Highly Controlled, Fine-grained Delegation of Signing Capabilities

Michael Backes
MPI-SWS
Saarland University
Germany

Sebastian Meiser
Saarland University
Germany

Dominique Schröder
Saarland University
Germany

June 20, 2013

Abstract

Delegation of signing rights is a central problem in security. Whereas delegating by giving power of attorney is well studied and digitally realized via delegatable anonymous credentials, directly delegating signing possibilities without the need for an external logic, can be done via malleable signature schemes. However, the existing schemes do not allow for privacy preserving, fine-grained malleability and they do not allow for a controlled way of further delegating the malleability. We bridge this gap by introducing *delegatable functional signatures* (DFS).

We present the first construction of a DFS scheme. This construction is based on standard cryptographic primitives and shows that our strong unforgeability and privacy notions are achievable for arbitrary efficiently computable forms of malleability and delegatability.

1 Introduction

Delegation of signing rights is a central problem in security. Under delegation of signing rights we understand that one entity grants the rights to sign new documents or to modify already existing signatures in its own name to a second entity. This entity can in turn again delegate the rights further and so on.

An approach to delegating signing capabilities that is commonly used in everyday life is to give power of attorney. The signer authorizes a third person to sign or modify certain documents in his name. This approach is digitally realized via the versatile and well studied notion of delegatable (anonymous) credentials. Valid credentials attest that a given entity has the rights and capabilities it claims to have. If credentials are anonymous, this means that only the levels and types of delegation have to be revealed, but not the intermediate entities.

An orthogonal approach is to make signatures *malleable*. The signer gives a third person, a so called evaluator, a number of tools to modify signed documents by evaluating functions on them without voiding the signature. For coarse-grained capabilities one could give blank paper with signatures to another person or leave blank spaces in contracts. However, fine-grained malleability is only possible in the digital world, as the concept of being able to apply only specific functions, e.g., to be able to fill a blank space with one name but not with another, cannot be enforced on paper.

There are two main differences between issuing credentials and providing malleability. The first one is the privacy guarantee that the respective approach can offer. Authorizing a third party via credential (even anonymously) means revealing the structure of the authorization chain, whereas a

malleable signature can be modified without revealing that anyone but the signer was involved. In practice, this is of particular interest, because it hides the internal operations in a company.

The second difference is, that classically, malleability cannot be *further delegated* in a controlled manner whereas the owner of a credential can delegate (parts of) his power to another person. This is a true shortcoming of malleable signature schemes. A malleable signature scheme that allowed not only for fine-grained malleability, but also for fine-grained delegation of power would be useful in many scenarios.

Imagine that the signer of a message can specify *for this signature*, which evaluator is allowed to apply which functions without voiding the validity of the signature (Malleability). Moreover, the signer can specify *for this signature* how the malleability can be delegated (Delegatability). This way, the signer would retain full control over the signature.

An appealing property of a signature primitive that allows for controlled malleability and delegation is that it still is a signature primitive. Any system that already uses digital signatures can be extended with such a scheme by simply replacing the primitives. This means that such a powerful signature scheme can seamlessly be integrated into already existing architectures.

However, so far, there is no signature primitive that captures both fine-grained malleability and a controlled form of delegation. In this paper we close this gap by introducing *delegatable functional signatures*.

1.1 Applications

This primitive has many applications and we sketch only two simple ones that illustrate its broad application spectrum.

Signed Code. Consider one of the mayor players in the development of enterprise software solutions, such as Oracle or SAP. A simple methodology to guarantee the authenticity of their code, is to let Oracle sign (at least critical parts of) its software. Although this idea intuitively makes sense, it is not practical, not even conceptually. The reason is that the software is extremely general: it can be applied to the business processes of banks, car industry, telecommunication and many more. In fact, whenever a company decides to use such a system, it needs to be configured and customized for each customer. Since this is often done by third party companies or freelancers that obtained the right to deploy the system, we envision a malleable signature for which Oracle specifies how the signed code may be changed. After the software is deployed, it has to be maintained (e.g. modified slightly, to cope with small changes in the company's business process). In some cases, these enterprise solutions provide interfaces where the customers can adapt specific parts of the system on their own. Thus, a controlled part of the malleability has to be delegatable. A regular signature scheme is simply not sufficient, because of the delegation steps and also because of the individual adjustment of each system.

With our primitive, however, the authenticity of the software can be verified, even if third parties deploy it and even if the customer changes dedicated parts on his own. Our primitive allows Oracle to specify a delegation policy that dictates how the third party company is allowed to adjust their software without losing the validity of the signature. Since the signing and delegation process is completely opaque to the outside, any customer of the company (that uses the Oracle solution), would only learn that they are working with a verified version of Oracle enterprise solutions.

Digital Coupons. Coupon websites, such as Groupon, are currently up-and-coming. The business

model is that companies want to increase their visibility by issuing digitally signed coupons for cheaper services. Here, we envision a scenario where these companies apply a malleable signature scheme that allows Groupon to modify the coupons in a certain way such as, e.g., to apply their corporate design or to turn non-valid certificates into valid certificates if a certain number of orders has been reached. Furthermore, Groupon delegates a different malleability to each (new) customer who then is allowed to modify this coupon even further without voiding the validity. For example, the customer may wish to hide that he purchased the (cheap) coupon through a website and not directly from the company (which might be more expensive) or the coupon may allow the customer to personalize an item (e.g., modify text on a T-Shirt) or choose between some alternatives. For example, the purchaser of a coupon for a hotel may get to choose between a wellness or sport activity. The customer might also want to change the name on the coupon and send it to a friend as a gift without revealing the alternatives to this friend. The challenge in this setting arises from the highly dynamic content, as new products and possibilities are developed regularly that might allow new options for personalization. In particular, the company wants a dynamic scheme in which they can specify per coupon to which degree its signature should be malleable.

With our primitive, the company can specify for every coupon what Groupon is allowed to do in terms of modification and how the customers can modify the coupon without losing the validity of the signature.

1.2 Our Contribution

We introduce *delegatable functional signatures* (DFS) as a new primitive that supports highly controlled, fine-grained delegation of signing capabilities to designated third parties. First we will explain our primitive, then we will briefly present our contribution in terms of a new security model for signatures and finally we will show that constructing a DFS that fulfills the definitions of our security model is possible.

Delegatable Functional Signatures. The basic idea of delegatable functional signatures is that the signer can specify *for each message individually* how a designated party can perform the following two tasks.

Malleability. The signer defines how the evaluator can modify the message without voiding its validity. We formalize this intuition by defining a functionality $\mathcal{F}(f, \alpha, m)$. We explain the parameters with the following example.

Example 1. *Suppose that Alice wants to allow Bob to censor parts of a message that she signs. Her choice of f describes the subsets of the message that can be censored by Bob without harming the validity of the signature. Bob chooses the parts of the message he wishes to censor by choosing the corresponding value for α, and he derives a signature on $m' = \mathcal{F}(f, \alpha, m)$.*

Delegatability. The signer defines how the evaluator can delegate signing capabilities to third parties. This is formalized by a functionality $\mathcal{G}(g, \beta, pk, f)$ and we explain it by continuing the example from above.

Example 2. *Suppose that Alice wants to restrict an evaluator Bob in delegating his capabilities. The evaluator Bob owns a secret key corresponding to some public key pk_B and can apply the function*

f to censor parts parts of the message m without voiding the validity of its signature (see above). Now, suppose that Alice wants to allow Bob to only delegate the possibility to censor the first or second part of the message and let us describe the corresponding functions with $f_{\text{1st.}}$ and $f_{\text{2nd.}}$. Alice chooses a value g such that $\mathcal{G}(g, \beta, pk, f) = f_{\text{1st.}}$ for a specific value $\beta = \beta_1$ and $\mathcal{G}(g, \beta, pk, f) = f_{\text{2nd.}}$ otherwise, for all $pk \neq pk_B$. Now Bob can choose β to either delegate $f_{\text{1st.}}$ or $f_{\text{2nd.}}$.

Note that we do not need to define explicitly how signing capabilities for fresh messages can be delegated with this primitive. If Alice wants to give Bob the capability to sign certain messages in her name, she can simply generate a new (empty) message and use f and g to specify which capabilities Bob has.

Security Model for DFS. Defining a proper security model turns out to be highly non-trivial due to the flexibility of our primitive. In this paper we suggest new definitions for unforgeability and privacy. The basic idea behind our unforgeability notions is, that an adversary should not be able to forge signatures for new messages, aside from those that have been allowed by the signer. The basic idea behind our privacy notions is that it should be hard to distinguish if a message-signature pair has been computed by the signer or derived from another signature by an evaluator.

In both cases we present three different security notions for DFS schemes: The weakest one, unforgeability/privacy against outsider attackers, holds only for attackers that do not have access to the private key of an evaluator. The second one, unforgeability/privacy against insider attackers, assumes that an evaluator is malicious and possesses a honestly generated evaluator key. The third one, unforgeability/privacy against strong insider attackers assumes a malicious evaluator that might generate its own keys.

Constructing a DFS scheme. Finally, we provide the first construction of a DFS scheme. This construction is based on standard cryptographic primitives, such as digital signature schemes, public key encryption, and non-interactive zero-knowledge proofs. Our scheme shows that our strong definitions for unforgeability and privacy are achievable for arbitrary, efficiently computable, choices of $\mathcal{F}$ and $\mathcal{G}$.

1.3 Related Work

(Delegatable) Anonymous Credentials. In anonymous credential systems users can prove the possession of a credential (that may grant a power of attorney) without revealing their identity. We view this very successful line of research as orthogonal to our work: Credentials can be applied on top of a signature scheme in order to prove properties that are specified in an external logic. In fact, one could combine delegatable functional signatures with credentials in order to partially leak the delegation chain, while allowing to issue or modify credentials in an anonymous but controlled way.

Anonymous credential systems have been investigated extensively, e.g., [12, 13, 17, 18, 6, 19, 30, 16, 21]. The main difference between delegatable anonymous credential schemes, such as [5, 1], and our approach is that delegation is done by extending the proof chain (and thus leaking information about the chain). Restricting the properties of the issuer in a credential system has been considered in [4]. However, they only focus on access control proofs and their proof chain is necessarily visible, whereas our primitive allows for privacy-preserving schemes.

Malleable Signature Schemes. Allowing a small degree of malleability for digital signatures has been considered in many different ways. First we give an overview over schemes that do not consider a special secret key for modifying signatures, which means that everyone with access to the correct public key and one or more valid message-signature pairs can derive new valid message-signature pairs. There are schemes that allow for redacting signatures [29, 27, 28, 14] that allow for deriving valid signatures on parts (or subsets) of the message m. There are schemes that allow for deriving subset and union relations on signed sets [27], linearly homomorphic signature schemes [23, 11] and schemes that allow for evaluating polynomial functions [10]. Another line of research considers a general approach for computing on authenticated data [2].

However, all approaches mentioned above only consider static functions or predicates (one function or predicate for every scheme) and leave the signer little room for bounding a class of functions to a specific message. As the signatures can be modified by everyone with access to public information, they do not allow for a concept of controlled delegation.

Sanitizable signature schemes [3, 15] extend the concept of malleable signatures by a new secret key sk_{San} for the evaluator. Only a party in possession of this key can modify signatures. In general, this primitive allows the signer to specify which blocks of the message can be changed, without restricting the possible content. However, they do not consider delegation and they do not allow for computing arbitrary functions on signed data.

Anonymous Proxy Signatures [24] consider delegation of signing rights in a specific context. For example, the delegator may choose a subset of signing rights for the tasks of quoting. Their notion of privacy makes sure that all delegators remain anonymous. The main difference with our work is that they only allow delegation on the basis of the keys and that they do not support restricting further delegation, whereas we support restricting delegation capabilities depending on each message.

Constructing delegatable anonymous credentials out of malleable signatures has very recently been investigated by Chase et al. [20]. However, the authors only consider one fixed set of allowable transformations per malleable signature scheme and do not allow the signer to restrict malleability (per message) nor does their system allow any way to restrict delegation.

1.4 Outline

As a warm up we first present *functional signature schemes* (without delegation) in Section 2 and introduce unforgeability and privacy notions for functional signature schemes. In Section 3 we present a formal definition for *delegatable functional signatures*, together with slightly modified notions of unforgeability and privacy. In Section 5 we give the first construction of a delegatable functional signature scheme for n-times delegation and prove it to be secure and privacy friendly.

2 Functional Signatures

As a warm up for our main result, and also to make our definitions more accessible, we introduce functional signature schemes (FSS). A functional signature scheme for a specific functionality $\mathcal{F}$ allows the delegation of signing capabilities for arbitrary functions that belong to $\mathcal{F}$ to a designated third party, called the evaluator. A functional signature scheme already suffices for instantiating many known signature schemes like rerandomizable signatures, identity based signatures, and certain forms of malleable signatures like, e.g., sanitizable and redactable signatures (namely all that

$\underline{\text{PROC INITIALIZE}(\lambda):}$
$(pp, msk) \leftarrow \mathsf{Setup}(\lambda)$
$(sk_{sig}, pk_{sig}) \leftarrow \mathsf{KGen}_{sig}(pp, msk)$
$(pk_{\mathcal{F}}, sk_{\mathcal{F}}) \leftarrow \mathsf{KGen}_{\mathcal{F}}(pp, msk)$
store $(pp, msk, sk_{sig}, pk_{sig}, sk_{\mathcal{F}}, pk_{\mathcal{F}})$
set $\mathcal{K}_{\mathcal{C}} := \{(sk_{\mathcal{F}}, pk_{\mathcal{F}})\}$
set $\mathcal{K}_{\mathcal{A}} := \emptyset, \mathcal{Q} := \emptyset$
output $(pp, pk_{sig}, pk_{\mathcal{F}})$

$\underline{\text{PROC QUERY[KGENS]}():}$
retrieve $(sk_{\mathcal{F}}, pk_{\mathcal{F}})$
set $\mathcal{K}_{\mathcal{A}} := K_{\mathcal{A}} \cup \{pk_{\mathcal{F}}\}$
output $sk_{\mathcal{F}}$

$\underline{\text{PROC QUERY[SIGN]}(pk^*_{\mathcal{F}}, f, m):}$
if $pk^*_{\mathcal{F}} \notin \mathcal{K}_{\mathcal{C}}$ output $\perp$
retrieve (pp, sk_{sig})
$\sigma \leftarrow \mathsf{Sig}(pp, sk_{sig}, pk^*_{\mathcal{F}}, f, m)$
set $\mathcal{Q} := \mathcal{Q} \cup (f, m, pk^*_{\mathcal{F}}, \sigma)$
output σ

$\underline{\text{PROC QUERY[EVAL]}(\alpha, m, \sigma):}$
retrieve $(pp, sk_{\mathcal{F}}, pk_{sig})$
$\sigma' \leftarrow \mathsf{Eval}_{\mathcal{F}}(pp, sk_{\mathcal{F}}, pk_{sig}, \alpha, m, \sigma)$
if $\sigma' \neq \perp$
  extract f from σ using $sk_{\mathcal{F}}$
  set $\mathcal{Q} := \mathcal{Q} \cup \{f, \mathcal{F}(\lambda, f, \alpha, m), pk_{\mathcal{F}}, \sigma'\}$
output σ'

$\underline{\text{PROC FINALIZE}(m^*, \sigma^*, pk^*_{\mathcal{F}}):}$
if $(\cdot, m^*, \cdot, \cdot) \in \mathcal{Q}$ output 0
if $(f, m, pk'_{\mathcal{F}}, \cdot) \in \mathcal{Q}$ s.t.
$pk'_{\mathcal{F}} \in \mathcal{K}_{\mathcal{A}} \wedge m^* \in \mathcal{F}^*(\lambda, f, m)$
  output 0
retrieve (pp, pk_{sig})
$b \leftarrow \mathsf{Vf}(pp, pk_{sig}, pk^*_{\mathcal{F}}, m^*, \sigma^*)$
output b

$\underline{\text{PROC QUERY[REGKEY]}(sk^*_{\mathcal{F}}, pk^*_{\mathcal{F}}):}$
set $\mathcal{K}_{\mathcal{C}} := \mathcal{K}_{\mathcal{C}} \cup \{(sk^*_{\mathcal{F}}, pk^*_{\mathcal{F}})\}$
set $\mathcal{K}_{\mathcal{A}} := \mathcal{K}_{\mathcal{A}} \cup \{pk^*_{\mathcal{F}}\}$

Figure 1: The unforgeability game for functional signature schemes.

work on one signature). Because of space constraints we only sketch how to construct an identity based signature scheme in Appendix **??** and omit the other implications. Intuitively, in a functional signature scheme the signer signs a message m and defines the class of functions $f \subseteq \mathcal{F}$ that can be evaluated on m. The evaluator holds a secret key $sk_{\mathcal{F}}$ and can compute valid signatures on messages $m' = f_\alpha(m)$, where α defines the evaluator's choice of a specific function $f_\alpha \in f$.

2.1 Formal Definition

A functionality $\mathcal{F} : \mathbb{N} \times \mathcal{P}_f \times \mathcal{P}_\alpha \times \mathcal{M} \to \mathcal{M} \cup \{\perp\}$ is a deterministic polynomial-time algorithm that takes as input a security parameter λ, a function parameter f, a parameter α, and a payload m. When signing a message m via an algorithm Sig, the signer can choose the public key of an evaluator and a function parameter f. Thereafter the chosen evaluator can choose a value for α and call an algorithm $\mathsf{Eval}_{\mathcal{F}}$ to compute valid signatures on messages $\mathcal{F}(\lambda, f, \alpha, m)$.

Definition 1 (Functional Signature)**.** *A* functional signature scheme FSS *is a tuple of efficient algorithms* $\mathsf{FSS} = (\mathsf{Setup}, \mathsf{KGen}_{sig}, \mathsf{KGen}_{\mathcal{F}}, \mathsf{Sig}, \mathsf{Eval}_{\mathcal{F}}, \mathsf{Vf})$ *with the following interfaces:*

$\mathsf{Setup}(\lambda)$: *The setup algorithm* Setup *outputs some public parameters pp and a master secret key msk.*

$\mathsf{KGen}_{sig}(pp, msk)$: *The signature key generation algorithm outputs a secret signing key* sk_{sig} *and a public signing key* pk_{sig}*.*

$\mathsf{KGen}_{\mathcal{F}}(pp, msk)$: *The evaluation key generation algorithm* $\mathsf{KGen}_{\mathcal{F}}$ *outputs a secret evaluator key* $sk_{\mathcal{F}}$ *and a public evaluator key* $pk_{\mathcal{F}}$*.*

$\mathsf{Sig}(pp, sk_{sig}, pk_{\mathcal{F}}, f, m)$: *The signing algorithm* Sig *outputs a signature* σ *on* m*, to which the owner of they secret key* $sk_{\mathcal{F}}$ *corresponding to* $pk_{\mathcal{F}}$ *can apply functions from the class* f *(or an error symbol* $\perp$*).*

$\mathsf{Eval}_{\mathcal{F}}(\boldsymbol{pp}, \boldsymbol{sk}_{\mathcal{F}}, \boldsymbol{pk}_{sig}, \alpha, m, \sigma)$**:** *The evaluation algorithm* $\mathsf{Eval}_{\mathcal{F}}$ *outputs a derived signature on the message* $\mathcal{F}(\lambda, f, \alpha, m)$ *(or an error symbol* $\perp$*).*

$\mathsf{Vf}(\boldsymbol{pp}, \boldsymbol{pk}_{sig}, \boldsymbol{pk}_{\mathcal{F}}, m, \sigma)$**:** *The verification algorithm* Vf *outputs a bit* $b \in \{0,1\}$ *depending on the validity of the signature.*

We consider a functional signature scheme to be correct for a functionality $\mathcal{F}$ if the verification succeeds for all honestly generated signatures and for all valid modifications of honestly generated signatures.

Definition 2. *(Correctness). We say that a functional signature scheme* $\mathsf{FSS} = (\mathsf{Setup}, \mathsf{KGen}_{sig}, \mathsf{KGen}_{\mathcal{F}}, \mathsf{Sig}, \mathsf{Eval}_{\mathcal{F}}, \mathsf{Vf})$ *is* $\mathcal{F}$-correct *for a functionality* $\mathcal{F}$*, if for all* $\lambda \in \mathbb{N}$*, all* $m \in \mathcal{M}, \alpha \in \mathcal{P}_\alpha$ *and for all* $(pp, msk) \in [\mathsf{Setup}(\lambda)]$ *and all* $(sk_{sig}, pk_{sig}) \in [\mathsf{KGen}_{sig}(pp, msk)]$*, and all* $(sk_{\mathcal{F}}, pk_{\mathcal{F}}) \in [\mathsf{KGen}_{\mathcal{F}}(pp, msk)]$*, the following two properties hold:*

1) $$\mathsf{Vf}(pp, pk_{sig}, pk_{\mathcal{F}}, m, \mathsf{Sig}(pp, sk_{sig}, pk_{\mathcal{F}}, f, m)) = 1$$

2) $$\mathcal{F}(\lambda, f, \alpha, m) \neq \perp \wedge \sigma \leftarrow \mathsf{Sig}(pp, sk_{sig}, pk_{\mathcal{F}}, f, m) \wedge$$
$$\sigma' \leftarrow \mathsf{Eval}_{\mathcal{F}}(pp, sk_{\mathcal{F}}, pk_{sig}, \alpha, m, \sigma) \Rightarrow \mathsf{Vf}(pp, pk_{sig}, pk_{\mathcal{F}}, \mathcal{F}(\lambda, f, \alpha, m), \sigma') = 1.$$

2.2 Security of Functional Signature Schemes

In this section, we define unforgeability and sketch privacy. In the case of unforgeability we distinguish between outsider and insider attacks: In an *outsider* attack, the adversary only knows both public keys, whereas an adversary launching an *insider* attack knows the private key of the evaluator. Informally we say that a functional signature scheme provides privacy if it is computationally hard to distinguish whether a signature was created by the signer or whether it was modified by the evaluator. In the following subsections we discuss the intuition behind each definition in more detail and provide formal definitions.

For the following security definitions we follow the concept of Bellare and Rogaway in defining the security notions as a game $G(\mathsf{FSS}, \mathcal{F}, \mathcal{A}, \lambda)$ [9]. Each game G behaves as follows: First, it invokes an algorithm Initialize with the security parameter and sends its output to the algorithm $\mathcal{A}$. Then it simulates $\mathcal{A}$ with oracle access to all specified algorithms Query[x] that are defined for G. It also allows $\mathcal{A}$ to call the algorithm Finalize once and ends as soon as Finalize is called. The output of Finalize is a boolean value and is also the output of G. Note that G is allowed to maintain state. We say that $\mathcal{A}$ "wins" the game if $G(\mathsf{FSS}, \mathcal{F}, \mathcal{A}, \lambda) = 1$.

2.2.1 Unforgeability

Intuitively, a functional signature scheme is unforgeable, if no adversary $\mathcal{A}$ is able to compute a fresh message-signature pair that is not trivially deducible from the knowledge of $\mathcal{A}$. In the case of regular signature schemes this means that the attacker needs to compute a signature on a fresh message. The situation here is more complex, because our signatures are malleable. Therefore, we present three different unforgeability notions:

Unforgeability against outsider attacks. We model the outsider as an active adversary that knows the public keys $(pk_{sig}, pk_{\mathcal{F}})$ and has oracle access to both the Sig method and the $\mathsf{Eval}_{\mathcal{F}}$

algorithms. Our definition of unforgeability against outsider attacks resembles the traditional definition of unforgeability for signature schemes [25], where the adversary knows the public-key and has access to a signing oracle.

