = F(\{(i, U_i)\}_{i \in [N]}, m)$. Set $\sigma = (m, \{c_{i,F}^{\mathsf{UID},U_i}\}_{i \in [N]}, \{\sigma_i\}_{i \in [N]})$, then output (m^*, σ). If there is no $\{\mathsf{UID}, i^*, U_{i^*}, F\}$ entry in the dictionary, $\mathcal{B}_{\mathsf{Sig}}$ sets it to Chal. And $\mathsf{numkeys}$ is then incremented.

If $\mathcal{B}_{\mathsf{Sig}}$ does not abort, $\mathcal{A}_{\mathsf{MA}}$ will output a signature $\left(m^*, \sigma^* = (\tilde{m}, \{c_{i^*,F^*}^{\mathsf{UID},U_{i^*}}, \sigma_{i^*}^*\}, \{c_{i,F^*}^{\mathsf{UID},U_i}, \sigma_i^*\}_{i \in [N] \backslash i^*})\right)$ where $c_{i,F^*}^{\mathsf{UID},U_i} = (\mathsf{vk}_i, \sigma_{\mathsf{vk}_i}, \mathsf{UID}, U_i, F^*)$ for every $i \in [N]$. $\mathcal{B}_{\mathsf{Sig}}$ outputs $(\tilde{m}, \sigma_{i^*}^*)$ as its forgery for the standard signature scheme under $\mathsf{vk}_{\mathsf{Sig}}$.

We will now argue that if $\mathcal{A}_{\mathsf{MA}}$ forges in the MAFS scheme with non-negligible probability then $\mathcal{B}_{\mathsf{Sig}}$ wins the unforgeability game for the standard signature scheme with non-negligible probability. We note that as long as $\mathcal{A}_{\mathsf{MA}}$ doesn't query the $\mathsf{O_{key}}$ oracle for the secret key corresponding to the embedded $\mathsf{vk}_{\mathsf{Sig}}$ challenge, then $\mathcal{B}_{\mathsf{Sig}}$ perfectly simulates the $\mathsf{O_{key}}$ and $\mathsf{O_{sign}}$ oracle.

Now, if $\mathcal{A}_{\mathsf{MA}}$ produces a **Type I** forgery, then this forgery must consist of a signature on a new message $\mathsf{UID} \| U_{i^*} \| F^* \| \mathsf{vk}_{i^*}$ that was not ever signed under the i^* authority's master verification key mvk_{i^*} during the queries of $\mathcal{A}_{\mathsf{MA}}$ to both the oracle $\mathsf{O_{key}}$ and $\mathsf{O_{sign}}$, which means that $\mathcal{A}_{\mathsf{MA}}$ produces a forgery on a new message $\mathsf{UID} \| U_{i^*} \| F^* \| \mathsf{vk}_{i^*}$ that $\mathcal{B}_{\mathsf{Sig}}$ did not query to his signature oracle, which yields a forgery for the standard signature scheme.

If $\mathcal{A}_{\mathsf{MA}}$ produces a **Type II** forgery, by definition the corresponding $\mathsf{UID} \| U_{i^*} \| F^* \| \mathsf{vk}_{i^*}$ should already have been signed under the i^* authority's master verification key mvk_{i^*} during the queries of $\mathcal{A}_{\mathsf{MA}}$ to both the oracle $\mathsf{O_{key}}$ and $\mathsf{O_{sign}}$. We declare that the tuple $(\mathsf{UID}, i^*, U_{i^*}, F^*)$ cannot be queried to $\mathsf{O_{key}}$ by $\mathcal{A}_{\mathsf{MA}}$, since if it is then given the responded signing key, producing a signature under this signing key is not a valid forgery in the MAFS scheme. Therefore, the tuple $(\mathsf{UID}, i^*, U_{i^*}, F^*)$ must then have been issued as a query to $\mathsf{O_{sign}}$. Namely, the verification key vk_{i^*} must be freshly generated for a query of form $\mathsf{O_{sign}}(\{\mathsf{UID}, i^*, U_{i^*}, F^*\}, \{K_{i,F^*}^{\mathsf{UID},U_i}\}_{i \in [N] \backslash i^*}, m)$ for which no entry under index $(\mathsf{UID}, i^*, U_{i^*}, F^*)$ previously existed, and then the pair $\mathsf{UID} \| U_{i^*} \| F^* \| \mathsf{vk}_{i^*}$ was signed under the master