Unforgeability against (weak/strong) insider attacks. Our second definition considers the case where the evaluator is malicious. We define two different notions depending on the capabilities of the adversary. That is, our first definition that we call unforgeability against weak insider attacks (or just insider attacks), gives the attacker access to a honestly generated private key $sk_{\mathcal{F}}$. The second notion allows the adversary to choose its own private key(s) maliciously. We refer to this notion as unforgeability against *strong* insider attacks.

An insider that receives a signature σ on m that is delegated to him, can trivially deduce some signatures. If σ allows the capability f, this insider can trivially produce signatures on all messages that he can produce by (repeatedly) applying $\mathcal{F}$ on f and m with (possibly different) values for α. We call the set of all messages that can be derived from m via the functionality $\mathcal{F}$ and the capability f the *transitive closure*.

Definition 3. *(Transitive closure of functionality $\mathcal{F}$). Given a functionality $\mathcal{F}$, we define the n-transitive closure $\mathcal{F}^n$ of $\mathcal{F}$ on parameters (λ, f, m) recursively as follows:*

- *For $n = 0$, $\mathcal{F}^0(\lambda, f, m) := \{m\}$.*
- *For $n > 0$, $\mathcal{F}^n(\lambda, f, m) := \{m\} \bigcup_{\alpha} \mathcal{F}^{n-1}(\lambda, f, \mathcal{F}(\lambda, f, \alpha, m))$.*

We define the transitive closure *$\mathcal{F}^*$ of $\mathcal{F}$ on parameters (λ, f, m) as $\mathcal{F}^*(\lambda, f, m) := \bigcup_{i=0}^{\infty} \mathcal{F}^i(\lambda, f, m)$.*

Note that the transitive closure $\mathcal{F}^*$ on (λ, f, m) might not be efficiently computable (and thus a challenger for Unf might not be efficient). However, this does not influence the security or efficiency of a FSS. If necessary for a specific construction, one can require every functional signature scheme to provide an efficient algorithm $\mathsf{Check}{-}\mathcal{F}$ such that $\mathsf{Check}{-}\mathcal{F}(\lambda, f, m, m^*) = 1$ iff $m^* \in \mathcal{F}^*(\lambda, f, m)$.

Remark: We assume that whenever $\mathsf{Eval}_{\mathcal{F}}$ is called on a signature σ with a key $sk_{\mathcal{F}}$ and does not return $\perp$, then it is possible to extract f from σ using $sk_{\mathcal{F}}$. This is reasonable, because the evaluator that transforms a signature should learn the value f, as it describes the capabilities of the evaluator.

Definition 4. *(Unforgeability Against $X \in \{$Outsider, Insider, S-Insider$\}$ Attacks). Let $\mathsf{FSS} = (\mathsf{Setup}, \mathsf{KGen}_{sig}, \mathsf{KGen}_{\mathcal{F}}, \mathsf{Sig}, \mathsf{Eval}_{\mathcal{F}}, \mathsf{Vf})$ be a functional signature scheme. The definition uses the game $\mathsf{Unf}(\mathsf{FSS}, \mathcal{F}, \mathcal{A}, \lambda)$ defined in Figure 1. The functional signature scheme FSS is* existential unforgeable against X-attacks ($\mathsf{EU\text{-}X\text{-}A}$) *for a functionality $\mathcal{F}$, if for all PPT adversaries $\mathcal{A}_X$*

$$\mathbf{Adv}^{\mathsf{EU\text{-}X\text{-}A}}_{\mathsf{FSS},\mathcal{F},\mathcal{A}_X} = \mathrm{Prob}[\,\mathsf{Unf}(\mathsf{FSS}, \mathcal{F}, \mathcal{A}_X, \lambda) = 1]$$

is negligible in λ. In this definition, $\mathcal{A}_{Outsider}$ does not query $\mathsf{Query[KGenS]}$ or $\mathsf{Query[RegKey]}$, the attacker $\mathcal{A}_{Insider}$ does not invoke $\mathsf{Query[RegKey]}$, and $\mathcal{A}_{S\text{-}Insider}$ may use any oracle.

Strong unforgeability, where the adversary already succeeds if it computes a *new* signature on a (eventually previously signed) message, can be easily defined by letting the $\mathsf{Finalize}$ algorithm check if $(\cdot, m^*, \sigma^*) \in \mathcal{Q}$ (instead of checking whether $(\cdot, m^*, \cdot) \in \mathcal{Q}$). We discuss the relation between the different types of adversaries in the context of delegatable FSS in Section 4.2, and the same relation holds here as well.

2.2.2 Privacy

PROC INITIALIZE(λ) :
$b \leftarrow \{0,1\}$
$(pp, msk) \leftarrow \mathsf{Setup}(\lambda)$
$(sk_{sig}, pk_{sig}) \leftarrow \mathsf{KGen}_{sig}(pp, msk)$
$(pk_{\mathcal{F}}, sk_{\mathcal{F}}) \leftarrow \mathsf{KGen}_{\mathcal{F}}(pp, msk)$
store $(b, pp, sk_{sig}, sk_{\mathcal{F}}, pk_{sig}, pk_{\mathcal{F}})$
output $(pp, pk_{sig}, pk_{\mathcal{F}})$

PROC FINALIZE(b^*) :
retrieve b
if $b = b^*$ then
  output 1
else
  output 0

PROC QUERY[SIGN]($f, m, pk^*_{\mathcal{F}}$) :
retrieve (pp, sk_{sig})
$\sigma \leftarrow \mathsf{Sig}(pp, sk_{sig}, pk^*_{\mathcal{F}}, f, m)$
output σ

PROC QUERY[EVAL]($pk^*_{sig}, \alpha, m, \sigma$) :
retrieve $(pp, sk_{\mathcal{F}})$
$\sigma' \leftarrow \mathsf{Eval}_{\mathcal{F}}(pp, sk_{\mathcal{F}}, pk^*_{sig}, \alpha, m, \sigma)$
output σ

PROC QUERY[SIGN-$\mathcal{F}$]($f, [\alpha]_1^t, m_0$) :
retrieve $(b, pp, sk_{sig}, sk_{\mathcal{F}}, pk_{sig}, pk_{\mathcal{F}})$
$\sigma \leftarrow \mathsf{Sig}(pp, sk_{sig}, pk_{\mathcal{F}}, f, m_0)$
for $i \in \{1, \ldots, t\}$
  $\sigma_i \leftarrow \mathsf{Eval}_{\mathcal{F}}(pp, sk_{\mathcal{F}}, pk_{sig}, \alpha, m_{i-1}, \sigma_{i-1})$
  $m_i := \mathcal{F}(\lambda, f, \alpha_i, m_{i-1})$
if $b = 0 \wedge \sigma_t \neq \perp$ then
  $\sigma \leftarrow \mathsf{Sig}(pp, sk_{sig}, pk_{\mathcal{F}}, f, m_t)$
else
  $\sigma := \sigma_t$
output σ

Figure 2: Privacy under Chosen Functionality Attacks CFA.

Our privacy notion captures the privacy of the evaluator with respect to the signer. The basic idea behind this notion is that it should be hard to distinguish if a message-signature pair has been computed by the signer or by the evaluator via the evaluation algorithm. Our definition of privacy demands that even if multiple evaluations are applied to a signature successively, no polynomially bounded adversary can distinguish the resulting signature from a fresh signature on the underlying message.

More formally, we define *indistinguishability under chosen function attack* for a functional signature scheme $\mathsf{FSS} = (\mathsf{Setup}, \mathsf{KGen}_{sig}, \mathsf{KGen}_{\mathcal{F}}, \mathsf{Sig}, \mathsf{Eval}_{\mathcal{F}}, \mathsf{Vf})$ and a functionality $\mathcal{F}$ as follows:

Definition 5. *(Privacy under chosen functionality attacks (CFA)). Let* $\mathsf{FSS} = (\mathit{Setup}, \mathit{KGen}_{sig}, \mathit{KGen}_{\mathcal{F}}, \mathit{Sig}, \mathit{Eval}_{\mathcal{F}}, \mathit{Vf})$ *be a functional signature scheme. The definition uses* $\mathsf{CFA}(\mathsf{FSS}, \mathcal{F}, \mathcal{A}, \lambda)$ *defined in Figure 2. The scheme* FSS *is* privacy preserving under chosen function attacks (PP-X-CFA) *for a functionality* $\mathcal{F}$*, if for all PPT adversaries* $\mathcal{A}$

$$\mathbf{Adv}^{\mathsf{PP\text{-}X\text{-}CFA}}_{\mathsf{FSS},\mathcal{F},\mathcal{A}} = \left| \mathrm{Prob}[\,\mathsf{CFA}(\mathsf{FSS}, \mathcal{F}, \mathcal{A}, \lambda) = 1] - \frac{1}{2} \right|$$

is negligible in λ.

3 Delegatable Functional Signatures

To move from a functional signature scheme as defined above to a delegatable functional signature scheme requires us to consider a more flexible notion of delegation: Before, we only considered the possibility of the signer for allowing one fixed evaluator to modify a given signature according to a functionality f. An evaluator could only "delegate" its power by giving away its secret key $sk_{\mathcal{F}}$

to another party. We extend the primitive by adding the possibility of delegating (a subset of) capabilities from one evaluator to another without leaking its own secret key.

In this section we present a formal description of a delegatable functional signature (DFS) scheme and update our notions of security and privacy. Although the delegation gives more capabilities to the evaluator, we still want to give the signer of a message as much control as possible over the resulting signature and the capabilities of changing this signature. Thus, we allow the signer to additionally restrict which capabilities can be delegated (and optionally to whom) by specifying a derivation function g on the applicable function f: intuitively, g describes how f can be changed.

Our notions of unforgeability make disrespecting the choices of the signer impossible. On the other hand, the notions of privacy hide whether or not and how many transformations have been performed on a signature and by whom.

3.1 Formal Description of a DFS scheme

A delegatable functional signature (DFS) scheme over a message space $\mathcal{M}$, a key space $\mathcal{K}$ and parameter spaces $\mathcal{P}_f$, $\mathcal{P}_g$, $\mathcal{P}_\alpha$, $\mathcal{P}_\beta$ is a functional signature (FS) scheme that additionally supports a controlled form of delegation. We now require two functionalities instead of one:

- A functionality $\mathcal{F} : \mathbb{N} \times \mathcal{P}_f \times \mathcal{P}_\alpha \times \mathcal{M} \to \mathcal{M} \cup \{\perp\}$ as in Section 2 that specifies how messages can be changed.
- A functionality $\mathcal{G} : \mathbb{N} \times \mathcal{P}_g \times \mathcal{P}_\beta \times \mathcal{K} \times \mathcal{P}_f \to \mathcal{P}_f \cup \{\perp\}$ that specifies how capabilities can be delegated.

In particular, each evaluator that has the capability to compute signatures on messages of its choice can delegate (a modified version of) these capabilities to another evaluator. We model this property by replacing $\mathsf{Eval}_\mathcal{F}$ by an algorithm $\mathsf{Trans}_{\mathcal{FG}}$. To explain the functionality of this algorithm, let σ be a signature on a message m, on which an evaluator that owns $sk_\mathcal{F}$ can apply modifications specified with the capability f. Now, like $\mathsf{Eval}_\mathcal{F}$, $\mathsf{Trans}_{\mathcal{FG}}$ computes a signature σ' on $\mathcal{F}(\lambda, f, \alpha, m)$. However, this new signature σ' can be changed by an evaluator that owns a (possibly different) key $sk_\mathcal{F}{}'$ and this evaluator can transform it further with the new capability $f' := \mathcal{G}(\lambda, g, \beta, pk'_\mathcal{F}, f)$. The following example illustrates the algorithm.

Example 3. *Suppose that the functionalities $\mathcal{F}$ and $\mathcal{G}$ support the following scenario: Alice wants to make sure that an evaluator Bob can only rerandomize signatures on a specific message m_1. She choses an appropriate f_{ID}, such that $\mathcal{F}(\lambda, f_{\mathsf{ID}}, \alpha, m_1) = m_1$ for all α. Such a functionality models the case of a regular signature scheme. Now, suppose that Alice grants Bob the right to compute signatures on some function f on a different message m_2, but at the same time, she wants to restrict the delegation capabilities to subset f' of f. To do so, she chooses a g_B such that $\mathcal{G}(\lambda, g_B, \beta, pk'_\mathcal{F}, f) = f'$ for all $pk'_\mathcal{F} \neq pk_B$.*

Now we formally define the key primitive of this paper: delegatable functional signature schemes.

Definition 6. *(Delegatable functional signatures). A* delegatable functional signature scheme DFSS *is a tuple of efficient algorithms* $\mathsf{DFSS} = (\mathsf{Setup}, \mathsf{KGen}_{sig}, \mathsf{KGen}_\mathcal{F}, \mathsf{Sig}, \mathsf{Trans}_{\mathcal{FG}}, \mathsf{Vf})$. *The algorithms $\mathsf{Setup}, \mathsf{KGen}_{sig}, \mathsf{KGen}_\mathcal{F}$ and Vf have the same interface as for functional signature schemes (see Definition 1). Thus, we concentrate on the new interfaces for Sig and $\mathsf{Trans}_{\mathcal{FG}}$ here.*

$\mathsf{Sig}(\boldsymbol{pp}, \boldsymbol{sk}_{sig}, \boldsymbol{pk}_{\mathcal{F}}, f, g, m)$: *The signing algorithm* Sig *outputs a signature* σ *on* m*, on which functions from the class* f *can be applied, and on which functions from* $g(f, \cdot, \cdot)$ *can be delegated (or an error symbol* $\perp$*).*

$\mathsf{Trans}_{\mathcal{FG}}(\boldsymbol{pp}, \boldsymbol{sk}_{\mathcal{F}}, \boldsymbol{pk}_{sig}, \alpha, \beta, m, \boldsymbol{pk}'_{\mathcal{F}}, \sigma)$: *The transformation algorithm* $\mathsf{Trans}_{\mathcal{FG}}$ *outputs a derived signature for the functionality* $\mathcal{F}(\lambda, f, \alpha, m)$ *(or an error symbol* $\perp$*), that can be modified using the evaluator key* $sk_{\mathcal{F}}{}'$ *associated with* $pk'_{\mathcal{F}}$ *on the capability* $f' := \mathcal{G}(\lambda, g, \beta, pk'_{\mathcal{F}}, f)$*.*

Intuitively, we consider a delegatable functional signature scheme DFSS to be correct if the verification algorithm outputs 1 for all honestly generated signatures and for all valid transformations of honestly generated signatures.

To formally state our notion of correctness, we define a *correctness set* S for which Vf should output 1.

Definition 7. *($(\mathcal{F},\mathcal{G})$-Correctness). Let the correctness set* S *be a set such that:*

$\forall f \in \mathcal{P}_f, g \in \mathcal{P}_g, \alpha \in \mathcal{P}_\alpha, \beta \in \mathcal{P}_\beta, m \in \mathcal{M}, (msk, pp) \in [\mathsf{Setup}(\lambda)], (sk_{sig}, pk_{sig}) \in [\mathsf{KGen}_{sig}(pp, msk)]$, $(sk_{\mathcal{F}}, pk_{\mathcal{F}}), (sk_{\mathcal{F}}{}', pk'_{\mathcal{F}}) \in [\mathsf{KGen}_{\mathcal{F}}(pp, msk)]$

1) $(pp, m, f, g, \sigma, pk_{sig}, pk_{\mathcal{F}}) \in S$ *for all* $\sigma \in [\mathsf{Sig}(pp, sk_{sig}, pk_{\mathcal{F}}, f, g, m)]$.

2) *If* $(pp, m, f, g, \sigma, pk_{sig}, pk_{\mathcal{F}}) \in S$*,* $\mathcal{F}(\lambda, f, \alpha, m) = \hat{m}$*,* $\mathcal{G}(\lambda, g, \beta, pk'_{\mathcal{F}}, f) = \hat{f}$ *with* $\hat{m} \in \mathcal{M}$*,* $\hat{f} \in \mathcal{P}_f$*, then* $(pp, \hat{m}, \hat{f}, g, \hat{\sigma}, pk_{sig}, pk'_{\mathcal{F}}) \in S$ *for all* $\hat{\sigma} \in [\mathsf{Trans}_{\mathcal{FG}}(pp, sk_{\mathcal{F}}, pk_{sig}, \alpha, \beta, m, pk'_{\mathcal{F}}, \sigma)]$

A delegatable functional signature scheme $\mathsf{DFSS} = (\mathsf{Setup}, \mathsf{KGen}_{sig}, \mathsf{KGen}_{\mathcal{F}}, \mathsf{Sig}, \mathsf{Trans}_{\mathcal{FG}}, \mathsf{Vf})$ *is* $(\mathcal{F},\mathcal{G})$-correct *for functionalities* $\mathcal{F}$*,* $\mathcal{G}$*, if for all elements* $(pp, m, f, g, \sigma, pk_{sig}, pk_{\mathcal{F}}) \in S$ *it holds that*

$$\mathsf{Vf}(pp, pk_{sig}, pk_{\mathcal{F}}, m, \sigma) = 1.$$

4 Security Notions for DFSS

As for a functional signature schemes, we propose definitions for unforgeability and privacy for delegatable functional signature schemes.

4.1 Unforgeability

Our definition of unforgeability is similar to the one for FSS (Definition 4), as we handle security against outsider and (strong) insider. We model this by giving the adversary access to three different KGen oracles. An adversary that only uses $\mathsf{Query[KGenP]}$ for public keys is considered an *outsider*, an adversary that additionally can query $\mathsf{Query[KGenS]}$ to retrieve one or more secret evaluator keys is considered an *insider*, and an adversary that additionally can use the oracle $\mathsf{Query[RegKey]}$ to register its own (possibly malicious) evaluator keys is considered a *S-Insider*. All adversaries have access to the honestly generated public signer key pk_{sig}.

The notion of unforgeability against an outsider for DFSS is similar to our definition for FSS with the difference being that the adversary has access to the Sig and the $\mathsf{Trans}_{\mathcal{FG}}$ oracle.

Checking whether a message/signature pair is a valid forgery w.r.t. insiders is more complicated. To handle the information that an adversary can trivially deduce from its queries, we define the transitive closure for functionalities.

PROC INITIALIZE (λ):
$(pp, msk) \leftarrow \mathsf{Setup}(\lambda)$
$(sk_{sig}, pk_{sig}) \leftarrow \mathsf{KGen}_{sig}(pp, msk)$
store $(pp, msk, sk_{sig}, pk_{sig})$
set $\mathcal{K}_{\mathcal{C}} := \emptyset, \mathcal{K}_{\mathcal{A}} := \emptyset, \mathcal{Q} := \emptyset$
output (pp, pk_{sig})

PROC FINALIZE$(m^*, \sigma^*, pk^*_{\mathcal{F}})$:
if $\exists (f, g, m, pk_{\mathcal{F}}, \cdot) \in \mathcal{Q}, s.t.$
$pk_{\mathcal{F}} \in \mathcal{K}_{\mathcal{A}} \wedge m^* \in \mathcal{F}^*_{\mathcal{G}}(\lambda, f, g, m)$
  output 0
else
  if $(\cdot, \cdot, m^*, \cdot, \cdot) \in \mathcal{Q}$
    output 0
  else
    retrieve (pp, pk_{sig})
    $b \leftarrow \mathsf{Vf}(pp, pk_{sig}, pk^*_{\mathcal{F}}, m^*, \sigma^*)$
    output b

PROC QUERY[KGENP]() :
retrieve (pp, msk)
$(sk_{\mathcal{F}}, pk_{\mathcal{F}}) \leftarrow \mathsf{KGen}_{\mathcal{F}}(pp, msk)$
set $\mathcal{K}_{\mathcal{C}} := \mathcal{K}_{\mathcal{C}} \cup (sk_{\mathcal{F}}, pk_{\mathcal{F}})$
output $(pk_{\mathcal{F}})$

PROC QUERY[KGENS]() :
retrieve (pp, msk)
$(sk_{\mathcal{F}}, pk_{\mathcal{F}}) \leftarrow \mathsf{KGen}_{\mathcal{F}}(pp, msk)$
set $\mathcal{K}_{\mathcal{C}} := \mathcal{K}_{\mathcal{C}} \cup \{(sk_{\mathcal{F}}, pk_{\mathcal{F}})\}$
set $\mathcal{K}_{\mathcal{A}} := \mathcal{K}_{\mathcal{A}} \cup \{pk_{\mathcal{F}}\}$
output $(sk_{\mathcal{F}}, pk_{\mathcal{F}})$

PROC QUERY[REGKEY]$(sk^*_{\mathcal{F}}, pk^*_{\mathcal{F}})$:
set $\mathcal{K}_{\mathcal{C}} := \mathcal{K}_{\mathcal{C}} \cup \{(sk^*_{\mathcal{F}}, pk^*_{\mathcal{F}})\}$
set $\mathcal{K}_{\mathcal{A}} := \mathcal{K}_{\mathcal{A}} \cup \{pk^*_{\mathcal{F}}\}$

PROC QUERY[SIGN]$(pk^*_{\mathcal{F}}, f, g, m)$:
retrieve (pp, sk_{sig})
if $(\cdot, pk^*_{\mathcal{F}}) \in \mathcal{K}_{\mathcal{C}}$
  $\sigma \leftarrow \mathsf{Sig}(pp, sk_{sig}, pk^*_{\mathcal{F}}, f, g, m)$
  set $\mathcal{Q} := \mathcal{Q} \cup \{(f, g, m, pk^*_{\mathcal{F}}, \sigma)\}$
  output σ
else output $\perp$

PROC QUERY[TRANS]$(pk^*_{\mathcal{F}}, \alpha, \beta, m, pk'_{\mathcal{F}}, \sigma)$:
retrieve (pp, pk_{sig})
if $(sk^*_{\mathcal{F}}, pk^*_{\mathcal{F}}) \in \mathcal{K}_{\mathcal{C}} \wedge (\cdot, pk'_{\mathcal{F}}) \in \mathcal{K}_{\mathcal{C}}$
  $\sigma' \leftarrow \mathsf{Trans}_{\mathcal{FG}}(pp, sk^*_{\mathcal{F}}, pk_{sig}, \alpha, \beta, m, pk'_{\mathcal{F}}, \sigma)$
  if $\sigma' \neq \perp$
    extract (f, g) from σ using $sk^*_{\mathcal{F}}$
    let $f' := \mathcal{G}(\lambda, g, \beta, pk'_{\mathcal{F}}, f)$
    set $\mathcal{Q} := \mathcal{Q} \cup \{f', g, \mathcal{F}(\lambda, f, \alpha, m), pk'_{\mathcal{F}}, \sigma'\}$
  output σ'
else output $\perp$

Figure 3: Unforgeability for delegatable functional signature schemes.

Definition 8. *(Transitive closure of functionality $\mathcal{F}$ under functionality $\mathcal{G}$). Given functionalities $\mathcal{F}$ and $\mathcal{G}$, we define the n-transitive closure $\mathcal{F}^n_{\mathcal{G}}$ of $\mathcal{F}$ under functionality $\mathcal{G}$ on parameters (λ, f, g, m) recursively as follows:*

- *For $n = 0$, $\mathcal{F}^0_{\mathcal{G}}(\lambda, f, g, m) := \{m\}$.*
- *For $n > 0$, $\mathcal{F}^n_{\mathcal{G}}(\lambda, f, g, m) := \{m\} \bigcup_{\alpha, \beta, pk'_{\mathcal{F}}} \mathcal{F}^{n-1}_{\mathcal{G}}(\lambda, \mathcal{G}(\lambda, g, \beta, pk'_{\mathcal{F}}, f), g, \mathcal{F}(\lambda, f, \alpha, m))$*

We define the transitive closure $\mathcal{F}^*_{\mathcal{G}}$ *of $\mathcal{F}$ under functionality $\mathcal{G}$ on parameters (λ, f, g, m) as*

$$\mathcal{F}^*_{\mathcal{G}}(\lambda, f, g, m) := \bigcup_{i=0}^{\infty} \mathcal{F}^i_{\mathcal{G}}(\lambda, f, g, m).$$

Again, the transitive closure might not be efficiently computable. However, for the security of our construction (see Section 5) we do not need to compute the closure explicitly. If necessary for another construction, one can require a DFSS to provide an efficient algorithm $\mathsf{Check}{-}\mathcal{F}_{\mathcal{G}}$ such that $\mathsf{Check}{-}\mathcal{F}_{\mathcal{G}}(\lambda, f, g, m, m^*) = 1$ iff $m \in \mathcal{F}^*_{\mathcal{G}}(\lambda, f, g, m)$.

Remark: We assume that whenever $\mathsf{Trans}_{\mathcal{FG}}$ is called on a signature σ with a key $sk_{\mathcal{F}}$ and does not return $\perp$, then it is possible to extract (f, g) from σ using $sk_{\mathcal{F}}$. This is reasonable, because the evaluator that transforms a signature should learn both values, as they describe its capabilities. In fact, our construction (Section 5) also has this property.

Definition 9. *(Unforgeability Against $X \in \{$Outsider, Insider, S-Insider$\}$ Attacks). Let* DFSS *be a delegatable functional signature scheme defined by the following efficient algorithms* ($\mathsf{Setup}, \mathsf{KGen}_{sig}$, $\mathsf{KGen}_{\mathcal{F}}, \mathsf{Sig}, \mathsf{Trans}_{\mathcal{FG}}, \mathsf{Vf}$). *The definition uses the game* $\mathsf{Unf}(\mathsf{DFSS}, \mathcal{F}, \mathcal{G}, \mathcal{A}, \lambda)$ *defined in Figure 3. We say that* DFSS *is* existential unforgeable against X-attacks (EU-X-A) *for functionalities $\mathcal{F}$ and $\mathcal{G}$ if for all PPT adversaries $\mathcal{A}_X$*

$$\mathbf{Adv}^{\mathsf{EU\text{-}X\text{-}A}}_{\mathsf{DFSS},\mathcal{F},\mathcal{G},\mathcal{A}_X} = Pr\left[\mathsf{Unf}(\mathsf{DFSS}, \mathcal{F}, \mathcal{G}, \mathcal{A}_X, \lambda) = 1\right]$$

is negligible in λ, where $\mathcal{A}_{Outsider}$ does neither invoke the oracles Query[KGenS] *nor* Query[RegKey]*; the attacker $\mathcal{A}_{Insider}$ does not make use of* Query[RegKey] *and the adversary $\mathcal{A}_{S\text{-}Insider}$ is not restricted in its queries.*

Remark on measuring the success of $\mathcal{A}$: The success of the adversary is determined by the challenger and measured in the Finalize algorithm. Although not stated explicitly, Finalize distinguishes between outsiders and insiders. Within the oracles Query[Sign] and Query[Trans], the challenger only allows to delegate to keys that are "known" to it, which is formalized with the set $\mathcal{K}_{\mathcal{C}}$. The oracle Query[Trans] only allows the delegation to keys that are known to the challenger. Note that his does not restrict the adversary, but allows the challenger to distinguish between weak insider and strong insider. Whenever a message has been signed either by Query[Sign] or Query[Trans], this message is included in $\mathcal{Q}$, together with the public key of the evaluator to whom the message was delegated and together with the parameters that state what this evaluator can trivially deduce from the signature.

The set $\mathcal{K}_{\mathcal{A}}$ is the set of all public evaluator keys $pk_{\mathcal{F}}$ for which the adversary knows the secret key $sk_{\mathcal{F}}$. Consequently, $\mathcal{K}_{\mathcal{A}}$ is initially empty and is only extended by Query[KGenS] and Query[RegKey]. Whenever $\mathcal{A}$ delegates a signature to a key $pk'_{\mathcal{F}} \in \mathcal{K}_{\mathcal{A}}$, the finalize algorithm will later discard all message-signature pairs that are trivially deducible from this signature.

For both outsiders ($\mathcal{K}_{\mathcal{A}} = \emptyset$) and insiders ($\mathcal{K}_{\mathcal{A}} \neq \emptyset$), we require that the forgery message m^* is a fresh message, i.e., it has not been signed by the challenger, which is formally expressed by $(\cdot, \cdot, m^*, \cdot, \cdot) \neq \mathcal{Q}$.

Observe that a different public key $pk_{\mathcal{F}}$ might have been used when signing a message as compared to when verifying the resulting signature. We leave it up to the signature scheme to decide whether a signature can verify under different evaluator keys. As a matter of fact: There can be schemes where Vf does not need to receive $pk_{\mathcal{F}}$ at all.

4.2 Relations between the security notions depending on the strength of the adversary

The three notions of unforgeability describe a hierarchy of adversaries. It is intuitive, that security against outsider attacks does not imply security against insider attacks, as the key $sk_{\mathcal{F}}$ of the evaluator can indeed leak enough information to construct the signature key sk_{sig} out of it.

However, although an *insider* adversary is stronger than an *outsider* adversary, making use of the additional oracle can weaken an adversary. Consider a scheme with only one valid public evaluator key $pk_{\mathcal{F}}$, that allows an insider to change messages inside signatures to arbitrary values, but that also leaks the secret signing key sk_{sig} with every signature. An insider that received sk_{sig} can not create a forgery, since every message he creates after receiving at least one signature is not considered a forgery: he could have computed them trivially using $\mathsf{Trans}_{\mathcal{FG}}$. Without invoking

Query[KGenS], the adversary can request a signature and subsequently forge signatures for arbitrary messages, using the key sk_{sig} he received with the signature.

An *S-Insider* is again stronger than an *insider* or an *outsider*. A scheme can become insecure if a certain key pair $(sk_{\mathcal{F}}, pk_{\mathcal{F}})$ is used that is highly unlikely to be an output of $\mathsf{KGen}_{\mathcal{F}}$ (e.g., one of them is 0^{λ}).

Proposition 1 (EU-X-A-Implications)**.** *Let* DFSS *be a functional signature scheme.*

(i) *For all PPT adversaries* $\mathcal{A}_{Outsider}$ *there exists a PPT adversary* $\mathcal{A}_{Insider}$ *s.t.*

$$\mathbf{Adv}^{\mathsf{EU\text{-}IA}}_{\mathsf{DFSS},\mathcal{F},\mathcal{G},\mathcal{A}_{Insider}} \geq \mathbf{Adv}^{\mathsf{EU\text{-}OA}}_{\mathsf{DFSS},\mathcal{F},\mathcal{G},\mathcal{A}_{Outsider}}$$

(ii) *For all PPT adversaries* $\mathcal{A}_{Insider}$ *there exists a PPT adversary* $\mathcal{A}_{S\text{-}Insider}$ *s.t.*

$$\mathbf{Adv}^{\mathsf{EU\text{-}SIA}}_{\mathsf{DFSS},\mathcal{F},\mathcal{G},\mathcal{A}_{S\text{-}Insider}} \geq \mathbf{Adv}^{\mathsf{EU\text{-}IA}}_{\mathsf{DFSS},\mathcal{F},\mathcal{G},\mathcal{A}_{Insider}}$$

4.3 Privacy

PROC INITIALIZE (λ):
$b \leftarrow \{0,1\}$
$(pp, msk) \leftarrow \mathsf{Setup}(\lambda)$
$(sk_{sig}, pk_{sig}) \leftarrow \mathsf{KGen}_{sig}(pp, msk)$
store $(b, pp, msk, sk_{sig}, pk_{sig})$
set $\mathcal{K}_{\mathcal{C}} := \emptyset, \mathcal{K}_X := \emptyset$
set $\mathcal{K}_{\mathcal{A}} := \emptyset$
output (pp, pk_{sig})

PROC FINALIZE(b^*) :
retrieve b
if $b = b^* \wedge \mathcal{K}_X \cap \mathcal{K}_{\mathcal{A}} = \emptyset$ then output 1
else output 0

PROC QUERY[TRANS]$(pk^*_{\mathcal{F}}, \alpha, \beta, m, pk'_{\mathcal{F}}, \sigma)$:
retrieve (pp, pk_{sig})
if $(sk^*_{\mathcal{F}}, pk^*_{\mathcal{F}}) \in \mathcal{K}_{\mathcal{C}} \wedge (pk'_{\mathcal{F}}, \cdot) \in \mathcal{K}_{\mathcal{C}}$
  $\sigma' \leftarrow \mathsf{Trans}_{\mathcal{FG}}(pp, sk^*_{\mathcal{F}}, pk_{sig}, \alpha, \beta, m, pk'_{\mathcal{F}}, \sigma)$
  output σ'

PROC QUERY[KGENP]() :
retrieve (pp, msk)
$(sk_{\mathcal{F}}, pk_{\mathcal{F}}) \leftarrow \mathsf{KGen}_{\mathcal{F}}(pp, msk)$
set $\mathcal{K}_{\mathcal{C}} := \mathcal{K}_{\mathcal{C}} \cup (sk_{\mathcal{F}}, pk_{\mathcal{F}})$
output $(pk_{\mathcal{F}})$

PROC QUERY[KGENS]() :
retrieve (pp, msk)
$(sk_{\mathcal{F}}, pk_{\mathcal{F}}) \leftarrow \mathsf{KGen}_{\mathcal{F}}(pp, msk)$
set $\mathcal{K}_{\mathcal{C}} := \mathcal{K}_{\mathcal{C}} \cup \{(sk_{\mathcal{F}}, pk_{\mathcal{F}})\}$
set $\mathcal{K}_{\mathcal{A}} := \mathcal{K}_{\mathcal{A}} \cup \{pk_{\mathcal{F}}\}$
output $(sk_{\mathcal{F}}, pk_{\mathcal{F}})$

PROC QUERY[SIGN]$(pk^*_{\mathcal{F}}, f, g, m)$:
retrieve (pp, sk_{sig})
if $(\cdot, pk^*_{\mathcal{F}}) \in \mathcal{K}_{\mathcal{C}}$
  $\sigma \leftarrow \mathsf{Sig}(pp, sk_{sig}, pk^*_{\mathcal{F}}, f, g, m)$
  output σ

PROC QUERY[REGKEY]$(sk^*_{\mathcal{F}}, pk^*_{\mathcal{F}})$:
set $\mathcal{K}_{\mathcal{C}} := \mathcal{K}_{\mathcal{C}} \cup \{(sk^*_{\mathcal{F}}, pk^*_{\mathcal{F}})\}$
set $\mathcal{K}_{\mathcal{A}} := \mathcal{K}_{\mathcal{A}} \cup \{pk^*_{\mathcal{F}}\}$

PROC QUERY[SIGN-$\mathcal{F}$]$(f_0, g, [pk_{\mathcal{F}}, \alpha, \beta]^t_0, t, m_0)$:
retrieve $(b, pp, sk_{sig}, pk_{sig})$
if $(\cdot, pk_{\mathcal{F}}[0]) \notin \mathcal{K}_{\mathcal{C}} \vee (\cdot, pk_{\mathcal{F}}[t]) \notin \mathcal{K}_{\mathcal{C}}$ output $\perp$
$\sigma_0 \leftarrow \mathsf{Sig}(pp, sk_{sig}, pk_{\mathcal{F}}[0], f_0, g, m_0)$
for $i \in \{1, \ldots, t\}$
  if $\neg \exists sk^*_{\mathcal{F}}.\ (sk^*_{\mathcal{F}}, pk_{\mathcal{F}}[i-1]) \in \mathcal{K}_{\mathcal{C}}$
    output $\perp$
  $f_i := \mathcal{G}(\lambda, g, \beta[i], pk_{\mathcal{F}}[i], f_{i-1})$
  $m_i := \mathcal{F}(\lambda, f_{i-1}, \alpha[i], m_{i-1})$
  $q_i := (pp, sk^*_{\mathcal{F}}, pk_{sig}, \alpha[i], \beta[i], m_{i-1}, pk_{\mathcal{F}}[i], \sigma_{i-1})$
  $\sigma_i \leftarrow \mathsf{Trans}_{\mathcal{FG}}(q_i)$
set $\mathcal{K}_X := \mathcal{K}_X \cup \{pk_{\mathcal{F}}[t]\}$
if $b = 0 \wedge \sigma_t \neq \perp$
  $\sigma \leftarrow \mathsf{Sig}(pp, sk_{sig}, pk_{\mathcal{F}}[t], f_t, g, m_t)$
else
  $\sigma := \sigma_t$
output σ

Figure 4: Privacy under chosen functionality attacks CFA for delegatable functional signature schemes.

Our privacy notion for delegatable functional signatures captures the idea that it should be hard to distinguish the following two signatures:

- a signature on a message m' that has been derived from a signature on a challenge message m by one or more applications of $\mathsf{Trans}_{\mathcal{FG}}$.

- a signature on m' that was output of Sig.

This indistinguishability should hold even against an adversary with oracle access to $\mathsf{KGen}_{\mathcal{F}}$, Sig and $\mathsf{Trans}_{\mathcal{FG}}$ that can choose which transformations are to be applied to which challenge message m and under which evaluator keys (even if they are known to the adversary), as long as the resulting signature is not delegated to the adversary.

Analogously to our definitions of unforgeability, we distinguish between three different types of adversaries, depending on their strength: outsiders, insiders and strong insiders. We model this by giving the adversary access to three different KGen oracles. An adversary that only can invoke $\mathsf{Query[KGenP]}$ for public keys is considered an *outsider*, an adversary that additionally can query $\mathsf{Query[KGenS]}$ for retrieving one or more secret evaluator keys is considered an *insider*, and an adversary that $\mathsf{Query[RegKey]}$ to register its own possibly malicious evaluator keys is considered a *S-Insider*. All adversaries have access to the honestly generated public signer key pk_{sig}.

Definition 10. *(Privacy under chosen function attacks (CFA)) against $X \in \{$Outsider, Insider, S-Insider$\}$. Let $\mathsf{DFSS} = (\mathsf{Setup}, \mathsf{KGen}_{sig}, \mathsf{KGen}_{\mathcal{F}}, \mathsf{Sig}, \mathsf{Trans}_{\mathcal{FG}}, \mathsf{Vf})$ be a delegatable functional signature scheme. The definition uses the game $\mathsf{CFA}(\mathsf{DFSS}, \mathcal{F}, \mathcal{G}, \mathcal{A}, \lambda)$ defined in Figure 4. We say that* DFSS *is* privacy-preserving under chosen function attacks ($\mathsf{X\text{-}CFA}$) *for functionalities $\mathcal{F}$ and $\mathcal{G}$ if for all PPT adversaries $\mathcal{A}_X$*

$$\mathbf{Adv}^{\mathsf{PP\text{-}X\text{-}CFA}}_{\mathsf{DFSS},\mathcal{F},\mathcal{G},\mathcal{A}_X} = \left| Pr\left[\mathsf{CFA}(\mathsf{DFSS}, \mathcal{F}, \mathcal{G}, \mathcal{A}, \lambda) = 1\right] - \frac{1}{2} \right|$$

is negligible in λ, where $\mathcal{A}_{Outsider}$ does neither invoke the oracles $\mathsf{Query[KGenS]}$ nor $\mathsf{Query[RegKey]}$; the attacker $\mathcal{A}_{Insider}$ does not make use of $\mathsf{Query[RegKey]}$ and the adversary $\mathcal{A}_{S\text{-}Insider}$ is not restricted in its queries.

Remark on measuring the success of $\mathcal{A}$: The adversary may choose an arbitrary challenge message m_0, together with a capability f_0 and a value g to define the delegatability. The challenger constructs a respective signature σ_0. Furthermore the challenger repeatedly applies $\mathsf{Trans}_{\mathcal{FG}}$ to σ_0 and allows the adversary to choose the parameters α_i and β_i that are input to $\mathsf{Trans}_{\mathcal{FG}}$ and the key $pk_{\mathcal{F}}[i]$ to which the signature is delegated. However, $\mathcal{A}$ may not choose keys that are not known to the challenger. By this restriction we distinguish between outsiders, insiders and strong insiders.

Whenever the challenger applies $\mathsf{Trans}_{\mathcal{FG}}$ within the query $\mathsf{Query[Sign\text{-}\mathcal{F}]}$, it additionally computes the new values for m and f for the resulting signature. Thus, it finally produces a signature σ_t, on a message m_t on which the owner of $pk_{\mathcal{F}}[t]$ can apply the capability f_t. The challenger adds the key $pk_{\mathcal{F}}[t]$ to which a challenge has been issued, to the set $\mathcal{K}_X$.

If one of the transformations failed and the resulting signature is not a valid signature ($\sigma_t = \perp$), $\mathsf{Query[Sign\text{-}\mathcal{F}]}$ outputs $\perp$ independently from the value of b. The reason is, that we only want to give guarantees for valid signatures and not extend the notion of correctness from Definition 7.

Otherwise, depending on the value of b, the challenger outputs σ_t or creates a new signature on m_t with capability f_t and delegatability g.

The success of $\mathcal{A}$ is computed by checking whether $\mathcal{A}$ guessed the correct value for b. However, if $\mathcal{A}$ delegated a challenge signature to a key $pk_{\mathcal{F}}[t]$ to which $\mathcal{A}$ knows the secret key ($pk_{\mathcal{F}}[t] \in \mathcal{K}_{\mathcal{A}} \cap \mathcal{K}_X$), the challenger outputs 0. This way we allow a scheme to leak some information to the evaluator to which a signature is delegated. However, only "local" information is allowed. After one delegation, this information has to vanish, since $\mathcal{A}$ has access to a $\mathsf{Trans}_{\mathcal{FG}}$ oracle and can delegate the signature σ_t to a key $pk^*_{\mathcal{F}} \in \mathcal{K}_{\mathcal{A}}$.

4.4 Relations between the security notions depending on the strength of the adversary

For privacy, we have the same hierarchy: A scheme that is secure against outsiders may be insecure against insiders, as the key $sk_{\mathcal{F}}$ of an evaluator can help to distinguish between delegated and fresh signatures. Again, calling Query[KGenS] might weaken the adversary. Consider a scheme that does not preserve privacy against outsiders and that only has one valid evaluator key. An insider that calls both Query[KGenS] and Query[Sign-$\mathcal{F}$] is discarded, because it knows the only valid evaluator key (and thus $\mathcal{K}_X \cap \mathcal{K}_{\mathcal{A}} \neq \emptyset$).

As for unforgeability, an S-Insider, is stronger than outsider or against an insider. A scheme can leak information about delegation if a certain key pair $(sk_{\mathcal{F}}, pk_{\mathcal{F}})$ is used that is highly unlikely to be an output of $\mathsf{KGen}_{\mathcal{F}}$ (e.g., one of them is 0^{λ}).

Proposition 2 (PP-X-CFA-Implications)**.** *Let* DFSS *be a functional signature scheme.*

(i) *For all PPT adversaries $\mathcal{A}_{Outsider}$ there exists a PPT adversary $\mathcal{A}_{Insider}$ s.t.*

$$\mathbf{Adv}^{\mathsf{PP\text{-}I\text{-}CFA}}_{\mathsf{DFSS},\mathcal{F},\mathcal{G},\mathcal{A}_{Insider}} \geq \mathbf{Adv}^{\mathsf{PP\text{-}O\text{-}CFA}}_{\mathsf{DFSS},\mathcal{F},\mathcal{G},\mathcal{A}_{Outsider}}$$

(ii) *For all PPT adversaries $\mathcal{A}_{Insider}$ there exists a PPT adversary $\mathcal{A}_{S\text{-}Insider}$ s.t.*

$$\mathbf{Adv}^{\mathsf{PP\text{-}SI\text{-}CFA}}_{\mathsf{DFSS},\mathcal{F},\mathcal{G},\mathcal{A}_{S\text{-}Insider}} \geq \mathbf{Adv}^{\mathsf{PP\text{-}I\text{-}CFA}}_{\mathsf{DFSS},\mathcal{F},\mathcal{G},\mathcal{A}_{Insider}}$$

Proof. The proposition follows trivially. Observe:

(i) The adversary $\mathcal{A}_{\text{Insider}}$ runs a black-box simulation of $\mathcal{A}_{\text{Outsider}}$ and makes no use of the additional oracle.

(ii) The adversary $\mathcal{A}_{\text{S-Insider}}$ runs a black-box simulation of $\mathcal{A}_{\text{Insider}}$ and makes no use of the additional oracle.

□

5 Constructing DFSS

In this section we construct a delegatable functional signature scheme DFSS as defined in Section 3.1. Our construction is based on (regular) unforgeable signature schemes, a public-key encryption scheme, and a non-interactive zero-knowledge proof system. Before presenting the construction we give a brief overview over these underlying primitives. We will omit the (standard) definitions for correctness and security.

5.1 Cryptographic Primitives

Digital Signatures. A (regular) signature scheme is a tuple of efficient algorithms $S = (\mathsf{Setup}_S, \mathsf{KGen}_S, \mathsf{Sig}_S, \mathsf{Vf}_S)$, where Setup_S returns some public parameters pp and a master secret key msk, $\mathsf{KGen}_S(pp, msk)$ outputs a secret signing key sk and a public verification key pk, Sig_S signs messages using a key sk and Vf_S checks whether a signature is valid for a given message and a given verification

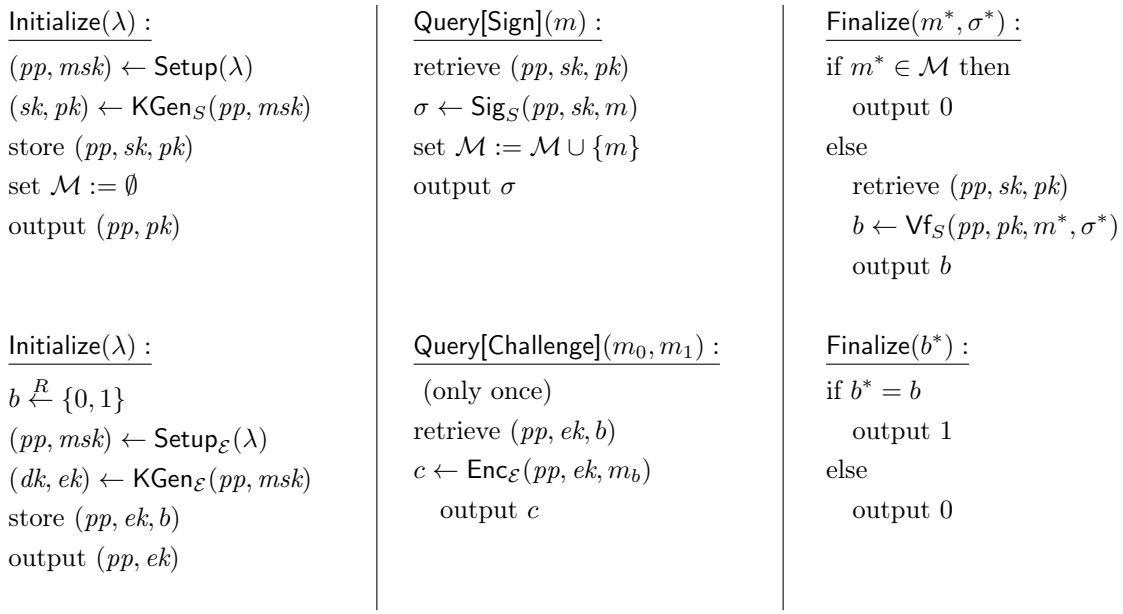

Figure 5: (u.) Unforgeability for (regular) signature schemes (Unf) and (d.) security under chosen plaintext attacks (CPA)

key pk. A signature scheme is length preserving, if for a fixed key-pair, the signing algorithm outputs signatures of equal length, if the messages have the same length, i.e., if $|m_1| = |m_2|$, then $|\sigma_1 \leftarrow \mathsf{Sig}_{\mathcal{S}}(pp, sk, m_1)| = |\sigma_2 \leftarrow \mathsf{Sig}_{\mathcal{S}}(pp, sk, m_2)|$

Definition 11. *(Correctness of a signature scheme). A signature scheme* $S = (\mathsf{Setup}_S, \mathsf{KGen}_S, \mathsf{Sig}_S, \mathsf{Vf}_S)$ *is* correct*, if for all* $\lambda \in \mathbb{N}$*, all* $m \in \mathcal{M}$ *and for all* $(pp, msk) \in [\mathsf{Setup}_S(\lambda)]$ *and all* $(sk, pk) \in [\mathsf{KGen}_S(pp, msk)]$ *the following property holds:*

$$\mathsf{Vf}_S(pp, pk, m, (\mathsf{Sig}_S(pp, sk, m))) = 1$$

A signature is unforgeable, if it is computationally hard to forge signatures for new messages without access to the signing key, even if arbitrary messages have been signed before.