signing key of identity i^*. Note that if $\mathcal{A}_{\mathsf{MA}}$ produces a Type II forgery and $\mathcal{B}_{\mathsf{Sig}}$ correctly guessed q^* where his challenge is embedded in as well as $\mathcal{B}_{\mathsf{Sig}}$ does not abort, the forgery produced by $\mathcal{A}_{\mathsf{MA}}$ must consist of a signature on a new message $\tilde{m}$ under $\mathsf{vk}_{\mathsf{Sig}}$, for a $\tilde{m}$ that $\mathcal{B}_{\mathsf{Sig}}$ has not queried from his own signing oracle, and therefore such a signature also constitutes a forgery for the standard signature scheme.

We note that, if $\mathcal{B}_{\mathsf{Sig}}$ does abort, it only can occur when $\mathcal{B}_{\mathsf{Sig}}$ did not guess the correct q^*, since another possible condition of abort occurrence is when $\mathcal{A}_{\mathsf{MA}}$ made a query as $\mathsf{O}_{\mathsf{key}}(\mathsf{UID}, i^*, U_{i^*}, F^*)$, which is not allowed since if the adversary has queried the $\mathsf{O}_{\mathsf{key}}(\mathsf{UID}, i^*, U_{i^*}, F^*)$, no message in the range of F^* would be considered a forgery in the MAFS game.

Thus, if $\mathcal{A}_{\mathsf{MA}}$ produces a forgery in MAFS scheme with non-negligible probability $\frac{1}{Poly(\lambda)}$, then $\mathcal{B}_{\mathsf{Sig}}$ successfully forges in the underlying signature scheme with non-negligible probability $\frac{1}{2Q(\lambda)Poly(\lambda)}$, which contradicts with the assumption that Sig is existentially unforgeable. We conclude that the MAFS scheme as specified above is unforgeable as defined in Section 4. □

5.2 Function-private MAFS from SNARKs

While the construction of MAFS above does not provide function privacy property, in this section we show how to obtain a MAFS scheme satisfying function privacy in the common reference string (CRS) model by employing the building blocks of a FS without function privacy guarantee and a succinct non-interactive argument of knowledge (SNARK). In the CRS model there is an additional setup algorithm, which takes as input the security parameter and outputs the CRS for the system.

As mentioned in the Remark 1, we considered the policy function F consisting of a predicate Predic which is to indicate the set of properties are accepted or not, and a deterministic function f which is to work on the input message. The functionality of policy function $F(\{(id, U_{id})\}_{(id,U_{id})\in\Gamma}, m)$ is first to check $\mathsf{Predic}(\{(id, U_{id})\}_{(id,U_{id})\in\Gamma}) = 1$, if it is true then output $f(m)$; else output $\perp$. In this section, we represent the policy function F as $\mathsf{Predic}\|f$, which means for input $(\{(id, U_{id})\}_{(id,U_{id})\in\Gamma}, m)$, $\mathsf{Predic}\|f$ firstly compute $\mathsf{Predic}(\{(id, U_{id})\}_{(id,U_{id})\in\Gamma})$, if the predicate circuit accepts then output $f(m)$, otherwise output $\perp$.

Let $\mathsf{Sig} = (\mathsf{Sig.Setup}, \mathsf{Sig.Sign}, \mathsf{Sig.Verify})$ be a signature scheme that is existentially unforgeable under chosen message attacks, $\mathsf{FS} = (\mathsf{FS.Setup}, \mathsf{FS.KeyGen}, \mathsf{FS.Sign}, \mathsf{FS.Verify})$ be a FS scheme satisfying the unforgeability but not function privacy, and $\mathsf{SNARK}=\big(\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify}, \mathcal{S} = (\mathcal{S}^{\mathsf{crs}}, \mathcal{S}^{\mathsf{Proof}}), \mathsf{E} = (\mathsf{E}_1, {}_2)\big)$ be a zero knowledge (ZK) SNARK system for the following NP language L:

$$L = \{(M, \{\mathsf{mvk}\}_{i\in[N]}) | \text{for } \forall i \in [N],\ \exists \sigma_i\ s.t.\ \mathsf{FS.Verify}(\mathsf{mvk}_i, M, \sigma_i) = 1\}.$$

By using Sig, FS and SNARK we construct a new MAFS scheme as follows, which also satisfies function privacy.

- $\mathsf{Setup}(1^\lambda)$:
  - Choose a CRS for the ZK-SNARK: $\mathsf{crs} \leftarrow \mathsf{SNARK.Gen}(1^\lambda)$.
- $\mathsf{ASetup}(i, 1^\lambda)$:
  - Authority $i \in [N]$ samples a pair of keys for Sig: $(\mathsf{sk}_i, \mathsf{vk}_i) \leftarrow \mathsf{Sig.Setup}(1^\lambda)$.
  - Set the master secret key $\mathsf{MSK}_i = \mathsf{sk}_i$, and the master verification key $\mathsf{MVK}_i = \mathsf{vk}_i$.
- $\mathsf{KeyGen}(\mathsf{MSK}_i, \mathsf{UID}, U, \mathsf{Predic}\|f)$:
  - Parse $\mathsf{MSK}_i = \mathsf{sk}_i$ and sample a master secret and verification key for the functional signature scheme $\mathsf{FS.Setup}(1^\lambda) \rightarrow (\mathsf{msk}_i, \mathsf{mvk}_i)$.
  - Compute $\eta_i \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}_i, \mathsf{UID}\|U\|\mathsf{Predic}\|\mathsf{mvk}_i)$ and $\mathsf{sk}_i^f \leftarrow \mathsf{FS.KeyGen}(\mathsf{msk}_i, f)$.
  - Create the certificate $c_{i,\mathsf{Predic}}^{\mathsf{UID},U} = (\eta_i, \mathsf{UID}, U, \mathsf{Predic}, \mathsf{mvk}_i)$.
  - Set $K_{i,\mathsf{Predic}\|f}^{\mathsf{UID},U} = (\mathsf{sk}_i^f, c_{i,\mathsf{Predic}}^{\mathsf{UID},U})$ and output a key $K_{i,\mathsf{Predic}\|f}^{\mathsf{UID},U}$.
- $\mathsf{Sign}(\mathsf{crs}, \{K_{i,\mathsf{Predic}\|f}^{\mathsf{UID},U_i}\}_{i\in[N]}, m)$:
  - Parse $K_{i,\mathsf{Predic}\|f}^{\mathsf{UID},U_i} = (\mathsf{sk}_i^f, c_{i,\mathsf{Predic}}^{\mathsf{UID},U_i})$ where $c_{i,\mathsf{Predic}}^{\mathsf{UID},U_i} = (\eta_i, \mathsf{UID}, U, \mathsf{Predic}, \mathsf{mvk}_i)$.
  - For all $i \in [N]$ if UID are the same and $\mathsf{Predic}(\{(i, U_i)\}_{i\in[N]}) = 1$, then run $(f(m), \sigma_i) \rightarrow \mathsf{FS.Sign}(\mathsf{sk}_i^f, m)$.
  - Run $\pi_i \leftarrow \mathsf{SNARK.Prove}\big((f(m), \{\mathsf{mvk}_i\}_{i\in[N]}), \{\sigma_i\}_{i\in[N]}, \mathsf{crs}\big)$, a ZK-SNARK that $(f(m), \{\mathsf{mvk}_i\}_{i\in[N]}) \in L$, where L is defined as above.
  - Set $m^* = f(m)$ and $\sigma = (\{c_{i,\mathsf{Predic}}^{\mathsf{UID},U_i}\}_{i\in[N]}, \pi)$.
  - Output (m^*, σ).
- $\mathsf{Verify}(\mathsf{crs}, \{\mathsf{MVK}_i\}_{i\in[N]}, m^*, \sigma)$:
  - Parse $\mathsf{MVK}_i = \mathsf{vk}_i$ and $\sigma = (\{c_{i,\mathsf{Predic}}^{\mathsf{UID},U_i}\}_{i\in[N]}, \pi)$ where $c_{i,\mathsf{Predic}}^{\mathsf{UID},U_i} = (\eta_i, \mathsf{UID}, U, \mathsf{Predic}, \mathsf{mvk}_i)$,

and perform the following checks. If all of the checks pass output 1; otherwise output 0.