Definition 12. *(Unforgeability). A signature scheme* $S = (\mathsf{Setup}_S, \mathsf{KGen}_S, \mathsf{Sig}_S, \mathsf{Vf}_S)$ *is* unforgeable *if for all PPT adversaries* $\mathcal{A}$ *the probability*

$$Pr[\mathrm{Unf}(S, \mathcal{A}, \lambda) = 1].$$

is negligible in λ*. The definition uses* $\mathsf{Unf}(\mathcal{S}, \mathcal{A}, \lambda)$ *defined in Figure 5.*

The definition can easily be strengthened to *strong unforegability* by adding the requirement that the pair (m, σ) has never been learned from the queries/answer pair to the signing oracle. Obviously, this definition is stronger because an attacker succeeds even if he outputs a new signature for a message he has sent to the signing oracle before.

For our construction we need to make two additional assumption about the signature scheme. The first property says that no master key is necessary in order to generate a key-pair and the second property demands that signatures on message of equal length have the same size. More precisely, a signature scheme S has a *simple key generation* algorithm if the key generation does not depend on a master secret key.

Definition 13. *(Simple Key Generation). A signature scheme* $S = (\mathsf{Setup}_S, \mathsf{KGen}_S, \mathsf{Sig}_S, \mathsf{Vf}_S))$ *has a* simple key generation *algorithm if there exists an* ε *such that for ll* $(pp, msk) \in [\mathsf{Setup}_S]$ *it holds that* $msk = \varepsilon$.

A signature scheme S has length preserving signatures if the length of a signature does only depend on public parameters and on the length of the underlying message.

Definition 14. *(Length Preservation). A signature scheme* $S = (\mathsf{Setup}_S, \mathsf{KGen}_S, \mathsf{Sig}_S, \mathsf{Vf}_S))$ *has* length preserving signatures *if for all* $m_1, m_2 \in \mathcal{M}$*, for all* $(pp, msk) \in [\mathsf{Setup}_S]$*, and for all* $(sk, vk) \in [\mathsf{KGen}_S(pp, msk)]$ *it holds that*

$$|\sigma_1 \leftarrow \mathsf{Sig}_{\mathcal{S}}(pp, sk, m_1)| = |\sigma_2 \leftarrow \mathsf{Sig}_{\mathcal{S}}(pp, sk, m_2)|.$$

Encryption. A *public key encryption scheme* $\mathcal{E}$ is a tuple of efficient algorithms $\mathcal{E} = (\mathsf{Setup}_{\mathcal{E}}, \mathsf{KGen}_{\mathcal{E}}, \mathsf{Enc}_{\mathcal{E}}, \mathsf{Dec}_{\mathcal{E}})$, where $\mathsf{Setup}_{\mathcal{E}}(\lambda)$ returns some public parameters pp and a master secret key msk, $\mathsf{KGen}_{\mathcal{E}}(pp, msk)$ outputs a secret decryption key dk and a public encryption key ek, $\mathsf{Enc}_{\mathcal{E}}$ encrypts messages using ek and $\mathsf{Dec}_{\mathcal{E}}$ decrypts cipher texts using dk.

Definition 15. *(Correctness of a public key encryption scheme). A public-key encryption scheme* $\mathcal{E} = (\mathsf{Setup}_{\mathcal{E}}, \mathsf{KGen}_{\mathcal{E}}, \mathsf{Enc}_{\mathcal{E}}, \mathsf{Dec}_{\mathcal{E}})$ correct*, if for all* $\lambda \in \mathbb{N}$*, all* $m \in \{0,1\}^{\ell_m(\lambda)}$*, and for all* $(pp, msk) \in [\mathsf{Setup}_{\mathcal{E}}(\lambda)]$ *and all* $(dk, ek) \in [\mathsf{KGen}_{\mathcal{E}}(pp, msk)]$ *the following property holds:* $\mathsf{Dec}_{\mathcal{E}}(pp, dk, (\mathsf{Enc}_{\mathcal{E}}(pp, ek, m))) = m$.

A public key encryption scheme is secure against chosen plaintext attack (CPA) if no adversary with access to the public parameters pp and the public (encryption) key ek is able to distinguish between the encryptions of two messages of its own choice.

Definition 16. *(Security against chosen plaintext attacks (CPA)). A public-key encryption scheme* $\mathcal{E} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}_{\mathcal{E}}, \mathsf{Dec}_{\mathcal{E}})$ *is* secure against chosen plaintext attacks (CPA)*, if for all PPT adversaries* $\mathcal{A}$

$$\mathbf{Adv}^{\mathsf{CPA}}_{\mathcal{E},\mathcal{A}} = \left| Pr\left[\mathsf{CPA}(\mathcal{E}, \mathcal{A}, \lambda) = 1\right] - \frac{1}{2} \right|,$$

is negligible in λ*, where* $\mathsf{CPA}(\mathcal{E}, \mathcal{A}, \lambda)$ *is defined in Figure 5.*

Non-interactive zero-knowledge (NIZK). A non-interactive zero-knowledge proof system for a relation R is a tuple of efficient algorithms $\mathsf{NIZK} = (\mathsf{KGen}, P, \mathsf{Vf})$, where the key generation algorithm KGen produces a common reference string CRS, the prover P on a CRS, a statement x and a witness ω returns a proof Π that $x \in L_R := \{x | \exists \omega.\ (x, \omega) \in R\}$ (or an error symbol $\perp$) and the verifier Vf on a CRS, a statement x and a proof Π outputs 1 if Π is a correct proof that $x \in L_R$ and 0 otherwise [22, 26].

Definition 17. *(Correctness of a non-interactive zero knowledge scheme). A non-interactive zero knowledge scheme* $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{KGen}, P, \mathsf{Vf})$ *for a relation* R *is* correct*, if for all* $\lambda \in \mathbb{N}$*, all* $x \in L_R$*, all* ω *s.t.* $(x, \omega) \in R$*, and for all* $CRS \in [\mathsf{KGen}(1^\lambda)]$ *the following property holds:* $\mathsf{Vf}(CRS, P, (CRS, x, \omega)) = 1$.

A non-interactive zero-knowledge proof system for a relation R is *sound* if no malicious prover can construct a proof for a wrong statement ($x \notin L_R$) for which the verification succeeds.

Definition 18. *(Soundness of a NIZK scheme). Let* $\mathsf{NIZK} = (\mathsf{KGen}, P, \mathsf{Vf})$ *be a non-interactive zero knowledge scheme for a relation* R*.* NIZK *is* sound*, if for all* $\lambda \in \mathbb{N}$*, all* $x \notin L_R$*, and for all ppt* $\mathcal{A}$*, the following probability is negligible in* λ*.*

$$Pr[\mathsf{Vf}(CRS, x, \Pi^*) = 1 \mid CRS \leftarrow \mathsf{KGen}(1^\lambda), \Pi^* \leftarrow \mathcal{A}(CRS, x)]$$

A non-interactive zero-knowledge proof system for a relation R is *zero knowledge* if a proof leaks no information other than the fact that the statement is correct. This is formalized via a simulator that may choose the common reference string CRS itself such that this simulator can produce proofs for arbitrary statements $x \in L_R$ without knowledge of a witness. If those (simulated) proofs are indistinguishable from real proofs, the real proofs can not leak information.

Definition 19. *(Zero knowledge). Let* $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{KGen}, P, \mathsf{Vf})$ *be a proof scheme for a relation* R*.* NIZK *is* zero knowledge*, if for all* $x \in L_R$ *with* $|x| = \lambda$*, and any witness* w *for* x*, there exists a (possibly stateful) efficient simulator* $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$ *such that the following two experiments are computationally indistinguishable for any (possibly stateful) algorithm* $D = (D_0, D_1)$*:*

Game REAL	*Game* Sim
$CRS \leftarrow \mathsf{KGen}_{\mathsf{NIZK}}(1^\lambda)$	$CRS \leftarrow \mathcal{S}_0(1^\lambda)$
$(x, w) \leftarrow D_0(CRS)$	$(x, w) \leftarrow D_0(CRS)$
$\pi \leftarrow \mathcal{P}(CRS, x, w)$	$\pi \leftarrow \mathcal{S}_1(x)$
$b \leftarrow D_1(CRS, x, \pi)$	$b \leftarrow D_1(CRS, x, \pi)$

5.2 Our scheme

Our scheme constitutes a delegatable functional signature scheme as defined in Section 3.1. It is completely general with respect to $\mathcal{F}$ and $\mathcal{G}$ (as long as they are efficiently computable) with the exception that $\mathcal{G}$ may allow only for up to n applications of $\mathsf{Trans}_{\mathcal{FG}}$. We let the signer choose how many applications he allows by defining f as a tuple $(f', k) \in \mathcal{P}_f \times \{0, \dots n\}$.

For our construction to provide the strong notion of privacy under chosen function attack (CFA) for delegatable functional signature schemes (Definition 10), we apply the following trick: If the signer choses a number of k possible applications of $\mathsf{Trans}_{\mathcal{FG}}$, we still create $n+1$ encryptions, but place the encryption a signature on m at the $k+1^{th}$ position (and only encryptions of zero-strings at the other positions). The evaluators fill up the the encryptions from the k^{th} position to the first one. Although each evaluator receives information from his predecessor in the chain of delegations (the first evaluator will know, that the signature originates from the signer), even the second evaluator in the chain will be unable to find out more than its predecessor and the number of applications of $\mathsf{Trans}_{\mathcal{FG}}$ that are still allowed. Figure 6 shows the construction in more detail.

Given a signature scheme $S = (\mathsf{Setup}_\mathcal{S}, \mathsf{KGen}_\mathcal{S}, \mathsf{Sig}_\mathcal{S}, \mathsf{Vf}_\mathcal{S})$ with a simple key generation algorithm and with signatures of equal length, an encryption scheme $\mathcal{E} = (\mathsf{Setup}_\mathcal{E}, \mathsf{KGen}_\mathcal{E}, \mathsf{Enc}_\mathcal{E}, \mathsf{Dec}_\mathcal{E})$ and a zero-knowledge scheme $\mathsf{NIZK} = (\mathsf{KGen}_{\mathsf{NIZK}}, P_{\mathsf{NIZK}}, \mathsf{Vf}_{\mathsf{NIZK}})$ for languages in NP we construct a delegatable functional signature scheme DFSS as follows:

We define a recursive class of languages L_i.
L_n: $x = (pp_\mathcal{S}, pp_\mathcal{E}, pk_{sig}, pk_\mathcal{F}, S, m, CRS, f, g, \sigma) \in L_n$ means that there exists a witness $\omega = (r, k)$ such that

$$pk_{sig} = (vk_\mathcal{S}, \widetilde{ek}) \quad \bigwedge \quad s_k = \mathsf{Enc}_\mathcal{E}(pp_\mathcal{E}, \widetilde{ek}, (\sigma, (f, g, m, pk_\mathcal{F}, k)); r) \bigwedge \quad \mathsf{Vf}_\mathcal{S}(pp_\mathcal{S}, vk_\mathcal{S}, (f, g, m, pk_\mathcal{F}, k), \sigma) = 1$$

L_i for $0 \leq i < n$: $x = (pp_{\mathcal{S}}, pp_{\mathcal{E}}, pk_{sig}, pk_{\mathcal{F}}, S, m, CRS, f, g, \sigma) \in L_i$ if there exists a witness $\omega = (r, \Pi, pk'_{\mathcal{F}}, m', f', \alpha, \beta)$ s.t.

$$
\begin{aligned}
& pk_{sig} = (vk_{\mathcal{S}}, \widetilde{ek}) \\
\bigwedge \quad & s_i = \mathsf{Enc}_{\mathcal{E}}(pp_{\mathcal{E}}, \widetilde{ek}, (\sigma, (f, g, m, pk_{\mathcal{F}}, k)); r) \\
\bigwedge \quad & \mathsf{Vf}_{\mathcal{S}}(pp_{\mathcal{S}}, vk', (f, g, m, pk_{\mathcal{F}}, k), \sigma) = 1 \\
\bigwedge \quad & x' := (pp_{\mathcal{S}}, pp_{\mathcal{E}}, pk_{sig}, pk'_{\mathcal{F}}, S', m', CRS, f', g) \\
\bigwedge \quad & S = S' \left\{ s'_i := s_i \right\} \bigwedge pk'_{\mathcal{F}} = (vk', \cdot) \\
\bigwedge \quad & m = \mathcal{F}(\lambda, f', \alpha, m') \bigwedge f = \mathcal{G}(\lambda, g, \beta, f') \\
\bigwedge \quad & \left(\mathsf{Vf}_{\mathsf{NIZK}_{i+1}}(CRS, x') = 1 \vee \mathsf{Vf}_{\mathsf{NIZK}_n}(CRS, x') = 1 \right)
\end{aligned}
$$

The signer proves that $x = (pp = (CRS, pp_{\mathcal{S}}, pp_{\mathcal{E}}), pk_{sig},\ pk_{\mathcal{F}} = (vk_{\mathcal{F}}, ek), S, d, m) \in L$, where L contains tuples for which there exists a witness $\omega = (f, g, i, r_d)$ such that

$$\mathsf{Vf}_{\mathsf{NIZK}_i}(CRS, (pp_{\mathcal{S}}, pp_{\mathcal{E}}, pk_{sig}, pk_{\mathcal{F}}, S, m, CRS, f, g, \sigma)) = 1$$

$$\bigwedge d \leftarrow \mathsf{Enc}_{\mathcal{E}}(pp_{\mathcal{E}}, ek, (f, g, i, \sigma); r_d)$$

5.2.1 Security

Concerning security, we prove the following theorems.

Theorem 1. *If $\mathcal{E}$ is a correct public key encryption scheme, $\mathcal{S}$ a length preserving unforgeable signature scheme with a simple key generation, and NIZK is a sound non-interactive proof scheme (Definition 18), then the construction* DFSS *presented in this section is* unforgeable against outsider and (strong) insider attacks *according to Definition 9.*

We will show the theorem via a reduction proof. Given an adversary $\mathcal{A}$ that can break our construction we will show that there must be an adversary $\mathcal{B}$ that breaks the underlying signature scheme (with a smaller, but non-negligible probability) — or the encryption scheme was not correct or NIZK was not sound.

Proof for Theorem 1. By Proposition 1 it suffices to show unforgeability against a S-Insider adversary. Assume towards contradiction that DFSS is not unforgeable against strong insider attacks. Then there exists an efficient adversary $\mathcal{A}_{\text{S-Insider}}$ that makes at most $p(\lambda)$ many steps for a polynomial p and that wins the game $\mathsf{Unf}(\mathsf{DFSS}, \mathcal{F}, \mathcal{G}, \mathcal{A}_{\text{S-Insider}}, \lambda)$, formalized in Definition 9, with non-negligible probability.

For simplicity we will write $\mathcal{A}$ for $\mathcal{A}_{\text{S-Insider}}$ in this proof. Since $\mathcal{A}$ makes at most $p(\lambda)$ many steps, $\mathcal{A}$ invokes the oracle Query[KgenP] at most $p(\lambda)$ many times. We show how to build an adversary $\mathcal{B}$ that runs $\mathcal{A}$ in a black-box way in order to break the unforgeability of $\mathcal{S}$ with non-negligible probability. In the following we denote the values and the oracles that the challenger $\mathcal{C}$ from the game $\mathsf{Unf}(S, \mathcal{B}, \lambda)$ provides to $\mathcal{B}$ with the index $\mathcal{C}$.

The algorithm $\mathcal{B}$, upon receiving as input a tuple $(pp_{\mathcal{C}}, vk_{\mathcal{C}})$ from $\mathsf{Initialize}_{\mathcal{C}}$, simulates a challenger for the game $\mathsf{Unf}(\mathsf{DFSS}, \mathcal{F}, \mathcal{G}, \mathcal{A}, \lambda)$. First, the algorithm $\mathcal{B}$ generates the public parameters and the

master public/private key-pair, computing $(pp_{\mathcal{E}}, msk_{\mathcal{E}}) \leftarrow \mathsf{Setup}_{\mathcal{E}}(1^\lambda)$, $CRS \leftarrow \mathsf{KGen}_{\mathsf{NIZK}}(1^\lambda)$and setting $pp := (CRS, pp_{\mathcal{C}}, pp_{\mathcal{E}})$, $msk := (\epsilon, msk_{\mathcal{E}})$.

Subsequently, $\mathcal{B}$ computes $(\widetilde{dk}, \widetilde{ek}) \leftarrow \mathsf{KGen}_{\mathcal{E}}(pp_{\mathcal{E}}, msk_{\mathcal{E}})$, $(sk_{\mathcal{S}}, vk_{\mathcal{S}}) \leftarrow \mathsf{KGen}_{\mathcal{S}}(pp_{\mathcal{C}}, \varepsilon)$ and sets $pk_{sig} := vk_{\mathcal{S}}$.

The algorithm $\mathcal{B}$ embeds its own challenge key $vk_{\mathcal{C}}$ in a randomly chosen position $z \in \{0, \ldots, p(\lambda)\}$; if $z = 0$, then $\mathcal{B}$ replaces $vk_{\mathcal{S}}$ by $vk_{\mathcal{C}}$. Finally, $\mathcal{B}$ runs a black-box simulation of $\mathcal{A}$ on input (pp, pk_{sig}), where $pk_{sig} = vk_{\mathcal{S}}$ or $pk_{sig} = vk_{\mathcal{C}}$, depending on z and $\mathcal{B}$ simulates the oracles $\mathsf{Query[Sign]}$, $\mathsf{Query[Trans]}$, $\mathsf{Query[KGenP]}$ and $\mathsf{Query[Finalize]}$. The algorithm $\mathcal{B}$ handles the oracle queries from $\mathcal{A}$ as follows:

KGenP(): The algorithm $\mathcal{B}$ answers the i^{th} invocation of $\mathsf{Query[KgenP]}$ as follows. First, $\mathcal{B}$ generates a key pair for encryption and decryption $(dk, ek) \leftarrow \mathsf{KGen}_{\mathcal{E}}(pp_{\mathcal{E}}, msk_{\mathcal{E}})$. Then it behaves differently depending on i:

If $i = z$, then $\mathcal{B}$ sends $vk_{\mathcal{C}}$ to $\mathcal{A}$. Otherwise, $\mathcal{B}$ generates a new key-pair $(sk_{\mathcal{F}}, pk_{\mathcal{F}}) \leftarrow \mathsf{KGen}_{\mathcal{S}}(pp_{\mathcal{C}}, \varepsilon)$, stores this pair, and sends $pk_{\mathcal{F}}$ to $\mathcal{A}$.

Sign($\boldsymbol{pk}^*_{\mathcal{F}}, f, g, m$): If $z \neq 0$, the algorithm $\mathcal{B}$ computes all necessary values locally exactly as a challenger for $\mathsf{Unf}(\mathsf{DFSS}, \mathcal{F}, \mathcal{G}, \mathcal{A}, \lambda)$ would. For computing the values locally, $\mathcal{B}$ needs to know pp (publicly known), $sk_{sig} = (ssk_{\mathcal{S}}, vk_{\mathcal{S}})$ (generated by $\mathcal{B}$ since $z \neq 0$) and the values $pk^*_{\mathcal{F}}, f, g$ and m (provided to $\mathcal{B}$ by $\mathcal{A}$.

If $z = 0$, this local computation is not possible since $\mathcal{B}$ replaced $vk_{\mathcal{S}}$ with $vk_{\mathcal{C}}$. Thus, the algorithm $\mathcal{B}$ sets $h_k := (f, g, m, pk_{\mathcal{F}}, k)$ and invokes $\mathsf{Query[Sig]}_{\mathcal{C}}(h_k)$. It sets σ_k to the output of the challenger and otherwise proceeds as above.

Trans($\boldsymbol{pk}^*_{\mathcal{F}}, \alpha, \beta, m, \boldsymbol{pk}'_{\mathcal{F}}, \sigma$): Parse $pk^*_{\mathcal{F}} = (vk, ek)$. $\mathcal{B}$ behaves differently depending on the value of vk.

If $vk \neq vk_{\mathcal{C}}$, $\mathcal{B}$ computes all necessary values locally exactly as a challenger for $\mathsf{Unf}(\mathsf{DFSS}, \mathcal{F}, \mathcal{G}, \mathcal{A}, \lambda)$ would. For computing the values locally, $\mathcal{B}$ needs to know pp (publicly known), a value for $sk^*_{\mathcal{F}}$ corresponding to $pk^*_{\mathcal{F}}$ (discussed below), pk_{sig} (known to $\mathcal{B}$) and the values for $\alpha, \beta, m, pk'_{\mathcal{F}}$ and σ (provided by $\mathcal{A}$). There are four cases for $sk^*_{\mathcal{F}}$. If $pk^*_{\mathcal{F}}$ was output by $\mathsf{Query[KGenP]}$ (and since $vk \neq vk_{\mathcal{C}}$, this was not the z^{th} invocation of $\mathsf{Query[KGenP]}$), $\mathcal{B}$ has generated the value $sk^*_{\mathcal{F}} = (ssk_{\mathcal{F}}, dk)$ itself. The same applies if $pk^*_{\mathcal{F}}$ was output by $\mathsf{Query[KGenS]}$. If $pk^*_{\mathcal{F}}$ was registered by $\mathcal{A}$ via $\mathsf{Query[RegKey]}$, $\mathcal{B}$ uses the corresponding (registered) key $sk^*_{\mathcal{F}}$. If none of the three cases applies, then the key $pk^*_{\mathcal{F}}$ is unknown and $\mathcal{B}$ returns $\perp$ instead.