1. UID are the same and $\mathsf{Predic}(\{(i, U_i)\}_{i\in[N]}) = 1$;
2. $\mathsf{Sig.Verify}(\mathsf{vk}_i, \mathsf{UID}\|U\|\mathsf{Predic}\|\mathsf{mvk}_i, \eta_i) = 1$for all $i \in [N]$;
3. $\mathsf{SNARK.Verify}(\mathsf{crs}, (\{\mathsf{mvk}_i\}_{i\in[N]}, m^*), \pi) = 1$.

Theorem 4 *Assume the existence of functional signature scheme FS supporting the class $\mathcal{F}$ of polynomial-sized circuits that satisfies unforgeability but not function privacy, Sig is an existentially unforgeable signature scheme, and SNARK be an adaptive zero-knowledge SNARK system for NP. Then, there exists a MAFS scheme as specified above, which satisfies both the unforgeability (Definition 4) and function privacy (Definition 5).*

Proof We first prove that our scheme satisfies *unforgeability*. Let us fix a PPT adversary $\mathcal{A}_{\mathsf{MA}}$. We will use $\mathcal{A}_{\mathsf{MA}}$ to construct an adversary $\mathcal{B}$ such that, if $\mathcal{A}_{\mathsf{MA}}$ wins in the unforgeability game for multi-authority functional signatures with non-negligible probability, then $\mathcal{B}$ either breaks the underlying signature scheme or functional signature scheme, which are assumed to be secure.

$\mathcal{B}$ interacts with $\mathcal{A}_{\mathsf{MA}}$, playing the role of the challenger in the security game for the multi-authority functional signature scheme. $\mathcal{B}$ generates $(\mathsf{crs}, \mathsf{trap}) \leftarrow \mathsf{E}_1(1^\lambda)$, a simulated CRS for the ZK-SNARK, together with a trapdoor, and forwards crs to $\mathcal{A}_{\mathsf{MA}}$ as the CRS. $\mathcal{B}$ also must simulate the $\mathsf{O}_{\mathsf{key}}$ and $\mathsf{O}_{\mathsf{sign}}$ oracles. $\mathcal{B}$ flips a coin b, indicating his guess for the type of forgery $\mathcal{A}_{\mathsf{MA}}$ will produce, and places his challenge accordingly. Since $\mathcal{B}$ behaves almost the same as what he does in **Case 1** and **Case 2** of the proof in Theorem 3. We omit the details here. Eventually $\mathcal{A}_{\mathsf{MA}}$ outputs a forgery $\left(m^*, \sigma^* = (\{c^{\mathsf{UID},U_i}_{i,\mathsf{Predic}^*}\}_{i\in[N]}, \pi^*)\right)$ where $c^{\mathsf{UID},U_i}_{i,\mathsf{Predic}^*} = (\eta^*_i, \mathsf{UID}, U_i, \mathsf{Pre\text{-}dic}^*, \mathsf{mvk}_i)$ such that:

- π^* is a valid proof the statement $(\{\mathsf{mvk}_i\}_{i\in[N]}, m^*) \in L$ under CRS;
- For each $i \in \bar{S}$, η^*_i is a valid signature of $\mathsf{UID}\|U_i\|\mathsf{Pre\text{-}dic}^*\|\mathsf{mvk}_i$ under vk_i;
- $\mathcal{A}_{\mathsf{MA}}$ has not sent a query of form $\left(\{\mathsf{UID}, i, U_i, \mathsf{Predic}^*\|\tilde{f}\}_{i\in\bar{S}}, \{K^{\mathsf{UID},U_i}_{i,\mathsf{Predic}^*\|\tilde{f}}\}_{i\in S}, \tilde{m}\right)$ to the signing oracle $\mathsf{O}_{\mathsf{sign}}$ for any $\tilde{f}$ and $\tilde{m}$ such that $\tilde{f}(\tilde{m}) = m^*$.
- $\mathcal{A}_{\mathsf{MA}}$ has not sent queries $(\mathsf{UID}, i, U_i, \mathsf{Predic}^*\|\tilde{f})$ for all $i \in \bar{S}$, to the oracle $\mathsf{O}_{\mathsf{key}}$ such that the user UID's property set $\{(i, U_i)\}_{i\in[N]}$ is accepted by Predic^* and for any $\tilde{f}$ that has m^* in its range.

In **Case 1**, $\mathcal{B}$ finally outputs $(\mathsf{UID}\|U_{i^*}\|\mathsf{Predic}^*\|\mathsf{mvk}_{i^*}, \eta^*_{i^*})$ as its message-forgery pair for the standard signature scheme. Whereas, in **Case 2**, $\mathcal{B}$ runs the extractor $\mathsf{E}_2(\mathsf{crs}, \mathsf{trap}, (m^*, \{\mathsf{mvk}_i\}_{i\in[N]}), \pi^*)$ to recover a witness $w = \{\sigma^*_i\}_{i\in[N]}$ such that $\mathsf{FS.Verify}(\mathsf{mvk}_i, m^*, \sigma^*_i) = 1$ for all $i \in [N]$. $\mathcal{B}$ then outputs $(m^*, \sigma^*_{i^*})$ as a forgery for the FS scheme under $\mathsf{mvk}_{\mathsf{FS}}$.

We will now argue that if $\mathcal{A}_{\mathsf{MA}}$ forges the MAFS scheme with non-negligible probability, then $\mathcal{B}$ either wins the unforgeability game for the standard signature scheme or the functional signature with non-negligible probability.

Hybrid 0 The real MAFS challenge experiment. Namely, the CRS is generated in the honest manner $\mathsf{crs} \leftarrow \mathsf{SNARK.Gen}(1^\lambda)$. Let Forge_0 denote the probability of the adversary producing a valid forgery in the MAFS scheme in this game.

Hybrid 1 The same experiment as Hybrid 0, except that the CRS is generated using the extractor, $(\mathsf{crs}, \mathsf{trap}) \leftarrow \mathsf{E}_1(1^\lambda)$. Let Forge_1 denote the probability of the adversary producing a valid forgery in a MAFS scheme in this game. Following directly from the fact that the CRS values generated via the standard algorithm Gen and those generated by the extractor algorithm E_1 are statistically close, we have $\mathsf{Forge}_1 - \mathsf{Forge}_0 \leq negl(\lambda)$.

Hybrid 2 The same experiment as Hybrid 1. And at the end we apply the ZK-SNARK extraction algorithm on the adversary's forgery $\sigma^* = (\{c^{\mathsf{UID},U_i}_{i,\mathsf{Predic}^*}\}_{i\in[N]}, \pi^*)$ (on message m^*) in the MAFS scheme: *i.e.,* $\mathsf{E}_2(\mathsf{crs}, \mathsf{trap}, (m^*, \{\mathsf{mvk}_i\}_{i\in[N]}), \pi^*) \rightarrow \{\sigma^*_i\}_{i\in[N]}$. Let Forge_2 denote the probability that $\sigma^*_{i^*}$ is a valid signature in the underlying FS scheme on a message m^*. Following directly from the extraction property of the ZK-SNARK system, we have $\mathsf{Forge}_2 - \mathsf{Forge}_1 \leq negl(\lambda)$.

Since Forge_2 is precisely the probability that the adversary $\mathcal{B}$ constructed above produces a successful forgery in the unforgeability game for FS, due to the unforgeability of FS, we have $\mathsf{Forge}_2 \leq negl(\lambda)$.

To prove that our scheme satisfies *function privacy*, we show that if there exists a PPT adversary $\mathcal{A}_{\mathsf{priv}}$ wining the function privacy game for MAFS with non-negligible advantage, then there exists an PPT adversary breaking the zero knowledge security game of the ZK-SNARK. More specifically, consider the following two hybrid games:

Hybrid 0 The real function privacy game. Namely, the CRS for the ZK-SNARK system is honestly generated $\mathsf{crs} \leftarrow \mathsf{SNARK.Gen}(1^\lambda)$. To generate the challenge signature on message m_b for a random bit $b \in \{0,1\}$, the challenger first generates a signature on m_b under the underlying FS scheme: $(f_b(m_b), \sigma_i) \rightarrow \mathsf{FS.Sign}(\mathsf{sk}_i^{f_b}, m_b)$ for each $i \in [N]$ and then honestly generates a proof $\pi \leftarrow \mathsf{SNARK.Prove}\big((f_b(m_b), \{\mathsf{mvk}_i\}_{i\in[N]}), \{\sigma_i\}_{i\in[N]}, \mathsf{crs}\big)$.

Hybrid 1 The same game as Hybrid 0 except that the proof in the challenge signature is generated in the simulated manner. Namely, the CRS is generated using the simulator algorithm $(\mathsf{crs}, \mathsf{trap}) \leftarrow \mathcal{S}^{\mathsf{crs}}(1^\lambda)$. The challenger then uses the simulator to generate the proof $\pi \leftarrow \mathcal{S}^{\mathsf{Proof}}((\mathsf{crs}, \mathsf{trap}), (m^*, \{\mathsf{mvk}_i\}_{i\in[N]}))$, where $m^* = f_0(m_0) = f_1(m_1)$.

Following directly from the zero knowledge property of the ZK-SNARK system, the difference between the advantage of the adversary $\mathcal{A}_{\mathsf{priv}}$ in guessing the bit b in Hybrid 0 and Hybrid 1 is negligible. Note that the view of $\mathcal{A}_{\mathsf{priv}}$ in Hybrid 1 is independent with the bit b, hence any PPT adversary cannot correctly guess the bit b with non-negligible advantage. □

6 Conclusion

The emergence of the IoT, has led to millions of sensors and smart devices sending sensitive data across communication networks. Thus, it is of utmost importance to address the challenge of guaranteeing the integrity and authentication of the transmitted information. To achieve that we propose a generalized signature scheme with a fine grained access control, that can provide significant advantages to the digital world, in application settings where the authenticity of shared information is crucial e.g., smart healthcare, smart homes, smart cities. In this paper, we introduce the definition of *multi-authority functional signatures* (MAFSs) for general policy functions, and provide a general transformation from a standard signature scheme to a MAFS scheme satisfying unforgeability property but not function privacy. Given a non-function-private functional signature and a SNARK we provide a way to build a function private MAFS scheme.

Our MAFS can properly solve the problem we depicted in the IoT related delivery operation world and successfully provides both businesses and customers greater visibility and trackability of their products and parcels than ever before. In particular, by using our MAFS the records of all transactions during the en-route of delivery is able to be kept and renewed. Moreover, we allow the multiple authorities to be involved in the execution of the transactions, meanwhile ensuring that the users are able to verify all the transactions occurred in the network. We believe our MAFS can be applied to other IoT related application scenarios.

Acknowledgments This work has been partially supported by a STINT Sweden-Japan 150 Anniversary Grant and the VR project PRECIS.

Appendix A: Signature scheme

Definition 7 A signature scheme with message space $\mathcal{M}(\lambda)$, signature key space $\mathcal{SK}(\lambda)$ and verification key space $\mathcal{VK}(\lambda)$ consists of the PPT algorithms $(\mathsf{SIG.Setup}, \mathsf{SIG.Sign}, \mathsf{SIG.Verify})$:

- **Key generation.** $\mathsf{SIG.Setup}$ is a randomized algorithm that takes as input the security parameter 1^λ and outputs the signing key $\mathsf{sk} \in \mathcal{SK}$ and the verification key $\mathsf{vk} \in \mathcal{VK}$.
- **Signature generation.** $\mathsf{SIG.Sign}$ takes as input the signing key $\mathsf{sk} \in \mathcal{SK}$ and a message $m \in \mathcal{M}$ and outputs a signature σ.
- **Verification.** $\mathsf{SIG.Verify}$ takes as input a verification key $\mathsf{vk} \in \mathcal{VK}$, a message $m \in \mathcal{M}$ and a signature σ and outputs either 0 or 1.