If $vk = vk_{\mathcal{C}}$, a corresponding value $ssk_{\mathcal{F}}$ (the first part of the secret key $sk^*_{\mathcal{F}}$ corresponding to $pk^*_{\mathcal{F}}$) is not known to $\mathcal{B}$. This key is necessary to sign the value $h = (\hat{f}, g, \hat{m}, pk'_{\mathcal{F}}, k-1)$. Thus, instead of computing a signature with some key $ssk_{\mathcal{F}}$, $\mathcal{B}$ calls its own oracle $\mathsf{Query[Sig]}_{\mathcal{C}}(h)$ and otherwise proceeds as above.

Finalize($m^*, \sigma^*, \boldsymbol{pk}^*_{\mathcal{F}}$): Eventually, $\mathcal{A}$ invokes $\mathsf{Finalize}$ on a tuple $(m^*, \sigma^*, pk^*_{\mathcal{F}})$, then $\mathcal{B}$ parses $\sigma^* = (S, d, \pi)$ with $S = (s_0, \ldots, s_{n+1})$. Now, the algorithm $\mathcal{B}$ checks the validity of the signature computing $\mathsf{Vf}(pp, pk_{sig}, pk^*_{\mathcal{F}}, m^*, \sigma^*)$. If the verification algorithm outputs 0, then $\mathcal{B}$ stops. Otherwise $\mathcal{B}$ decrypts all signatures $(\sigma_i, h_i) := \mathsf{Dec}_{\mathcal{E}}(pp_{\mathcal{E}}, \widetilde{dk}, s_i)$. $\mathcal{B}$ tries to find a pair (σ_x, h_x) that verifies under the key $vk_{\mathcal{C}}$ and that has not been sent to $\mathsf{Query[Sign]}_{\mathcal{C}}$ by $\mathcal{B}$, , then $\mathcal{B}$ sends (h_x, σ_x) to its own $\mathsf{Finalize}_{\mathcal{C}}$ oracle. Otherwise it halts.

Claim 1. *The algorithm $\mathcal{B}$ is efficient.*

Proof for Claim 1. The algorithm $\mathcal{B}$ simulates a challenger for $\mathsf{Unf}(\mathsf{DFSS}, \mathcal{F}, \mathcal{G}, \mathcal{A}, \lambda)$ that consists of efficient algorithms (except for the algorithm $\mathsf{Finalize}$).

$\mathcal{B}$ performs local computations to initialize the game. All algorithms of the underlying schemes that are used in a black-box manner are efficient (key generation, signature creation and verification, encryption and decryption, proof and verify). $\mathcal{B}$ also performs a black-box simulation of the (polynomially bounded) algorithm $\mathcal{A}$ and answers $\mathcal{A}$'s queries. Both the simulation of $\mathcal{A}$ and all of the (polynomially many) computations of answers to oracle calls are efficient.

The only part of the simulation of a challenger for $\mathsf{Unf}(\mathsf{DFSS}, \mathcal{F}, \mathcal{G}, \mathcal{A}, \lambda)$ that might not be efficiently possible is the $\mathsf{Finalize}$ algorithm. However, if eventually $\mathcal{A}$ calls $\mathsf{Finalize}$, then $\mathcal{B}$ diverges from the simulation of a challenger in that $\mathcal{B}$ does not check whether the supposed forgery is in the transitive hull of a specific signature. Thus, $\mathcal{B}$ is an efficient algorithm. □

Claim 2. *The algorithm $\mathcal{B}$ perfectly simulates a challenger for* $\mathsf{Unf}(\mathsf{DFSS}, \mathcal{F}, \mathcal{G}, \mathcal{A}, \lambda)$.

Proof for Claim 2. We investigate the simulation of all oracles and local computations.

Simulation of Initialize: Observe that by construction and by the fact that $\mathcal{S}$ has a simple key generation as in Definition 13 the values pp and msk are identically distributed to values for pp and msk generated by a challenger for $\mathsf{Unf}(\mathsf{DFSS}, \mathcal{F}, \mathcal{G}, \mathcal{A}, \lambda)$. Thus, the keys generated out of them are also identically distributed. If $z \neq 0$ then $\mathcal{B}$ uses only pp and msk to compute the keys (sk_{sig}, pk_{sig}) and thus they are identically distributed as keys (sk_{sig}, pk_{sig}) generated by $\mathsf{Unf}(\mathsf{DFSS}, \mathcal{F}, \mathcal{G}, \mathcal{A}, \lambda)$.

If $z = 0$, then $\mathcal{B}$ replaces the verification $vk_{\mathcal{S}}$ of the signer with the verification key $vk_{\mathcal{C}}$ of the challenger. However, since $\mathcal{S}$ has a simple key generation algorithm (Definition 13), the key $vk_{\mathcal{C}}$ is identically distributed as the key $vk_{\mathcal{S}}$. Moreover, $\mathcal{B}$ does not use the corresponding signing key $ssk_{\mathcal{S}}$ in any way and queries its own signing oracle instead.

Simulation of Query[KGenP]: On any but the z^{th} invocation, $\mathcal{B}$ perfectly simulates a challenger for $\mathsf{Unf}(\mathsf{DFSS}, \mathcal{F}, \mathcal{G}, \mathcal{A}, \lambda)$ and computes a new key pair based on pp and msk. As pp and msk are identically distributed as for a challenger, the resulting keys are also identically distributed.

On the z^{th} invocation, however, $\mathcal{B}$ replaces the verification key $vk_{\mathcal{F}}$ with the verification key $vk_{\mathcal{C}}$ of the challenger. However, since $\mathcal{S}$ has a simple key generation algorithm (Definition 13), the key $vk_{\mathcal{C}}$ is identically distributed as the key $vk_{\mathcal{S}}$. Moreover, $\mathcal{B}$ does not use the corresponding signing key $ssk_{\mathcal{F}}$ in any way and queries its own signing oracle instead.

Simulation of Query[KGenS]: $\mathcal{B}$ uses the values pp and msk that are identically distributed to the corresponding values of a challenger for $\mathsf{Unf}(\mathsf{DFSS}, \mathcal{F}, \mathcal{G}, \mathcal{A}, \lambda)$. On them it performs a perfect simulation of $\mathsf{Query[KGenS]}$. Thus, the resulting keys have the same distribution as the keys output by $\mathsf{Query[KGenS]}$ of the challenger.

Simulation of Query[RegKey]: This oracle does not return an answer.

Simulation of Query[Sign] and Query[Trans]: $\mathcal{B}$ perfectly simulates these oracles as long as it does not have to create a signature with the key corresponding to $vk_{\mathcal{C}}$. However, in these cases $\mathcal{B}$ calls its own signature oracle. Since the keys are identically distributed, this still is a perfect simulation.

Since all messages that $\mathcal{B}$ sends to $\mathcal{A}$ are identically distributed to the messages that $\mathsf{Unf}(\mathsf{DFSS}, \mathcal{F}, \mathcal{G}, \mathcal{A}, \lambda)$ sends to $\mathcal{A}$, the algorithm $\mathcal{B}$ perfectly simulates a challenger for $\mathsf{Unf}(\mathsf{DFSS}, \mathcal{F}, \mathcal{G}, \mathcal{A}, \lambda)$. □

Claim 3. *Whenever $\mathcal{A}$ produces a forgery, then with probability at least $\frac{1}{p(\lambda)+1}$ $\mathcal{B}$ also produces a forgery.*

Proof of Claim 3. First we show the following statement: Whenever $\mathcal{A}$ produces a forgery $(m^*, \sigma^*, pk^*_{\mathcal{F}})$, then $\sigma^* = (S, d, \pi)$ and S contains the encryption of a signature σ_x that verifies under a key vk^* that either equals pk_{sig} or that has been sent to $\mathcal{A}$ as an answer to an oracle query $\mathsf{Query[KGenP]}$, for a message that has not been sent to $\mathsf{Query[Sign]}$ or achieved as result of $\mathsf{Query[Trans]}$.

Assume that $\mathcal{A}$ invokes $\mathsf{Finalize}$ with $(m^*, \sigma^*, pk^*_{\mathcal{F}})$ such that $(m^*, \sigma^*, pk^*_{\mathcal{F}})$ constitutes a forgery for DFSS. Technically: If our algorithm $\mathcal{B}$ would simulate the $\mathsf{Finalize}$ algorithm (as in Figure 7), it would output 1.[1]

If $\mathsf{Finalize}$ would output 1, $(\cdot, \cdot, m^*, \cdot, \cdot) \notin \mathcal{Q}$. This especially means that σ^* can not be output of $\mathsf{Query[Sign]}$ or $\mathsf{Query[Trans]}$. Moreover, there was no query to $\mathsf{Query[Sign]}(pk'_{\mathcal{F}}, f, g, m)$ for an adversary key $pk'_{\mathcal{F}}$ such that m^* is in the transitive hull $\mathcal{F}^*_{\mathcal{G}}(\lambda, f, g, m)$. Also, there was no query to $\mathsf{Query[Trans]}(pk_{\mathcal{F}}, \alpha, \beta, m, pk'_{\mathcal{F}}, \sigma')$ for an adversary key $pk'_{\mathcal{F}}$ such that (f, g) were extracted from σ' and such that m^* is in the transitive hull $\mathcal{F}^*_{\mathcal{G}}(\lambda, \mathcal{G}(\lambda, g, \beta, pk'_{\mathcal{F}}, f), g, \mathcal{F}(\lambda, f, \alpha, m))$.

If the NIZK Π verifies then there is a signature that verifies under pk_{sig} and that marks the start of the delegation chain. Let σ_k be this signature for a value $h_k = (f, g, m, pk_{\mathcal{F}}, k)$. The NIZK makes sure that m^* is in the transitive hull $\mathcal{F}^*_{\mathcal{G}}(\lambda, f, g, m)$ and that all transformations are legitimized by the previous ones (depending on the intermediate β's).

We distinguish the following cases:

$i = 0$: There was no call to $\mathsf{Query[Sig]}$ with parameters $(pk_{\mathcal{F}}, (f, k), g, m)$. Thus, $\mathcal{B}$ never sent h_k to $\mathsf{Query[Sig]}_{\mathcal{C}}$. and thus, S contains a signature $\sigma_x = \sigma_k$ that verifies with pk_{sig} for the message h_k.

$0 < i < k$: There was a call to $\mathsf{Query[Sig]}$ with parameters $(pk_{\mathcal{F}}, (f, k), g, m)$. And for all $0 < j \leq i$ there was a call to $\mathsf{Query[Trans]}$ with parameters $(pk_{\mathcal{F}j}, \alpha_j, \beta_j, m_j, pk'_{\mathcal{F}j}, \sigma'_j)$, such that $h_{k-j} = (f_j, g, m_j, pk'_{\mathcal{F}j}, k-j)$ with $f_j = \mathcal{G}(\lambda, g, \beta_j, pk'_{\mathcal{F}j}, f_{j-1})$, $m_j = \mathcal{F}(\lambda, f_{j-1}, g, m_{j-1})$, but there was no call to $\mathsf{Query[Trans]}$ with parameters $(pk_{\mathcal{F}i}, \alpha_i, \beta_i, m_i, pk'_{\mathcal{F}i}, \sigma'_i)$, such that $h_{k-i} = (f_i, g, m_i, pk'_{\mathcal{F}i}, k-i)$ with $f_i = \mathcal{G}(\lambda, g, \beta_i, pk'_{\mathcal{F}i}, f_{i-1})$, $m_i = \mathcal{F}(\lambda, f_{i-1}, g, m_{i-1})$, where $f_0 = f$ and $m_0 = m$.

Thus, $\mathcal{B}$ never sent h_i to $\mathsf{Query[Sig]}_{\mathcal{C}}$ and thus, σ_i and h_i fulfill our claim.

$i = k$: There was a call to $\mathsf{Query[Sig]}$ with parameters $(pk_{\mathcal{F}}, (f, k), g, m)$. And for all $0 < j \leq k$ there was a call to $\mathsf{Query[Trans]}$ with parameters $(pk_{\mathcal{F}j}, \alpha_j, \beta_j, m_j, pk'_{\mathcal{F}j}, \sigma'_j)$, such that $h_{k-j} = (f_j, g, m_j, pk'_{\mathcal{F}j}, k-j)$ with $f_j = \mathcal{G}(\lambda, g, \beta_j, pk'_{\mathcal{F}j}, f_{j-1})$, $m_j = \mathcal{F}(\lambda, f_{j-1}, g, m_{j-1})$. The NIZK makes sure that at most k transformations of the original message exist. Thus, all transformations have been done via calls to $\mathsf{Query[Trans]}$, which means that $(m^*, \sigma^*, pk^*_{\mathcal{F}})$ is not a forgery.

Thus, each forgery of $\mathcal{A}$ constitutes a forgery of a signature σ_x that verifies with a key vk^* that either equals pk_{sig} or a key that has been given to $\mathcal{A}$ as answer to an oracle query $\mathsf{Query[KGenP]}$.

[1] Note that simulating $\mathsf{Finalize}$ is not necessarily possible in polynomial time, which is of no concern, since $\mathcal{B}$ does not simulate $\mathsf{Finalize}$.

Note that if, by chance, $vk^* = vk_{\mathcal{C}}$, then σ_x is a valid forgery for the message h_x. By Claim 2, $\mathcal{B}$ performs a perfect simulation of a challenger for $\mathsf{Unf}(\mathsf{DFSS}, \mathcal{F}, \mathcal{G}, \mathcal{A}, \lambda)$ (from $\mathcal{A}$'s point of view), independent of the value z that $\mathcal{B}$ has chosen in the beginning. As $vk_{\mathcal{C}}$ is randomly placed in the set of possible honest keys ($p(\lambda)$ many), $\mathcal{B}$ produces a forgery for $vk_{\mathcal{C}}$ with probability at least $\frac{1}{p(\lambda)+1}$. □

For the analysis of the success of $\mathcal{B}$ let us assume that $\mathcal{A}$ produces a forgery with a non-negligible probability. However, by Claim 3, whenever $\mathcal{A}$ produces a forgery, there is a chance of $\frac{1}{p(\lambda)+1}$ that $\mathcal{B}$ will produce a forgery. Since $\mathcal{A}$ is assumed to succeed with a non-negligible probability, $\mathcal{B}$ will also succeed with a non-negligible probability, losing a polynomial factor of $p(\lambda) + 1$. By Claim 1, $\mathcal{B}$ is an efficient algorithm. This concludes the proof.

□

Theorem 2. *If $\mathcal{E}$ is a public key encryption scheme that is secure against chosen plaintext attacks (CPA), and the interactive proof scheme NIZK is zero knowledge (Definition 19), then the construction* DFSS *presented in this section is* secure against chosen function attacks (CFA) *as in Definition 10.*

For showing this theorem we will first give a game-based proof for an adversary that only uses the oracle Query[Sign-$\mathcal{F}$] once. We proceed using a hybrid argument that shows that the existence of a successful adversary that makes polynomially many calls to Query[Sign-$\mathcal{F}$] implies the existence of a successful adversary that only makes one call.

Proof. Let $\mathsf{DFSS} = (\mathsf{Setup}, \mathsf{KGen}_{sig}, \mathsf{KGen}_{\mathcal{F}}, \mathsf{Sig}, \mathsf{Trans}_{\mathcal{FG}}, \mathsf{Vf})$ be our construction for functionalities $\mathcal{F}$ and $\mathcal{G}$. Assume towards contradiction that DFSS is not secure against chosen function attacks against a strong insider. Then there exists an efficient adversary $\mathcal{A}_{\text{S-Insider}}$ that wins the game $\mathsf{CFA}(\mathsf{DFSS}, \mathcal{F}, \mathcal{G}, \mathcal{A}_{\text{S-Insider}}, \lambda)$ from Definition 10 with non negligible advantage. For simplicity we will write $\mathcal{A}$ for $\mathcal{A}_{\text{S-Insider}}$ in this proof.

Claim 4. *If $\mathcal{A}$ invokes the challenge oracle Query[Sign-$\mathcal{F}$] at most once, then the advantage of $\mathcal{A}$ is negligible.*

Proof for Claim 4. The challenger uses the uniformly distributed value b only when Query[Sign-$\mathcal{F}$] is called. Thus, if $\mathcal{A}$ does not call Query[Sign-$\mathcal{F}$], the advantage of $\mathcal{A}$ is 0.

For the case that $\mathcal{A}$ calls Query[Sign-$\mathcal{F}$] exactly once, we show the claim via a series of indistinguishable games that start with a game where $b = 0$ and end with a game $b = 1$. Our proof shows that all intermediate games are indistinguishable.

Let Game $\mathcal{G}_0$ be the original game from Definition 10 where $b = 0$. As by our claim $\mathcal{A}$ calls Query[Sign-$\mathcal{F}$] only once we will simplify the notation of the game by making the call to Query[Sign-$\mathcal{F}$] explicit. Moreover we make the invokation of Initialize explicit as we will modify it in the following games. The oracles that $\mathcal{A}$ can access (aside from Query[Sign-$\mathcal{F}$]) are as they are formalized in Definition 10.

Notation: We use the following notation for describing the games. We assign a number to each line where the first digit marks the game and the remaining digits the line in this game. Thus, 234 marks the 34^{th} line of game 2. Moreover we do not write down all lines explicitly. All lines that are not explicitly stated are as they were defined in the last game that defined them. If, e.g., we write for Game $\mathcal{G}_1$ the line

$$536\ XYZ$$

this means that GAME $\mathcal{G}_5$ differs from GAME $\mathcal{G}_4$ in line 36 which is replaced by XYZ for GAME $\mathcal{G}_5$.

GAME $\mathcal{G}_0$

$--$Initialize $--$

001 $b := 0$

$-$ Setup $-$

002 $CRS \leftarrow \mathsf{KGen}_{\mathsf{NIZK}}(1^\lambda)$

003 $(msk_{\mathcal{S}}, pp_{\mathcal{S}}) \leftarrow \mathsf{Setup}_{\mathcal{S}}(1^\lambda)$

004 $(msk_{\mathcal{E}}, pp_{\mathcal{E}}) \leftarrow \mathsf{Setup}_{\mathcal{E}}(1^\lambda)$

005 $(\widetilde{dk}, \widetilde{ek}) \leftarrow \mathsf{KGen}_{\mathcal{E}}(pp_{\mathcal{E}}, msk_{\mathcal{E}})$

006 $pp := (CRS, pp_{\mathcal{S}}, pp_{\mathcal{E}}, \widetilde{ek})$

007 $msk := (msk_{\mathcal{S}}, msk_{\mathcal{E}})$

$-$ KGen_{sig} $-$

008 $(ssk_{\mathcal{S}}, vk_{\mathcal{S}}) \leftarrow \mathsf{KGen}_{\mathcal{S}}(pp_{\mathcal{S}}, msk_{\mathcal{S}})$

009 $pk_{sig} := vk_{\mathcal{S}}$

010 $sk_{sig} := (ssk_{\mathcal{S}}, pk_{sig})$

$-$ output of (pp, pk_{sig}) to $\mathcal{A}-$

011 $c \leftarrow \mathcal{A}_1^{O_{pp,msk,ssk_{\mathcal{S}}}}(pp, pk_{sig})$

$--$Query[Sign-$\mathcal{F}$] $--$

012 parse $c = (f_0, g, k, [pk_{\mathcal{F}}, \alpha, \beta]_0^t, t, m_0)$

013 if $pk_{\mathcal{F}}[t] \in \mathcal{K}_{\mathcal{A}} \vee k < t$

014 $out := \bot$

015 if $(\cdot, pk_{\mathcal{F}}[0]) \notin \mathcal{K}_{\mathcal{F}} \quad out := \bot$

016 $\sigma_0 \leftarrow \mathsf{Sig}(pp, sk_{sig}, pk_{\mathcal{F}}[0], f_0, g, k, m_0)$

017 for $i \in \{1, \ldots, t\}$

018 if $\neg\exists sk^*_{\mathcal{F}} = (ssk_{\mathcal{F}}, dk).\ (sk^*_{\mathcal{F}}, pk_{\mathcal{F}}[i]) \in \mathcal{K}_{\mathcal{F}}$

019 $out := \bot$

020 $f_i := \mathcal{G}(\lambda, g, \beta[i], pk_{\mathcal{F}}[i], f_{i-1})$

021 $m_i := \mathcal{F}(\lambda, f_{i-1}, \alpha[i], m_{i-1})$

022 $q_i := (pp, sk^*_{\mathcal{F}}, pk_{sig}, \alpha[i], \beta[i], m_{i-1}, pk_{\mathcal{F}}[i], \sigma_{i-1})$

023 parse $pk_{\mathcal{F}}[i] = (vk'_{\mathcal{F}}, ek')$

024 parse $\sigma_{i-1} = (S, d, \Pi)$

025 $(f, g, i, \varsigma_i) \leftarrow \mathsf{Dec}_{\mathcal{E}}(pp_{\mathcal{E}}, dk, d)$

026 $x = (pp, pk_{sig}, pk_{\mathcal{F}_{i-1}}, S, d, m)$

027 if $pk_{\mathcal{F}} = (vk_{\mathcal{F}}, ek)$ belongs to $sk^*_{\mathcal{F}}$

028 $\wedge\ \mathsf{Vf}_{\mathsf{NIZK};Z_i}(CRS, x, \Pi) = 1$

029 $m_i := \mathcal{F}(\lambda_{i-1}, f, \alpha[i], m)$

030 $\hat{f} := \mathcal{G}(\lambda_{i-1}, g, \beta[i], pk_{\mathcal{F}}[i], f)$

031 $h_{i-1} := (\hat{f}, g, m_{i-1}, pk_{\mathcal{F}}[i], i-1)$

032 $\hat{\varsigma}_{i-1} \leftarrow \mathsf{Sig}_{\mathcal{S}}(pp_{\mathcal{S}}, ssk_{\mathcal{F}}, h_{i-1}; r_{\mathcal{S}})$

033 $s_{i-1} \leftarrow \mathsf{Enc}_{\mathcal{E}}(pp_{\mathcal{E}}, \widetilde{ek}, (\hat{\varsigma}_{i-1}, h_{i-1}); r_s)$

034 $\hat{d} \leftarrow \mathsf{Enc}_{\mathcal{E}}(pp_{\mathcal{E}}, ek', (f, g, i-1, \varsigma_{i-1}); r_d)$

035 $\hat{x} = (pp, pk_{sig}, pk_{\mathcal{F}}[i], S, \hat{d}, m_i)$

036 $\omega = (f, g, i-1, r_d)$

037 with $\omega_{i-1} = (\varsigma_{i-1}, r_{\mathcal{S}}, \Pi, pk_{\mathcal{F}_{i-1}}, m_{i-1}, f, \alpha[i], \beta[i])$

038 $\hat{\Pi} \leftarrow P_{\mathsf{NIZK}}(CRS, x, \omega)$

039 $\sigma_i := (S, \hat{d}, \hat{\Pi})$

040 else

041 $\sigma_i := \bot$

042 if $\sigma_t \neq \bot$

043 $h_{k-t} := (f_t, g, m_t, pk_{\mathcal{F}}[t], k-t)$

044 $\varsigma_{k-t} \leftarrow \mathsf{Sig}_{\mathcal{S}}(pp_{\mathcal{S}}, ssk_{\mathcal{S}}, h_{k-t}; r_{\mathcal{S}})$