Correctness For all $\lambda \in \mathbb{N}$, $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{SIG.Setup}(1^\lambda)$, messages $m \in \mathcal{M}(\lambda)$, we require that

$$\mathsf{SIG.Verify}(\mathsf{vk}, m, \mathsf{SIG.Sign}(\mathsf{sk}, m)) = 1.$$

We say that a signature scheme $\mathsf{SIG} = (\mathsf{SIG.Setup}, \mathsf{SIG.Sign}, \mathsf{SIG.Verify})$ is existentially unforgeable under adaptively chosen message attacks if

$$\Pr[\mathsf{Exp}^{\mathsf{uf\text{-}cma}}_{\mathsf{SIG},\mathcal{A}}(\lambda) = 1] \leq negl(\lambda)$$

for some negligible function $negl()$ and for all PPT attackers $\mathcal{A}$, where $\mathsf{Exp}^{\mathsf{uf\text{-}cma}}_{\mathsf{SIG},\mathcal{A}}(\lambda)$ is the following experiment with the scheme SIG and an attacker $\mathcal{A}$:

1. $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{SIG.Setup}(1^\lambda)$.
2. $(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk},\cdot)}(1^\lambda, \mathsf{vk})$.
   If $\mathsf{SIG.Verify}(\mathsf{vk}, m^*, \sigma^*) = 1$ and m^* was not queried to the $\mathsf{Sign}(\mathsf{sk}, \cdot)$ oracle, then return 1, else return 0.

Appendix B: Succinct Non-Interactive Arguments (SNARGs)

We employ the definition of SNARGs in [2].

Definition 8 $\Pi = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify})$ is a succinct non-intera- ctive argument for a language $L \in$ NP with witness relation R if it satisfies the following properties:

- Completeness: For all x, w such that $R(x, w) = 1$, and for all strings $\mathsf{crs} \leftarrow \mathsf{Gen}(1^k)$, $\mathsf{Verify}(crs, x, \mathsf{Prove}(x, w, \mathsf{crs})) = 1$.
- Adaptive Soundness: There exists a negligible function $\mu(k)$, such that, for all PPT adversaries $\mathcal{A}$, if $\mathsf{crs} \leftarrow \mathsf{Gen}(1^k)$ is sampled uniformly at random, then the probability that $\mathcal{A}(\mathsf{crs})$ will output a pair (x, π) such that $x \notin L$ and yet $\mathsf{Verify}(\mathsf{crs}, x, \pi) = 1$, is at most $\mu(k)$.
- Succinctness: There exists an universal polynomial $p(k)$ that does not depend on the relation R, such that $\forall x, w \; s.t R(x, w) = 1, \mathsf{crs} \leftarrow \mathsf{Gen}(1^k), \pi \leftarrow \mathsf{Prove}(x, w, \mathsf{crs})$,

  $$|\pi| \leq p(k + \log R)$$

  where R denotes the runtime of the relation associated with language L.

Definition 9 A SNARG $\Pi = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify})$ is a succinct non-interactive argument of knowledge(SNARK) for a language $L \in$ NP with witness relation R if there exists a negligible function $\mu(\cdot)$ such that, for all PPT provers P^*, there exists a PPT algorithm $\mathsf{E}_{P^*} = (\mathsf{E}^1_{P^*}, \mathsf{E}^2_{P^*})$ such that for every adversary A,

$$|\Pr[\mathcal{A}(\mathsf{crs}) \to 1 | \mathsf{crs} \leftarrow \mathsf{Gen}(1^k)] - \Pr[\mathcal{A}(\mathsf{crs}) \to 1 | (\mathsf{crs}, \mathsf{trap}) \leftarrow \mathsf{E}^1_{P^*}(1^k)]| = \mu(k),$$

and,

$$\Pr[P^*(\mathsf{crs}) \to (x, \pi) \text{ and } \mathsf{E}^2_{P^*}(\mathsf{crs}, \mathsf{trap}, x, \pi) \to w^* \; s.t. \; \mathsf{Verify}(\mathsf{crs}, x, \pi) \to 1 \text{ and } (x, w^*) \notin R] = \mu(k).$$

where the probabilities are taken over $(\mathsf{crs}, \mathsf{trap}) \leftarrow \mathsf{E}^1_{P^*}(1^k)$, and over the random coin tosses of the extractor algorithm $\mathsf{E}^2_{P^*}$.

Definition 10 A SNARK $\Pi = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify})$ is a zero-knowledge SNARK for a language $L \in$ NP with witness relation R if there exist PPT algorithms $S = (\mathcal{S}^{\mathsf{crs}}, \mathcal{S}^{\mathsf{Proof}})$ satisfying the following property:

- Adaptive Zero-Knowledge: For all PPT adversaries $\mathcal{A}$,

  $$|\Pr[\mathrm{Exp}_{\mathcal{A}}(k) \to 1] - \Pr[\mathrm{Exp}^S_{\mathcal{A}}(k) \to 1]| \leq \mu(k),$$

  where the experiments $\mathrm{Exp}_{\mathcal{A}}(k)$ and $\mathrm{Exp}^S_{\mathcal{A}}(k)$ are defined as follows:

$\mathrm{Exp}_{\mathcal{A}}(k)$:	$\mathrm{Exp}^S_{\mathcal{A}}(k)$:
$\mathsf{crs} \leftarrow \mathsf{Gen}(1^k)$;	$(\mathsf{crs}, \mathsf{trap}) \leftarrow \mathcal{S}^{\mathsf{crs}}(1^k)$
Return $\mathcal{A}^{\mathsf{Prove}}(\mathsf{crs}, \cdot, \cdot)(\mathsf{crs})$	Return $\mathcal{A}^{S'(\mathsf{crs}, \mathsf{trap}, \cdot, \cdot)}(\mathsf{crs})$,

  where $S'(\mathsf{crs}, \mathsf{trap}, x, w) = \mathcal{S}^{\mathsf{Proof}}(\mathsf{crs}, \mathsf{trap}, x)$.

References

1. Amendola S, Lodato R, Manzari S, Occhiuzzi C, Marrocco G (2014) Rfid technology for iot-based personal healthcare in smart spaces. IEEE Internet Things J 1(2):144–152
2. Boyle E, Goldwasser S, Ivan I (2014) Functional signatures and pseudorandom functions. In: Proceedings of PKC 2014, pp 501–519. Springer
3. Chandran N, Goyal V, Jain A, Sahai A (2015) Functional encryption: decentralised and delegatable. IACR Cryptology ePrint Archive 2015:1017
4. Cocchia A (2014) Smart and digital city: a systematic literature review. In: Smart city, pp 13–43. Springer
5. Diffie W, Hellman M (2006) New directions in cryptography. IEEE Trans Inf Theor 22(6):644–654
6. Du K-k, Wang Z-l, Mi H (2013) Human machine interactive system on smart home of iot. J China Univ Posts Telecommun 20:96–99
7. Jie Y, Pei JY, Jun L, Yun G, Wei X (2013) Smart home system based on iot technologies. In: 2013 fifth international conference on computational and information sciences (ICCIS), pp 1789–1791. IEEE
8. Jin J, Gubbi J, Marusic S, Palaniswami M (2014) An information framework for creating a smart city through internet of things. IEEE Internet Things J 1(2):112–121
9. Maji HK, Prabhakaran M, Rosulek M (2011) Attribute-based signatures. In: Cryptographers' track at the RSA conference, pp 376–392. Springer
10. Okamoto T, Takashima K (2011) Efficient attribute-based signatures for non-monotone predicates in the standard model. In: Proceedings of PKC
11. Okamoto T, Takashima K (2010) Fully secure functional encryption with general relations from the decisional linear assumption. In: Annual cryptology conference, pp 191–208. Springer
12. Okamoto T, Takashima K (2013) Decentralized attribute-based signatures. In: Proceedings of PKC 2013, pp 125–142