045 $s_{k-t} \leftarrow \mathsf{Enc}_{\mathcal{E}}(pp_{\mathcal{E}}, \widetilde{ek}, (\varsigma_{k-t}, h_{k-t}); r_{s_{k-t}})$

046 For $j \in \{0, \ldots, n\} \setminus \{k-t\}$

047 $\varsigma_j := 0^{|\varsigma_{k-t}|}$

048 $h_j := (0^{\ell_p(\lambda)}, 0^{\ell_p(\lambda)}, 0^{\ell_m(\lambda)}, 0^{|pk_{\mathcal{F}}[t]|}, 0)$

049 $s_j \leftarrow \mathsf{Enc}_{\mathcal{E}}(pp_{\mathcal{E}}, \widetilde{ek}, (\varsigma_j, h_j); r_{s_i})$

050 $d \leftarrow \mathsf{Enc}_{\mathcal{E}}(pp_{\mathcal{E}}, ek, (f_t, g, k-t, \varsigma_{k-t}; r_d))$

051 $S := (s_0, \ldots, s_n)$

052 $x := (pp, pk_{sig}, pk_{\mathcal{F}}[t], S, d, m_t)$

053 $\omega := (f, g, n, r_d)$

054 with $\omega_{k-t} := (\sigma_{k-t}, r_{\mathcal{S}}, k)$

055 $\Pi \leftarrow \mathcal{P}(CRS, x, \omega)$

056 $\sigma := (S, d, \Pi)$

057 else

058 $\sigma := \sigma_t$

059 if $out \neq \bot$ then $out := \sigma$

060 $b^* \leftarrow \mathcal{A}_2(out)$

GAME $\mathcal{G}_1$

102 $CRS \leftarrow \mathcal{S}_0(1^\lambda)$

137 $\Pi \leftarrow \mathcal{S}_1(x)$

GAME $\mathcal{G}_2$

228 for $j \in \{0, \ldots, n\}$

GAME $\mathcal{G}_3$

301 $b := 1$

316 for $j \in \{1, \ldots, t-1\}$

324 if $\neg\exists sk_{\mathcal{F}}{}^*.\ (sk_{\mathcal{F}}{}^*, pk_{\mathcal{F}}[t]) \in \mathcal{K}_{\mathcal{F}} \quad out := \bot$

325 $f_t := \mathcal{G}(\lambda, g, \beta[t], pk_{\mathcal{F}}[t], f_{t-1})$

$m_t := \mathcal{F}(\lambda, f_{t-1}, \alpha[t], m_{t-1})$

$q_t := (pp, sk_{\mathcal{F}}{}^*, pk_{sig}, \alpha[t], \beta[t], m_{t-1}, pk_{\mathcal{F}}[t], \sigma_{t-1})$

parse $sk_{\mathcal{F}}{}^* = (ssk_{\mathcal{F}}, dk)$

parse $pk_{\mathcal{F}}[t] = (vk'_{\mathcal{F}}, ek')$

parse $\sigma_{t-1} = (S', d', \Pi')$

$(f, g, i, \varsigma_i) \leftarrow \mathsf{Dec}_{\mathcal{E}}(pp_{\mathcal{E}}, dk, d')$

$x' = (pp, pk_{sig}, pk_{\mathcal{F}}, S', d', m_{t-1})$

if $\mathsf{Vf}_{\mathsf{NIZK};Z_i}(CRS, x', \Pi) \neq 1$ then $out := \bot$

$h_{i-1} := (f_t, g, m_t, pk_{\mathcal{F}}[t]i - 1)$

$\hat{\varsigma}_{i-1} \leftarrow \mathsf{Sig}_{\mathcal{S}}(pp_{\mathcal{S}}, ssk_{\mathcal{F}}, h_{i-1}; r_{\mathcal{S}})$

326 $s_{i-1} \leftarrow \mathsf{Enc}_{\mathcal{E}}(pp_{\mathcal{E}}, \widetilde{ek}, (\hat{\varsigma}_{i-1}, h_{i-1}); r_s)$

334 $\omega = (f, g, i-1, r_d)$

335 with $\omega_{i-1} = (\varsigma_{i-1}, r_{\mathcal{S}}, \Pi', pk_{\mathcal{F}}, m_{t-1}, f, \alpha, \beta)$

336 $\Pi \leftarrow P_{\mathsf{NIZK}}(CRS, \xi, x)$

337 $\sigma := (S, d, \Pi)$

GAME $\mathcal{G}_4$

428 for $j \in \emptyset$

432 $d \leftarrow \mathsf{Enc}_{\mathcal{E}}(pp_{\mathcal{E}}, ek', (f, g, i-1, \varsigma_{i-1}); r_d)$

GAME $\mathcal{G}_5$

501 $CRS \leftarrow \mathsf{KGen}_{\mathsf{NIZK}}(1^\lambda)$

536 $\Pi \leftarrow \mathcal{P}(CRS, x, \omega)$

GAME $\mathcal{G}_0 \Rightarrow$ GAME $\mathcal{G}_1$: Since NIZK is zero knowledge, there exists a (possibly stateful) efficient

simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$. In GAME $\mathcal{G}_1$, Initialize calls this simulator $\mathcal{S}_0$ to compute the common reference string *CRS*, instead of the algorithm $\mathsf{Setup}_{\mathsf{NIZK}}$. Moreover, in Query[Sign-$\mathcal{F}$] we call $\mathcal{S}_1$ to simulate the proof Π instead of computing it by calling the prover $\mathcal{P}$.

Claim 5. GAME $\mathcal{G}_0$ *and* GAME $\mathcal{G}_1$ *are computationally indistinguishable.*

Proof. The indistinguishability follows from the fact that NIZK is zero knowledge (Definition 19). If a PPT distinguisher could distinguish between GAME $\mathcal{G}_0$ and GAME $\mathcal{G}_1$, we could construct an efficient distinguisher for NIZK.

□

GAME $\mathcal{G}_1$ ⇒ GAME $\mathcal{G}_2$: The game GAME $\mathcal{G}_2$ is identical to GAME $\mathcal{G}_1$ except for the fact that now S contains only descriptions of zero-strings: we put encryptions of zero strings in all s_j for $j \in \{0, \ldots, n\}$ instead of leaving an encryption of a signature σ_k together with its message h_k at position k.

Claim 6. GAME $\mathcal{G}_1$ *and* GAME $\mathcal{G}_2$ *are computationally indistinguishable.*

Proof. If the games could be distinguished, then we could break the CPA security of $\mathcal{E}$. We distinguish two cases:

- The simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$ behaves differently. Although the simulatability of the NIZK only is defined for valid statements $x \in L_R$, a simulator that can distinguish with a non-negligible probability between a "normal" S (as in GAME $\mathcal{G}_1$) and an S that consists only of encryptions of zero-strings (as in GAME $\mathcal{G}_2$) can also be used to break the CPA security of $\mathcal{E}$.
- The adversary distinguishes the games. If the adversary is able to distinguish GAME $\mathcal{G}_1$ and GAME $\mathcal{G}_2$ with a non-negligible probability, it can be used to break the CPA security of $\mathcal{E}$.

Thus, GAME $\mathcal{G}_1$ and GAME $\mathcal{G}_2$ are computationally indistinguishable. □

GAME $\mathcal{G}_2$ ⇒ GAME $\mathcal{G}_3$: The difference in the games is that in the beginning, the bit b is set to 1 instead of 0. However, b is never used explicitly in the game.

Claim 7. GAME $\mathcal{G}_2$ *and* GAME $\mathcal{G}_3$ *are computationally indistinguishable.*

Proof. □

GAME $\mathcal{G}_3$ ⇒ GAME $\mathcal{G}_4$:

Claim 8. GAME $\mathcal{G}_3$ *and* GAME $\mathcal{G}_4$ *are computationally indistinguishable.*

Proof. □

GAME $\mathcal{G}_4$ ⇒ GAME $\mathcal{G}_5$:

Claim 9. GAME $\mathcal{G}_4$ *and* GAME $\mathcal{G}_5$ *are computationally indistinguishable.*

Proof. □

□

□

References

[1] T. Acar and L. Nguyen. Revocation for delegatable anonymous credentials. In , *PKC 2011*, volume 6571 of *LNCS*, pages 423–440, Taormina, Italy, Mar. 6–9, 2011. Springer, Berlin, Germany.

[2] J. H. Ahn, D. Boneh, J. Camenisch, S. Hohenberger, A. Shelat, and B. Waters. Computing on authenticated data. In *TCC 2012*, LNCS, pages 1–20. Springer, Berlin, Germany, 2012.

[3] G. Ateniese, D. H. Chou, B. de Medeiros, and G. Tsudik. Sanitizable signatures. In , *ESORICS 2005*, volume 3679 of *LNCS*, pages 159–177, Milan, Italy, Sept. 12–14, 2005. Springer, Berlin, Germany.

[4] L. Bauer, L. Jia, and D. Sharma. Constraining credential usage in logic-based access control. In *Computer Security Foundations Symposium, 2010. CSF'10. IEEE 23st.*, pages 154–168. IEEE Computer Society, 2010.

[5] M. Belenkiy, J. Camenisch, M. Chase, M. Kohlweiss, A. Lysyanskaya, and H. Shacham. Randomizable proofs and delegatable anonymous credentials. In , *CRYPTO 2009*, volume 5677 of *LNCS*, pages 108–125, Santa Barbara, CA, USA, Aug. 16–20, 2009. Springer, Berlin, Germany.

[6] M. Belenkiy, M. Chase, M. Kohlweiss, and A. Lysyanskaya. P-signatures and noninteractive anonymous credentials. In , *TCC 2008*, volume 4948 of *LNCS*, pages 356–374, San Francisco, CA, USA, Mar. 19–21, 2008. Springer, Berlin, Germany.

[7] M. Bellare, C. Namprempre, and G. Neven. Security proofs for identity-based identification and signature schemes. In , *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 268–286, Interlaken, Switzerland, May 2–6, 2004. Springer, Berlin, Germany.

[8] M. Bellare, C. Namprempre, and G. Neven. Security proofs for identity-based identification and signature schemes. *Journal of Cryptology*, 22(1):1–61, Jan. 2009.

[9] M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In , *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426, St. Petersburg, Russia, May 28 – June 1, 2006. Springer, Berlin, Germany.

[10] D. Boneh and D. M. Freeman. Homomorphic signatures for polynomial functions. In , *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 149–168, Tallinn, Estonia, May 15–19, 2011. Springer, Berlin, Germany.

[11] D. Boneh and D. M. Freeman. Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In , *PKC 2011*, volume 6571 of *LNCS*, pages 1–16, Taormina, Italy, Mar. 6–9, 2011. Springer, Berlin, Germany.

[12] S. Brands. Restrictive blinding of secret-key certificates. In , *EUROCRYPT'95*, volume 921 of *LNCS*, pages 231–247, Saint-Malo, France, May 21–25, 1995. Springer, Berlin, Germany.

[13] S. A. Brands. *Rethinking public key infrastructures and digital certificates: building in privacy.* The MIT Press, 2000.

[14] C. Brzuska, H. Busch, Ö. Dagdelen, M. Fischlin, M. Franz, S. Katzenbeisser, M. Manulis, C. Onete, A. Peter, B. Poettering, and D. Schröder. Redactable signatures for tree-structured data: Definitions and constructions. In , *ACNS 10*, volume 6123 of *LNCS*, pages 87–104, Beijing, China, June 22–25, 2010. Springer, Berlin, Germany.

[15] C. Brzuska, M. Fischlin, A. Lehmann, and D. Schröder. Unlinkability of sanitizable signatures. In , *PKC 2010*, volume 6056 of *LNCS*, pages 444–461, Paris, France, May 26–28, 2010. Springer, Berlin, Germany.

[16] J. Camenisch and T. Groß. Efficient attributes for anonymous credentials. In , *ACM CCS 08*, pages 345–356, Alexandria, Virginia, USA, Oct. 27–31, 2008. ACM Press.

[17] J. Camenisch and A. Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In , *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 93–118, Innsbruck, Austria, May 6–10, 2001. Springer, Berlin, Germany.

[18] J. Camenisch and A. Lysyanskaya. A signature scheme with efficient protocols. In , *SCN 02*, volume 2576 of *LNCS*, pages 268–289, Amalfi, Italy, Sept. 12–13, 2002. Springer, Berlin, Germany.

[19] J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In , *CRYPTO 2004*, volume 3152 of *LNCS*, pages 56–72, Santa Barbara, CA, USA, Aug. 15–19, 2004. Springer, Berlin, Germany.

[20] M. Chase, M. Kohlweiss, A. Lysyanskaya, and S. Meiklejohn. Malleable signatures: Complex unary transformations and delegatable anonymous credentials. 2013.

[21] S. E. Coull, M. Green, and S. Hohenberger. Controlling access to an oblivious database using stateful anonymous credentials. In , *PKC 2009*, volume 5443 of *LNCS*, pages 501–520, Irvine, CA, USA, Mar. 18–20, 2009. Springer, Berlin, Germany.

[22] U. Feige, D. Lapidot, and A. Shamir. Multiple non-interactive zero knowledge proofs based on a single random string. In *Foundations of Computer Science, 1990. Proceedings., 31st Annual Symposium on*, pages 308 –317 vol.1, oct 1990.

[23] D. M. Freeman. Improved security for linearly homomorphic signatures: A generic framework. In *PKC 2012*, LNCS, pages 697–714. Springer, Berlin, Germany, 2012.

[24] G. Fuchsbauer and D. Pointcheval. Anonymous proxy signatures. In , *SCN 08*, volume 5229 of *LNCS*, pages 201–217, Amalfi, Italy, Sept. 10–12, 2008. Springer, Berlin, Germany.

[25] S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, Apr. 1988.

[26] J. Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In , *ASIACRYPT 2006*, volume 4284 of *LNCS*, pages 444–459, Shanghai, China, Dec. 3–7, 2006. Springer, Berlin, Germany.

[27] R. Johnson, D. Molnar, D. X. Song, and D. Wagner. Homomorphic signature schemes. In , *CT-RSA 2002*, volume 2271 of *LNCS*, pages 244–262, San Jose, CA, USA, Feb. 18–22, 2002. Springer, Berlin, Germany.

[28] K. Miyazaki and G. Hanaoka. Invisibly sanitizable digital signature scheme. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 91(1):392–402, 2008.

[29] R. Steinfeld, L. Bull, and Y. Zheng. Content extraction signatures. In , *ICISC 01*, volume 2288 of *LNCS*, pages 285–304, Seoul, Korea, Dec. 6–7, 2001. Springer, Berlin, Germany.

[30] P. P. Tsang, M. H. Au, A. Kapadia, and S. W. Smith. Blacklistable anonymous credentials: blocking misbehaving users without ttps. In , *ACM CCS 07*, pages 72–81, Alexandria, Virginia, USA, Oct. 28–31, 2007. ACM Press.

A Preliminaries

Definition 20. *(Strong Unforgeability). A Signature Scheme S is* Strongly Unforgeable *if for all PPT adversaries $\mathcal{A}$ the probability*

$$Pr[\mathrm{SUnf}(S, \mathcal{A}, \lambda) = 1].$$

is negligible in λ.

B Relation to Other Primitives

In the following section we show that functional signature schemes imply several seemingly different signature primitives. Using only black-box access to a given functional signature scheme FSS, we construct (among others) identity based signature schemes, sanitizable signature, and redactable signature schemes. To simplify the exposition and to avoid redundancy, we assume that for every primitive Π, there is an algorithm $\mathsf{Setup}(\lambda)$ that outputs some public components *pp* and (potentially) some secret components *msk* for key generation. This setup algorithm is implemented by calling the $\mathsf{Setup}(\lambda)$ of FSS.

B.1 Signature Schemes

As a warm-up, we show that functional signature schemes imply the notion of standard signature schemes that are clearly non-malleable: Only the signer in possession of the secret signing key can generate signatures on message of its choice. In fact, it is (computationally) hard to construct a signature on *another* message, or (in the case of strong unforgeability) to construct a different signature for the same message. Thus, in the language of functional signatures, a regular signature scheme realizes the empty functionality $\mathcal{F}_\perp$. The easiest idea to build a signature scheme from a functional signature scheme for an arbitrary functionality $\mathcal{F}$ is to withhold the evaluation key $sk_\mathcal{F}$ such that evaluation algorithm cannot be executed.

More formally, let $\mathsf{FSS} = (\mathsf{Setup}, \mathsf{KGen}_{sig}, \mathsf{KGen}_\mathcal{F}, \mathsf{Sig}, \mathsf{Eval}_\mathcal{F}, \mathsf{Vf})$ be a functional signature scheme. A regular signature scheme is a tuple of algorithms $S = (\mathsf{Setup}_S, \mathsf{KGen}_S, \mathsf{Sig}_S, \mathsf{Vf}_S)$. We implement the Setup_S algorithm by the Setup algorithm of FSS. The other algorithms are defined in Figure 8. We set $\mathcal{F}_S$ to an arbitrary functionality $\mathcal{F}$ and require FSS to be $\mathcal{F}_S$ correct. If FSS is unforgeable against outsider attacks for functionality $\mathcal{F}_S$, then S is unforgeable in the classical sense, i.e., a polynomially bounded adversary is not able to generate new signatures by using only the public key *pk*.

Proposition 3. *Given a functional signature scheme* FSS *that is both unforgeable against outsider attacks* (UnfO) *and correct for any functionality $\mathcal{F}$, the (regular) signature scheme S as constructed above is unforgeable.*

Proof. Assume towards contradiction that S is not unforgeable. Then, there exists an efficient adversary $\mathcal{A}$ that has access to a signing oracle and outputs a valid forgery on a fresh message of its choice. We show how to build an algorithm $\mathcal{B}$ that breaks the unforgeability against outsider attacks for $\mathcal{F}$. The algorithm $\mathcal{B}$ receives as input a tuple $(pp, pk_{sig}, pk_{\mathcal{F}})$ from $\mathsf{Initialize}_{\mathsf{UnfO}}$, it sets $(pp, pk := (pk_{sig}, pk_{\mathcal{F}}))$ and runs a black-box simulation of $\mathcal{A}$ on (pp, pk). Whenever $\mathcal{A}$ invokes $\mathsf{Query[Sign]}_{\mathsf{Unf}}$ on a message m, then $\mathcal{B}$ forwards the message to its own signing oracle setting $f = 0^{\lambda}$ and it returns the answer $\sigma \leftarrow \mathsf{Sig}(pp, sk_{sig}, pk_{\mathcal{F}}, 0^{\lambda}, m)$ to $\mathcal{A}$. At some point, the algorithm $\mathcal{B}$ get $\mathsf{Finalize}_{\mathsf{Unf}}(m^*, \sigma^*)$ that it forwards to its own challenger.

For the analysis assume that $\mathcal{A}$ succeeds with non-negligible probability. Observe that $\mathcal{B}$ performs a perfect simulation from $\mathcal{A}$'s point of view and that it runs in polynomial-time because $\mathcal{A}$ is an efficient algorithm. But then, $\mathcal{B}$ forges a functional signature whenever $\mathcal{A}$ does contradicting our initial assumption that FSS is unforgeable. □

Note that if the underlying scheme is *strongly* unforgeable against outsider attacks, then the (regular) signature scheme is also strongly unforgeable.

B.2 Rerandomizable Signature Schemes

In difference to the (regular) signature scheme, a rerandomizable signature scheme allows for changing the underlying randomness of a signature, but not changing the message that is signed. This comes very close to a (regular) scheme that is unforgeable but not strongly unforgeable. The algorithms for a rerandomizable signature scheme are almost identical to the ones of the functional signature scheme. The main difference is, that we restrict the functionality $\mathcal{F}$ such that the signed messages m can not be changed.

More formally, let $\mathsf{FSS} = (\mathsf{Setup}, \mathsf{KGen}_{sig}, \mathsf{KGen}_{\mathcal{F}}, \mathsf{Sig}, \mathsf{Eval}_{\mathcal{F}}, \mathsf{Vf})$ be a functional signature scheme. A rerandomizable signature scheme is a tuple of algorithms $R = (\mathsf{Setup}_R, \mathsf{KGen}_R, \mathsf{Sig}_R, \mathsf{Vf}_R, \mathsf{Rerandomize}_R)$. We implement the Setup_R algorithm by the Setup algorithm of FSS.

Since we want R to allow for rerandomization without allowing to change signed messages, we set $\mathcal{F}_I$ to be the identity function $\mathcal{F}_I := (\lambda, f, \alpha, m) := m$ and require FSS to be $\mathcal{F}_I$ correct. if FSS is also unforgeable against insider attacks for $\mathcal{F}_I$, the rerandomizable scheme is unforgeable as well.

Proposition 4. *Given a functional signature scheme* FSS *that is unforgeable against Insider attacks (*UnfI*) for the functionality $\mathcal{F}$ with $\mathcal{F}(\lambda, f, \alpha, m) := m$, the rerandomizable signature scheme R as constructed above is unforgeable.*

Proof. Given an adversary $\mathcal{A}$ that breaks the unforgeability of R, we construct a new adversary $\mathcal{A}$' that breaks the unforgeability against insider attacks against FSS. Given an adversary $\mathcal{A}$ that breaks the unforgeability of R, we construct an adversary $\mathcal{A}$' against FSS that breaks the unforgeability against insider attacks for $\mathcal{F}$:

- Upon receiving $(pp, sk_{\mathcal{F}}, pk_{sig}, pk_{\mathcal{F}})$ from $\mathsf{Initialize}_{\mathsf{UnfI}}$, we send $(pp, pk := (sk_{\mathcal{F}}, pk_{sig}, pk_{\mathcal{F}}))$ to $\mathcal{A}$. By construction, this behavior is exactly the same as the $\mathsf{Initialize}_{\mathsf{Unf}}$ algorithm for R.

- Upon receiving a query $\mathsf{Query[Sign]}_{\mathsf{Unf}}$ from $\mathcal{A}$, we just forward the query to the *UnfI* challenger with $f = 0^\lambda$. This leads our challenger to generate $\sigma \leftarrow \mathsf{Sig}(pp, sk_{sig}, pk_{\mathcal{F}}, 0^\lambda, m)$ and by forwarding σ to $\mathcal{A}$, we give the same answer, that an honest challenger would have given.
- Upon receiving $\mathsf{Finalize}_{\mathsf{Unf}}(m^*, \sigma^*)$ we simply forward the message (as $\mathsf{Finalize}_{\mathsf{Unfl}}(m^*, \sigma^*)$) to our challenger for Unfl.

Since we correctly model an Unf challenger for R, $\mathcal{A}$ will win against us with a non negligible probability. We now lever this probability to our own success probability:

$$
\begin{aligned}
&\Pr[\mathsf{Unfl}(\mathsf{FSS}, \mathcal{F}, \mathcal{A}', \lambda) = 1] \\
&= \Pr[\mathsf{Finalize}_{\mathsf{Unfl}}(m^*, \sigma^*) = 1 | (m^*, \sigma^*) \leftarrow \mathcal{A}'^{\mathsf{Query[Sign]}_{\mathsf{Unfl}}}(pp, sk_{\mathcal{F}}, pk_{sig}, pk_{\mathcal{F}}); (pp, sk_{\mathcal{F}}, pk_{sig}, pk_{\mathcal{F}}) \leftarrow \mathsf{Initialize}_{\mathsf{Unfl}}(\lambda)] \\
&= \Pr[\mathsf{Finalize}_{\mathsf{Unfl}}(m^*, \sigma^*) = 1 | (m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Query}[\mathcal{A}']}(pp, (sk_{\mathcal{F}}, pk_{sig}, pk_{\mathcal{F}})); (pp, sk_{\mathcal{F}}, pk_{sig}, pk_{\mathcal{F}}) \leftarrow \mathsf{Initialize}_{\mathsf{Unfl}}(\lambda)] \\
&= \Pr[\mathsf{Finalize}_{\mathsf{Unf}}(m^*, \sigma^*) = 1 | (m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Query}[\mathcal{A}']}(pp, pk); (pp, pk) \leftarrow \mathsf{Initialize}_{\mathsf{Unf}}(\lambda)] \\
&= \Pr[\mathsf{Finalize}_{\mathsf{Unf}}(m^*, \sigma^*) = 1 | (m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Query[Sign]}_{\mathsf{Unf}}}(pp, pk); (pp, pk) \leftarrow \mathsf{Initialize}_{\mathsf{Unf}}(\lambda)] \\
&= \Pr[\mathsf{Unf}(R, \mathcal{A}, \lambda) = 1]
\end{aligned}
$$

□

B.3 Identity Based Signatures

An identity based signature scheme allows for using commonly known bit strings ID to serve as (public) verification keys for signatures that were generated by the respective party associated with the ID. We follow the general construction of [8], in which a trusted party generates a master secret key msk and for each party extracts a signing key sk_{ID} from msk and the ID of the respective party.

The basic idea for constructing an identity based signature scheme out of a functional signature scheme is that we first derive a secret signing key sk_{sig} that is never given to any party except for the trusted one. With this signing key, we generate the participants' secret keys by signing the individual participants' ID. This signature σ_{ID} constitutes the secret that a participant can use in order to sign their own messages. Signing is done by modifying the respective signing secret σ_{ID}: We add the (real) message to the (signed) ID and obtain a new signature.

Formally, let $\mathsf{FSS} = (\mathsf{Setup}, \mathsf{KGen}_{sig}, \mathsf{KGen}_{\mathcal{F}}, \mathsf{Sig}, \mathsf{Eval}_{\mathcal{F}}, \mathsf{Vf})$ be a functional signature scheme that is $\mathcal{F}_{\mathsf{IBS}}$-correct for $\mathcal{F}_{\mathsf{IBS}}(\lambda, f, \alpha, I) := (I, \alpha)$ [2]. An identity based signature scheme is a tuple $\mathsf{IBS} = (\mathsf{Setup}_{\mathsf{IBS}}, \mathsf{Extract}_{\mathsf{IBS}}, \mathsf{Sig}_{\mathsf{IBS}}, \mathsf{Vf}_{\mathsf{IBS}})$ of probabilistic algorithms. We implement the $\mathsf{Setup}_{\mathsf{IBS}}$ algorithm by calling the Setup algorithm of FSS. When a participants key is extracted from the master secret key and the participant's ID, we do not send them the signing key, but instead we give out a signature $\sigma_{\mathsf{ID}} = \mathsf{Sig}(pp, sk_{sig}, pk_{\mathcal{F}}, \mathsf{ID})$ on the ID and a evaluator key, by which messages can be added to the signed ID. If we want to sign a message m, we use the signature σ_{ID} and combine it with m using $\mathsf{Eval}_{\mathcal{F}}$.

[2] Actually, $\mathcal{F}_{\mathsf{IBS}}$ is required to be of type $\{0,1\}^{\ell_p(\lambda)} \times \{0,1\}^{\ell_p(\lambda)} \times \{0,1\}^{\ell_m(\lambda)+|\mathsf{ID}|}$ s.t. $\mathcal{F}_{\mathsf{IBS}}(\lambda, f, \alpha, (I, 0^{\ell_m(\lambda)})) := (I, \alpha)$. Whenever the second component of the message is not $0^{\ell_m(\lambda)}$, $\mathcal{F}_{\mathsf{IBS}}$ outputs $\perp$. However, for readability reasons, we do not enforce this type in the notation of the definitions or the proof.

The security of the scheme comes from the fact that σ_{ID} is secure and can not be generated by another participant. Even if a signature $\sigma = \mathsf{Sig}(pp, sk_{sig}, pk_{\mathcal{F}}, (\mathsf{ID}, m))$ is known for some message m, one can not generate a forgery for another message m' because then the first component of the signed message would differ from the identity ID (it would be the pair (ID, m)).

So, if FSS is secure against insider attacks and against outsider attacks for $\mathcal{F}$, then the identity based signature scheme IBS is unforgeable as well.

The algorithms KGen_{sig} and $\mathsf{KGen}_{\mathcal{F}}$ can even be deterministic and produce the same evaluator key every time. The important secret for signing is the signature σ_{ID} of the ID.

Proposition 5 (Security of identity based signatures)**.** *Given a functional signature scheme* FSS *that is unforgeable against Insider attacks (*Unfl*) for functionality the functionality* $\mathcal{F}$ *with* $\mathcal{F}(\lambda, f, \alpha, I) := (I, \alpha)$*, the identity based signature scheme* IBS *as constructed above is unforgeable (*$\mathsf{UnflIBS}$*) as in Definition 22.*

Proof. We assume that identities have a certain, fixed length. Furthermore, assume towards contradiction that IBS is not unforgeable. Then there exists an efficient adversary $\mathcal{A}$ that has access to a signing oracle and outputs a valid forgery on a fresh message or on a fresh identity of its choice. We show how to build an adversary $\mathcal{B}$ that breaks the unforgeability against insider attacks for $\mathcal{F}$. The algorithm $\mathcal{B}$ receives as input a tuple $(pp, pk_{sig}, pk_{\mathcal{F}})$ from $\mathsf{Initialize}_{\mathsf{Unfl}}$, calls $\mathsf{Query[KGenS]}()$ and receives $sk_{\mathcal{F}}$. Then it runs a black-box simulation of $\mathcal{A}$ on $pp' := (pp, pk_{sig}, pk_{\mathcal{F}})$. Whenever $\mathcal{A}$ invokes $\mathsf{Query[Init]}_{\mathsf{UnflIBS}}$ on an identity ID, then $\mathcal{B}$ returns 1 to $\mathcal{A}$. Whenever $\mathcal{A}$ invokes $\mathsf{Query[Corrupt]}_{\mathsf{UnflIBS}}(\mathsf{ID})$, $\mathcal{B}$ sends $\mathsf{Query[Sign]}_{\mathsf{Unfl}}(pk_{\mathcal{F}}, 0^{\lambda}, \mathsf{ID})$ to its signing oracle, receives a signature σ_{ID}, sets $sk_{\mathsf{ID}} := (sk_{\mathcal{F}}, \sigma_{\mathsf{ID}}, \mathsf{ID})$ and returns sk_{ID} to $\mathcal{A}$. Whenever $\mathcal{A}$ invokes $\mathsf{Query[Sign]}$ on a message m for an identity ID, $\mathcal{B}$ sends $\mathsf{Query[Sign]}_{\mathsf{Unfl}}(pk_{\mathcal{F}}, 0^{\lambda}, (\mathsf{ID}, m))$ to its signature oracle and forwards its answer σ to $\mathcal{A}$. At some point, the algorithm $\mathcal{B}$ gets $\mathsf{Finalize}_{\mathsf{UnflIBS}}(m^*, \mathsf{ID}^*, \sigma^*)$ and forwards $((m^*, \mathsf{ID}^*), \sigma^*)$ to its own challenger.

For the analysis assume that $\mathcal{A}$ succeeds with non-negligible probability. Observe that $\mathcal{B}$ performs a perfect simulation from $\mathcal{A}$'s point of view and that it runs in polynomial-time because $\mathcal{A}$ is an efficient algorithm.

$\mathcal{A}$ wins against $\mathsf{UnflIBS}$ if it didn't send $\mathsf{Query[Corrupt]}(\mathsf{ID}^*)$ or $\mathsf{Query[Sign]}(\mathsf{ID}^*, m^*)$ before and if ID^* is a known identity. This corresponds to the condition of the verification of Unfl: Assume for contradiction that $(\mathsf{ID}^*, m^*, \sigma')$ is a forgery for $\mathsf{UnflIBS}$ but $((\mathsf{ID}^*, m^*), \sigma^*)$ is not a forgery for Unfl. If the verification algorithm $\mathsf{Vf}_{\mathsf{IBS}}(pp' = (pp, pk_{sig}, pk_{\mathcal{F}}), \mathsf{ID}^*, m^*, \sigma^*) = 1$, by construction $\mathsf{Vf}(pp, pk_{sig}, pk_{\mathcal{F}}, (\mathsf{ID}^*, m^*), \sigma^*) = 1$.

$\mathcal{B}$ succeeds against Unfl, whenever $\mathcal{A}$ succeeds: $\mathcal{B}$ uses the key $pk_{\mathcal{F}}$ in $\mathsf{Finalize}_{\mathsf{Unfl}}$ and $pk_{\mathcal{F}} \in \mathcal{K}_{\mathcal{C}}$. If $(\cdot, (\mathsf{ID}^*, m^*), \cdot, \cdot) \in \mathcal{M}_{\mathsf{Unfl}}$, then $\mathcal{B}$ made a call to its signature oracle with the message (ID^*, m^*), but then there has been a $\mathsf{Query[Sign]}$ on m^* for identity ID^* by $\mathcal{A}$ and in this case $\mathcal{A}$ does not win.

If $\exists x, f, \alpha \ s.t.\ \mathcal{F}(\lambda, f, \alpha, x) = (\mathsf{ID}^*, m^*)$ and $x \in \mathcal{M}_{\mathsf{Unfl}}$, then by definition of $\mathcal{F}$, $x = \mathsf{ID}^*$. Then $\mathcal{B}$ made a query to its signature oracle for the message ID^*. $\mathcal{B}$ only queries its signature oracle in the following two cases:

- A $\mathsf{Query[Corrupt]}_{\mathsf{UnflIBS}}(\mathsf{ID}^*)$ by $\mathcal{A}$, but then $\mathsf{ID}^* \in \mathcal{U}_{\mathsf{UnflIBS}}$ and thus $\mathsf{ID}^*, m^*, \sigma^*$ is not a forgery for $\mathsf{UnflIBS}$.
- A $\mathsf{Query[Sign]}_{\mathsf{UnflIBS}}(\mathsf{ID}^+, m^+)$ by $\mathcal{A}$ with $\mathsf{ID}^* = (\mathsf{ID}^+, m^+)$. However, since identities have a fixed length, ID^* is not a valid identity and thus $\mathsf{ID}^*, m^*, \sigma^*$ is not a forgery for $\mathsf{UnflIBS}$.

□

B.4 Sanitizable Signatures

We show how to construct a sanitizable signature scheme as in [15] out of a functional signature scheme. However, we deviate from their definition on purpose in order to simplify the construction.

Intuitively, we express the possible sanitization options as a predicate P and implement the predicate by a functionality $\mathcal{F}$. Please note that we construct a "weaker form" of a sanitizable signature scheme that does not have a notion of accountability. However, extending the scheme to an accountable sanitizable signature scheme is an interesting future work.

More formally, let $\mathsf{FSS} = (\mathsf{Setup}, \mathsf{KGen}_{sig}, \mathsf{KGen}_{\mathcal{F}}, \mathsf{Sig}, \mathsf{Eval}_{\mathcal{F}}, \mathsf{Vf})$ anf set $\ell_p(\lambda) = \ell_m(\lambda)$ for all λ. A sanitizable signature scheme is a tuple $\mathsf{SanS} = (\mathsf{Setup}_{\mathsf{SanS}}, \mathsf{KGen}_{si}, \mathsf{KGen}_{sa}, \mathsf{Sig}_{\mathsf{SanS}}, \mathsf{Sanit}_{\mathsf{SanS}}, \mathsf{Vf}_{\mathsf{SanS}})$. We implement the $\mathsf{Setup}_{\mathsf{SanS}}$ algorithm by calling the Setup algorithm of FSS, KGen_{si} by calling KGen_{sig}, KGen_{sa} by calling $\mathsf{KGen}_{\mathcal{F}}$, $\mathsf{Sig}_{\mathsf{SanS}}$ by calling Sig, $\mathsf{Sanit}_{\mathsf{SanS}}$ by calling $\mathsf{Eval}_{\mathcal{F}}$and $\mathsf{Vf}_{\mathsf{SanS}}$ by calling Vf.[3]

For a desired predicate $P : \{0,1\}^{\ell_p(\lambda)} \times \{0,1\}^{\ell_p(\lambda)} \times \{0,1\}^{\ell_m(\lambda)} \to \{0,1\}$ we construct a functionality $\mathcal{F}_P$ as $\mathcal{F}_P(\lambda, f, \alpha, m) := \begin{cases} \alpha \text{ if } P(m, f, \alpha) = 1 \\ \perp \text{ otherwise} \end{cases}$

Proposition 6. *If* FSS *is unforgeable against outsider attacks for* $\mathcal{F}$ *(Definition 4), then* SanS *is unforgeable (Definition 21).*

Proof sketch. As we simply instantiate our scheme with a specific functionality, it follows directly that an adversary $\mathcal{A}$ that breaks the unforgeability of SanS also breaks the unforgeability against outsider attacks of FSS for $\mathcal{F}$.

□

Note that our predicate based definition implies other common definitions for sanitizable signatures (e.g. the MOD/ADM version of [15]) as they can be expressed by a predicate.

B.5 Redactable Signatures

Here we show how to construct redactable signatures as in [14] from functional signatures. Intuitively we can construct a redactable signature scheme RSS from a functional signature scheme by making the secret evaluator key $sk_{\mathcal{F}}$ public. This allows everyone to modify signatures according to a functionality $\mathcal{F}$ that implements the desired redaction predicate P. As for the case of sanitizable signatures we require that the length of $\ell_p(\lambda)$ and $\ell_m(\lambda)$ are equal.

More formally, let $P : \{0,1\}^{\ell_m(\lambda)} \times \{0,1\}^{\ell_m(\lambda)} \to \{0,1\}$ be a predicate and $\mathsf{FSS} = (\mathsf{Setup}, \mathsf{KGen}_{sig}, \mathsf{KGen}_{\mathcal{F}}, \mathsf{Sig}, \mathsf{Eval}_{\mathcal{F}}, \mathsf{Vf})$ be a functional signature scheme that is $\mathcal{F}_P$ correct for a functionality.

$\mathcal{F}_P(\lambda, f, \alpha, m) := \begin{cases} \alpha \text{ if } P(m, \alpha) \\ \perp \text{ otherwise} \end{cases}$

We implement the $\mathsf{Setup}_{\mathsf{RSS}}$ algorithm by calling the Setup algorithm of FSS. We implement the other algorithms as follows:

Proposition 7. *If* FSS *is unforgeable against insider attacks for* $\mathcal{F}$*, then* RSS *is unforgeable.*

Proof sketch. An adversary $\mathcal{A}$ that breaks the unforgeability of RSS also breaks the unforgeability against insider attacks of FSS for $\mathcal{F}$.

□

[3] We require that $\forall\lambda.\ell_p(\lambda) = \ell_m(\lambda)$

C Appendix (Security Notions)

In this section, we review the security notions for the basic cryptographic primitives.

C.1 Sanitizable signature schemes

A sanitizable signature scheme is called unforgeable, if it is computationally hard to forge signatures for new messages without access to the signing key, even if arbitrary messages have been signed before. More formally, we define unforgeability for a signature scheme $\mathsf{SanS} = (\mathsf{Setup}_{\mathsf{SanS}}, \mathsf{KGen}_{\mathsf{SanS}}, \mathsf{Sig}_{\mathsf{SanS}}, \mathsf{Vf}_{\mathsf{SanS}})$ as a game UnfSan *(SanS,$\mathcal{A}$,λ)* with the following game algorithms:

Definition 21 (Unforgeability for Sanitizable Signature Schemes)**.** *A Sanitizable Signature Scheme S is* Unforgeable *if for all PPT adversaries $\mathcal{A}$ the probability*

$$Pr[\mathrm{Unf}(S, \mathcal{A}, \lambda) = 1].$$

is negligible in λ.

Unforgeability for identity based signature schemes We follow the definition of [8, 7], where an identity based signature scheme is called unforgeable, if it is computationally hard to forge signatures for new messages of honest (not corrupted) participants, even if arbitrary messages have been signed for arbitrary participants before and if arbitrary participants were adaptively corrupted. More formally, we define unforgeability for a signature scheme $\mathsf{IBS} = (\mathsf{Setup}_{\mathsf{IBS}}, \mathsf{Extract}_{\mathsf{IBS}}, \mathsf{Sig}_{\mathsf{IBS}}, \mathsf{Vf}_{\mathsf{IBS}})$ as a game UnfIBS *(IBS,$\mathcal{A}$,λ)* as in Figure 11.

Definition 22 (Unforgeability for identity based signatures)**.** *Let $\mathsf{IBS} = (\mathsf{Setup}_{\mathsf{IBS}}, \mathsf{Extract}_{\mathsf{IBS}}, \mathsf{Sig}_{\mathsf{IBS}}, \mathsf{Vf}_{\mathsf{IBS}})$ be an identity based signature scheme. The definition uses $\mathsf{UnfIBS}(\mathsf{IBS}, \mathcal{A}, \lambda)$ defined in Figure 11. IBS is* existential unforgeable $(\mathsf{EU\text{-}IBS})$ *if for all PPT adversaries $\mathcal{A}$*

$$\mathbf{Adv}^{\mathsf{EU\text{-}IBS}}_{\mathsf{IBS},\mathcal{A}} = Pr\left[\mathsf{UnfIBS}(\mathsf{IBS}, \mathcal{A}, \lambda) = 1\right]$$

is negligible in λ,

D Postponed proofs

In this section we present the omitted proofs from the body of the paper.

D.1 Proofs for Section 3

sketch for Theorem 1. We assume that the key generator of the underlying signature scheme $\mathsf{KGen}_{\mathcal{S}}$ does not require a master secret key, as is the case for most signature schemes. Technically we assume that $\mathsf{Setup}_{\mathcal{S}}$ always outputs an empty $msk_{\mathcal{S}} = \varepsilon$

Assume towards contradiction that DFSS is not unforgeable against outsider attacks. Then there exists an efficient adversary $\mathcal{A}$ that has access to a public key oracle, a signing oracle and a transformation oracle and outputs a valid forgery on a fresh message of its choice. We show how to build an adversary $\mathcal{B}$ that breaks the unforgeability of $\mathcal{S}$. The algorithm $\mathcal{B}$ receives as input a

tuple $(pp_{\mathcal{C}}, vk_{\mathcal{C}})$ from $\mathsf{Initialize}_{\mathsf{Unf}}$ and computes $(pp_{\mathcal{E}}, msk_{\mathcal{E}}) \leftarrow \mathsf{Setup}_{\mathcal{E}}(1^\lambda)$, $CRS \leftarrow \mathsf{KGen}_{\mathsf{NIZK}}(1^\lambda)$. $\mathcal{B}$ then sets $pp := (CRS, pp_{\mathcal{C}}, pp_{\mathcal{E}})$ $msk := (\varepsilon, msk_{\mathcal{E}})$. $\mathcal{B}$ generates $(\widetilde{dk}, \widetilde{ek}) \leftarrow \mathsf{KGen}_{\mathcal{E}}(pp_{\mathcal{E}}, msk_{\mathcal{E}})$. If $\mathcal{A}$ is bounded by the polynomial $p(\lambda)$, $\mathcal{B}$ guesses a number $z \in \{0, \ldots, p(\lambda)\}$. Then $\mathcal{B}$ generates a new key pair $(sk_{sig}, pk_{sig}) \leftarrow \mathsf{KGen}_{sig}(pp, msk)$. If $z = 0$, $\mathcal{B}$ replaces $vk_{\mathcal{S}}$ by $vk_{\mathcal{C}}$. $\mathcal{B}$ then runs a black-box simulation of $\mathcal{A}$ on pp and pk_{sig}.

Whenever $\mathcal{A}$ invokes $\mathsf{Query[KgenP]}$, $\mathcal{B}$ generates a new key pair $(sk_{\mathcal{F}}, pk_{\mathcal{F}}) \leftarrow \mathsf{KGen}_{sig}(pp, msk)$, stores it and sends pk_{sig} to $\mathcal{A}$. On the z'th invocation, however, $\mathcal{B}$ replaces $vk_{\mathcal{S}}$ by $vk_{\mathcal{C}}$ before sending pk_{sig} to $\mathcal{A}$.

Whenever $\mathcal{A}$ invokes $\mathsf{Query[Sign]}$ or $\mathsf{Query[Trans]}$, $\mathcal{B}$ computes the required signature locally and forwards its answer σ to $\mathcal{A}$. However, if $vk_{\mathcal{S}} = vk_{\mathcal{C}}$, a local computation of the underlying signature is not possible. In this case, $\mathcal{B}$ sends a query to its signature oracle instead of computing the signature locally.

Whenever $\mathcal{A}$ produces a forgery (m^*, σ^*), $\mathcal{B}$ proceeds as follows: It parses $\sigma^* = (S, d, \Pi)$ and decrypts the signatures $(\sigma_i, h_i) = \mathsf{Dec}(pp_{\mathcal{E}}, \widetilde{dk}, s_i)$. Then it parses the $h_i = (f_i, g, m_i, pk_{\mathcal{F}i}, i)$. If one of the signatures validates with $vk_{\mathcal{C}}$ and $\mathcal{B}$ has not queried h_i to its oracle, $\mathcal{B}$ sends (h_i, σ_i) as a forgery.

For the analysis assume that $\mathcal{A}$ succeeds with non-negligible probability. Observe that $\mathcal{B}$ runs in polynomial-time because $\mathcal{A}$ is an efficient algorithm and that it performs a perfect simulation from $\mathcal{A}$'s point of view.

Claim 1: Whenever $\mathcal{A}$ produces a forgery, then this forgery contains the encryption of a signature that verifies for some key vk^* that either equals pk_{sig} or for which $\mathcal{A}$ has made a call $\mathsf{Query[KgenP]}$. The last signature in the chain (the one signing m^*) can not have been produced by $\mathsf{Query[Sign]}$ or $\mathsf{Query[Trans]}$, as otherwise $(m^*, \sigma^*, pk_{\mathcal{F}}{}^*)$ would not be a valid forgery. As the NIZK does not check which entity generated the keys, the last signature σ_l in the chain does not necessarily validate under a key that has been honestly generated. However, neither $\mathsf{Query[Sign]}$ nor $\mathsf{Query[Trans]}$ allow for delegation to entities with unknown keys and as $\mathcal{A}$ is an outsider, it will never call $\mathsf{Query[KgenS]}$ or $\mathsf{Query[RegisterKey]}$ to obtain secret keys to known public keys. Thus, in this case the previous signature σ_{l+1} must have been forged by $\mathcal{A}$ as well. This argument can be applied recursively. However, the NIZK makes sure that the first message in the chain has been signed with sk_{sig}, so at some point there has to be a signature that was forged by $\mathcal{A}$ and that verifies with either $vk^* = pk_{sig}$ or another vk^* for which $\mathcal{A}$ has made a call to $\mathsf{Query[KgenP]}$.

Note that if, by chance, $vk^* = vk_{\mathcal{C}}$, then (h_i, σ_i) is a valid forgery for $\mathcal{B}$. Recall that $\mathcal{B}$ performs a perfect simulation from $\mathcal{A}$'s point of view independent of the choice of z. This especially means that the choice of z does not influence the behavior of $\mathcal{A}$. As by claim 1 $\mathcal{A}$ produces a forgery for an honest key and $vk_{\mathcal{C}}$ is randomly placed in the set of possible honest keys ($p(\lambda)$ many, as $\mathcal{A}$ makes at most $p(\lambda)$ many steps), $\mathcal{A}$ produces a forgery for $vk_{\mathcal{C}}$ with probability at least $\frac{1}{p(\lambda)+1}$. Since $\mathcal{A}$ is assumed to succeed with a non-negligible probability, $\mathcal{B}$ will also succeed with a non-negligible probability, losing a polynomial factor of $p(\lambda) + 1$.

□

Against a (strong) insider $\mathcal{B}$ reacts to the calls to $\mathsf{Query[KGenS]}$ and $\mathsf{Query[RegisterKey]}$ as a challenger for Unf would. Otherwise $\mathcal{B}$ behaves as before. In the analysis, the argumentation for *claim 1* changes slightly: The NIZK makes sure that whatever $\mathcal{A}$ computes from an honestly generated signature that is included in S has been allowed in this signature: The trace of m's and f's can not be broken without either violating the soundness of the NIZK, or forging a signature for

a key that was output by Query[KGenP]. For the construction presented here, a strong insider does not have more capabilities.

sketch for Theorem 2. We begin with a conceptually simpler case where the adversary $\mathcal{A}$ against the CFA-security only invokes the challenge oracle Query[Sign-$\mathcal{F}$] once. In the following proof, we start with the game where $b := 0$ and via a hop of indistinguishable games we will end up with a game where $b := 1$. Our proof shows that all intermediate games are either indistinguishable such that $\mathcal{A}$'s success probability remains the same (except for a negligible amount). Thus, we conclude that $\mathcal{A}$'s advantage was negligible at the very beginning. Finally, we apply a standard hybrid argument to cover the more general case where $\mathcal{A}$ queries the challenge oracle Query[Sign-$\mathcal{F}$] multiple times.

More formally, let GAME GAME $\mathcal{G}_0$ as shown in Figure 12 be the original game in which $\mathcal{A}$ plays against a challenger for CFA and where $b = 0$ holds. For simplicity we will write down Initialize and give $\mathcal{A}$ orcle access to Query[Sign] (the number of queries to this oracle are only polynomially bounded). To improve the exposition of the proof, we assume that $\mathcal{A}$ is a stateful algorithm consisting of two procedures. The first one, that we denote by $\mathcal{A}_1$, creates a query for the Query[Sign-$\mathcal{F}$] oracle. The second algorithm, denoted by $\mathcal{A}_2$, obtains the response and outputs its guess b.

GAME $\mathcal{G}_0$ ⇒ GAME $\mathcal{G}_1$. We define GAME $\mathcal{G}_1$ as the game in which instead of a (real) prover $\mathcal{P}$ we use the simulator $\mathsf{Sim}_{\mathsf{NIZK}}$ to compute the CRS and also to generate the proofs Π that are requested by $\mathcal{A}$ in all Query[Sign-$\mathcal{F}$] queries. This modification is defined in Figure 12.

Claim 10. GAME $\mathcal{G}_0 \approx$ GAME $\mathcal{G}_1$.

sketch. The games GAME $\mathcal{G}_0$ and GAME $\mathcal{G}_1$ are indistinguishable by the zero knowledge property of the NIZK scheme: If an probabilistic poly time adversary distinguished between both games with a non-negligible probability, we could easily construct an efficient distinguisher, that can distinguish between a real proof and simulation with the same probability. Since this proof is fairly standard, we omit a full reduction. □

GAME $\mathcal{G}_1$ ⇒ GAME $\mathcal{G}_2$. The next games GAME $\mathcal{G}_2$ is identical to the previous game, with the difference being that we replace the encryptions $s_0, \ldots s_n$ by all 0-string encryptions. The game is depicted in Figure 12.

Claim 11. GAME $\mathcal{G}_1 \approx$ GAME $\mathcal{G}_2$.

sketch. Since neither $\mathcal{A}$ nor $\mathsf{Sim}_{\mathsf{NIZK}}$ receive decryptions keys or information about the randomness[4] that has been used for encrypting, the CPA property of the encryption scheme implies that the games GAME $\mathcal{G}_1$ and GAME $\mathcal{G}_2$ are indistinguishable. □

GAME $\mathcal{G}_2$ ⇒ GAME $\mathcal{G}_3$. Now, in GAME $\mathcal{G}_3$ we change the computation of the signature we generate. If $\sigma_t \neq \perp$, then we still generate σ_t as before and use it. In the case that $\sigma_t = \perp$, $\mathcal{B}$ did already set $\sigma := \sigma_t$.

Claim 12. GAME $\mathcal{G}_2 \approx$ GAME $\mathcal{G}_3$.

[4]Since we use a simulator here that has no access to this information, it can not pass it on to the adversary

sketch. However, the signatures are not used in the current game, since we only encrypt zero-strings in S and d and use a simulator that has no access to secret information. Thus, GAME $\mathcal{G}_2$ and GAME $\mathcal{G}_3$ are perfectly indistinguishable for $\mathcal{A}$. □

GAME $\mathcal{G}_3 \Rightarrow$ GAME $\mathcal{G}_4$.

Claim 13. GAME $\mathcal{G}_4 \approx$ GAME $\mathcal{G}_5$.

sketch. For the game GAME $\mathcal{G}_4$ we again replace the fake encryptions by encryptions of the s_i. Using the same argumentation as above we see that the CPA property of the encryption implies that the games GAME $\mathcal{G}_3$ and GAME $\mathcal{G}_4$ are indistinguishable. □

GAME $\mathcal{G}_4 \Rightarrow$ GAME $\mathcal{G}_5$.

Claim 14. GAME $\mathcal{G}_4 \approx$ GAME $\mathcal{G}_5$.

sketch. Finally, we define GAME $\mathcal{G}_5$ as the original game in which $\mathcal{A}$ plays against a challenger for CFA but where $b = 1$ holds. The only difference to GAME $\mathcal{G}_4$ is, that we replace the simulator by a real prover again, that uses a new witness $\hat{\omega}$ to show that the statement holds. Again, if one could distinguish GAME $\mathcal{G}_4$ and GAME $\mathcal{G}_5$ with non negligible probability, we could construct a PPT distinguisher, that can distinguish between a real proof and simulation with the same probability. □

□

E Appendix (Figures)

$\underline{\mathsf{Setup}(1^\lambda):}$

$CRS \leftarrow \mathsf{KGen}_{\mathsf{NIZK}}(1^\lambda)$

$(msk_{\mathcal{S}}, pp_{\mathcal{S}}) \leftarrow \mathsf{Setup}_{\mathcal{S}}(1^\lambda)$

$(msk_{\mathcal{E}}, pp_{\mathcal{E}}) \leftarrow \mathsf{Setup}_{\mathcal{E}}(1^\lambda)$

$(\widetilde{dk}, \widetilde{ek}) \leftarrow \mathsf{KGen}_{\mathcal{E}}(pp_{\mathcal{E}}, msk_{\mathcal{E}})$

$pp := (CRS, pp_{\mathcal{S}}, pp_{\mathcal{E}}, \widetilde{ek})$

$msk := (msk_{\mathcal{S}}, msk_{\mathcal{E}})$

output (pp, msk)

$\underline{\mathsf{KGen}_{sig}(pp, msk):}$

parse $pp = (CRS, pp_{\mathcal{S}}, pp_{\mathcal{E}}, \widetilde{ek})$

parse $msk = (msk_{\mathcal{S}}, msk_{\mathcal{E}})$

$(ssk_{\mathcal{S}}, vk_{\mathcal{S}}) \leftarrow \mathsf{KGen}_{\mathcal{S}}(pp_{\mathcal{S}}, msk_{\mathcal{S}})$

$pk_{sig} := vk_{\mathcal{S}}$

$sk_{sig} := (ssk_{\mathcal{S}}, pk_{sig})$

output (sk_{sig}, pk_{sig})

$\underline{\mathsf{KGen}_{\mathcal{F}}(pp, msk):}$

parse $pp = (CRS, pp_{\mathcal{S}}, pp_{\mathcal{E}}, \widetilde{ek})$

parse $msk = (msk_{\mathcal{S}}, msk_{\mathcal{E}})$

$(ssk_{\mathcal{F}}, vk_{\mathcal{F}}) \leftarrow \mathsf{KGen}_{\mathcal{S}}(pp_{\mathcal{S}}, msk_{\mathcal{S}})$

$(dk, ek) \leftarrow \mathsf{KGen}_{\mathcal{E}}(pp_{\mathcal{E}}, msk_{\mathcal{E}})$

$sk_{\mathcal{F}} := (ssk_{\mathcal{F}}, dk)$

$pk_{\mathcal{F}} := (vk_{\mathcal{F}}, ek)$

output $(sk_{\mathcal{F}}, pk_{\mathcal{F}})$

$\underline{\mathsf{Sig}(pp, sk_{sig}, pk_{\mathcal{F}}, (f,k), g, m):}$

parse $pp = (CRS, pp_{\mathcal{S}}, pp_{\mathcal{E}}, \widetilde{ek})$

parse $pk_{\mathcal{F}} = (vk_{\mathcal{F}}, ek)$

parse $sk_{sig} = (ssk_{\mathcal{S}}, pk_{sig}))$

$h_k := (f, g, m, pk_{\mathcal{F}}, k)$

$\sigma_k \leftarrow \mathsf{Sig}_{\mathcal{S}}(pp_{\mathcal{S}}, ssk_{\mathcal{S}}, h_k; r_{\mathcal{S}})$

$s_k \leftarrow \mathsf{Enc}_{\mathcal{E}}(pp_{\mathcal{E}}, \widetilde{ek}, (\sigma_k, h_k); r_{s_k})$

For $i \in \{0, \ldots, n\} \setminus \{k\}$

  $\sigma_i := 0^{|\sigma_k|}$

  $h_i := (0^{\ell_f(\lambda)}, 0^{\ell_g(\lambda)}, 0^{\ell_m(\lambda)}, 0^{|pk_{\mathcal{F}}|}, 0)$

  $s_i \leftarrow \mathsf{Enc}_{\mathcal{E}}(pp_{\mathcal{E}}, \widetilde{ek}, (\sigma_i, h_i); r_{s_i})$

$d \leftarrow \mathsf{Enc}_{\mathcal{E}}(pp_{\mathcal{E}}, ek, (f, g, k, \sigma_k); r_d)$

$S := (s_0, \ldots, s_n)$

$x = (pp, pk_{sig}, pk_{\mathcal{F}}, S, d, m)$

$\omega = (f, g, n, r_d)$

with $\omega_k = (\sigma_k, r_{\mathcal{S}}, k)$

$\Pi \leftarrow P_{\mathsf{NIZK}}(CRS, x, \omega)$

$\sigma := (S, d, \Pi)$

output σ

$\underline{\mathsf{Vf}(pp, pk_{sig}, pk_{\mathcal{F}}, m, \sigma):}$

parse $pp = (CRS, pp_{\mathcal{S}}, pp_{\mathcal{E}}, \widetilde{ek})$

parse $pk_{sig} = (vk_{\mathcal{S}}, \widetilde{ek})$

parse $pk_{\mathcal{F}} = (vk_{\mathcal{F}}, ek)$

parse $\sigma = (S, d, \Pi)$

$x := (pp_{\mathcal{S}}, pp_{\mathcal{E}}, pk_{sig}, pk_{\mathcal{F}}, S, d, m, CRS)$

$b \leftarrow \mathsf{Vf}_{\mathsf{NIZK}}(CRS, x, \Pi)$

output b

$\underline{\mathsf{Trans}_{\mathcal{FG}}(pp, sk_{\mathcal{F}}, pk_{sig}, \alpha, \beta, m, pk'_{\mathcal{F}}, \sigma):}$

parse $pp = (CRS, pp_{\mathcal{S}}, pp_{\mathcal{E}}, \widetilde{ek})$

parse $sk_{\mathcal{F}} = (ssk_{\mathcal{F}}, dk)$

parse $pk'_{\mathcal{F}} = (vk'_{\mathcal{F}}, ek')$

parse $\sigma = (S, d, \Pi)$

$(f, g, i, \sigma_i) \leftarrow \mathsf{Dec}_{\mathcal{E}}(pp_{\mathcal{E}}, dk, d)$

$x = (pp, pk_{sig}, pk_{\mathcal{F}}, S, d, m)$

if $pk_{\mathcal{F}} = (vk_{\mathcal{F}}, ek)$ belongs to $sk_{\mathcal{F}}$

$\wedge\, \mathsf{Vf}_{\mathsf{NIZK};Z_i}(CRS, x, \Pi) = 1$

  $\hat{m} := \mathcal{F}(\lambda, f, \alpha, m)$

  $\hat{f} := \mathcal{G}(\lambda, g, \beta, pk'_{\mathcal{F}}, f)$

  $h_{i-1} := (\hat{f}, g, \hat{m}, pk'_{\mathcal{F}}, i-1)$

  $\hat{\sigma}_{i-1} \leftarrow \mathsf{Sig}_{\mathcal{S}}(pp_{\mathcal{S}}, ssk_{\mathcal{F}}, h_{i-1}; r_{\mathcal{S}})$

  $s_{i-1} \leftarrow \mathsf{Enc}_{\mathcal{E}}(pp_{\mathcal{E}}, \widetilde{ek}, (\hat{\sigma}_{i-1}, h_{i-1}); r_s)$

  $d \leftarrow \mathsf{Enc}_{\mathcal{E}}(pp_{\mathcal{E}}, ek', (f, g, i-1, \sigma_{i-1}); r_d)$

  $\hat{x} = (pp, pk_{sig}, pk'_{\mathcal{F}}, S, d, \hat{m})$

  $\omega = (f, g, i-1, r_d)$

  with $\omega_{i-1} = (\sigma_{i-1}, r_{\mathcal{S}}, \Pi, pk_{\mathcal{F}}, m, f, \alpha, \beta)$

  $\hat{\Pi} \leftarrow P_{\mathsf{NIZK}}(CRS, x, \omega)$

  $\hat{\sigma} := (S, d, \Pi)$

  output $\hat{\sigma}$

else

  output $\perp$

Figure 6: Construction of a DFSS

PROC FINALIZE$(m^*, \sigma^*, pk^*_{\mathcal{F}})$:

if $\exists (f, g, m, pk_{\mathcal{F}}, \cdot) \in \mathcal{Q}, s.t.$
$pk_{\mathcal{F}} \in \mathcal{K}_{\mathcal{A}} \wedge m^* \in \mathcal{F}^*_{\mathcal{G}}(\lambda, f, g, m)$
  output 0
else
  if $(\cdot, \cdot, m^*, \cdot, \cdot) \in \mathcal{Q}$
    output 0
  else
    retrieve (pp, pk_{sig})
    parse $pp = (CRS, pp_{\mathcal{S}}, pp_{\mathcal{E}}, \widetilde{ek})$
    parse $pk_{sig} = (vk_S, \widetilde{ek})$
    parse $pk^*_{\mathcal{F}} = (vk_{\mathcal{F}}, ek)$
    parse $\sigma^* = (S, d, \Pi)$
    $x := (pp_{\mathcal{S}}, pp_{\mathcal{E}}, pk_{sig}, pk^*_{\mathcal{F}}, S, d, m^*, CRS)$
    $b \leftarrow \mathsf{Vf}_{\mathsf{NIZK}}(CRS, x, \Pi)$
    output b

Figure 7: A simulated version of Finalize for our construction DFSS

$\mathsf{KGen}_S(pp, msk)$:
$(sk_{sig}, pk_{sig}) \leftarrow \mathsf{KGen}_{sig}(pp, msk)$
$(sk_{\mathcal{F}}, pk_{\mathcal{F}}) \leftarrow \mathsf{KGen}_{\mathcal{F}}(pp, msk)$
set $sk := (sk_{sig}, pk_{\mathcal{F}})$
set $pk := (pk_{sig}, pk_{\mathcal{F}})$
output (sk, pk)

$\mathsf{Sig}_S(pp, sk, m)$:
parse $sk = (sk_{sig}, pk_{\mathcal{F}})$
$\sigma \leftarrow \mathsf{Sig}(pp, sk_{sig}, pk_{\mathcal{F}}, 0^\lambda, m)$
output σ

$\mathsf{Vf}_S(pp, pk, m, \sigma)$:
parse $pk = (pk_{sig}, pk_{\mathcal{F}})$
$b \leftarrow \mathsf{Vf}(pp, pk_{sig}, pk_{\mathcal{F}}, m, \sigma)$
output b

Figure 8: Construction of a signature scheme from an FSS.

$\underline{\mathsf{KGen}_R(pp, msk):}$
$(sk_{sig}, pk_{sig}) \leftarrow \mathsf{KGen}_{sig}(pp, msk)$
$(sk_{\mathcal{F}}, pk_{\mathcal{F}}) \leftarrow \mathsf{KGen}_{\mathcal{F}}(pp, msk)$
set $sk := (sk_{sig}, pk_{\mathcal{F}})$
set $pk := (sk_{\mathcal{F}}, pk_{sig}, pk_{\mathcal{F}})$
output (sk, pk)

$\underline{\mathsf{Sig}_R(pp, sk, m):}$
parse $sk = (sk_{sig}, pk_{\mathcal{F}})$
$\sigma \leftarrow \mathsf{Sig}(pp, sk_{sig}, pk_{\mathcal{F}}, 0^{\lambda}, m)$
output σ

$\underline{\mathsf{Vf}_R(pp, pk, m, \sigma):}$
parse $pk = (pk_{sig}, pk_{\mathcal{F}})$
$b \leftarrow \mathsf{Vf}(pp, pk_{sig}, pk_{\mathcal{F}}, m, \sigma)$
output b

$\underline{\mathsf{Rerandomize}_R(pp, pk, m, \sigma):}$
parse $pk = (sk_{\mathcal{F}}, pk_{sig}, pk_{\mathcal{F}})$
$\sigma' \leftarrow \mathsf{Eval}_{\mathcal{F}}(pp, sk_{\mathcal{F}}, pk_{sig}, m, m, \sigma)$
output σ'

$\underline{\mathsf{Extract}_{\mathsf{IBS}}(pp, msk, \mathsf{ID}):}$
parse $pp = (pp_{\mathcal{F}}, pk_{sig}, pk_{\mathcal{F}})$
parse $msk = (msk_{\mathcal{F}}, sk_{sig}, sk_{\mathcal{F}})$
$\sigma_{\mathsf{ID}} \leftarrow \mathsf{Sig}(pp_{\mathcal{F}}, sk_{sig}, pk_{\mathcal{F}}, 0^{\ell_p(\lambda)}, \mathsf{ID})$
set $sk := (sk_{\mathcal{F}}, \sigma_{\mathsf{ID}}, \mathsf{ID})$
output sk_{ID}

$\underline{\mathsf{Sig}_{\mathsf{IBS}}(pp, sk, m):}$
parse $pp = (pp_{\mathcal{F}}, pk_{sig}, pk_{\mathcal{F}})$
parse $sk = (sk_{\mathcal{F}}, \sigma_{\mathsf{ID}}, \mathsf{ID})$
$\sigma \leftarrow \mathsf{Eval}_{\mathcal{F}}(pp_{\mathcal{F}}, sk_{\mathcal{F}}, pk_{sig}, m, \mathsf{ID}, \sigma_{\mathsf{ID}})$
output σ

$\underline{\mathsf{Vf}_{\mathsf{IBS}}(pp, \mathsf{ID}, m, \sigma):}$
parse $pp = (pp_{\mathcal{F}}, pk_{sig}, pk_{\mathcal{F}})$
$b \leftarrow \mathsf{Vf}(pp_{\mathcal{F}}, pk_{sig}, pk_{\mathcal{F}}, (\mathsf{ID}, m), \sigma)$
output b

$\underline{\mathsf{Setup}_{\mathsf{IBS}}(\lambda):}$
$(pp_{\mathcal{F}}, msk_{\mathcal{F}}) \leftarrow \mathsf{Setup}$
$(sk_{sig}, pk_{sig}) \leftarrow \mathsf{KGen}_{sig}(pp_{\mathcal{F}}, msk_{\mathcal{F}})$
$(sk_{\mathcal{F}}, pk_{\mathcal{F}}) \leftarrow \mathsf{KGen}_{\mathcal{F}}(pp_{\mathcal{F}}, msk_{\mathcal{F}})$
set $pp := (pp_{\mathcal{F}}, pk_{sig}, pk_{\mathcal{F}})$
set $msk := (msk_{\mathcal{F}}, sk_{sig}, sk_{\mathcal{F}})$
output (msk, pp)

Figure 9: Construction of an IBS out of an FSS

$\underline{\mathsf{KGen}_{\mathsf{RSS}}(pp, msk):}$
$(sk_{sig}, pk_{sig}) \leftarrow \mathsf{KGen}_{sig}(pp, msk)$
$(pk_{\mathcal{F}}, sk_{\mathcal{F}}) \leftarrow \mathsf{KGen}_{\mathcal{F}}(pp, msk)$
set $pk := (sk_{\mathcal{F}}, pk_{sig}, pk_{\mathcal{F}})$
set $sk := (sk_{sig}, pk_{\mathcal{F}})$
output (pk, sk)

$\underline{\mathsf{Sig}_{\mathsf{RSS}}(pp, sk, m):}$
parse $sk = (sk_{sig}, pk_{\mathcal{F}})$
$\sigma \leftarrow \mathsf{Sig}(pp, sk_{sig}, pk_{\mathcal{F}}, m)$
output σ

$\underline{\mathsf{Vf}_{\mathsf{RSS}}(pp, pk, \sigma, m):}$
parse $pk = (sk_{\mathcal{F}}, pk_{sig}, pk_{\mathcal{F}})$
$b \leftarrow \mathsf{Vf}(pp, pk_{sig}, pk_{\mathcal{F}}, m)$
output b

$\underline{\mathsf{Redact}_{\mathsf{RSS}}(pp, pk, m, \sigma, m'):}$
parse $pk = (sk_{\mathcal{F}}, pk_{sig}, pk_{\mathcal{F}})$
$\sigma' \leftarrow \mathsf{Eval}_{\mathcal{F}}(pp, sk_{\mathcal{F}}, pk_{sig}, m', m, \sigma)$
output σ'

$\underline{\mathsf{Initialize}(\lambda):}$
$(pp, msk) \leftarrow \mathsf{Setup}(\lambda)$
$(sk_{sig}, pk_{sig}) \leftarrow \mathsf{KGen}_{sig}\mathsf{SanS}(pp, msk)$
$(sk_{\mathcal{F}}, pk_{\mathcal{F}}) \leftarrow \mathsf{KGen}_{\mathcal{F}}\mathsf{SanS}(pp, msk)$
store $(pp, sk_{sig}, sk_{\mathcal{F}}, pk_{sig}, pk_{\mathcal{F}})$
set $\mathcal{M} := \emptyset$
output (pp, pk)

$\underline{\mathsf{Query[Sign]}(pk_i, ?, m_i):}$
retrieve $(pp, sk_{sig}, sk_{\mathcal{F}}, pk_{sig}, pk_{\mathcal{F}})$
$\sigma \leftarrow \mathsf{Sig}_{\mathsf{SanS}}(pp, sk, m)$
set $\mathcal{M} := \mathcal{M} \cup \{m\}$
output σ

$\underline{\mathsf{Finalize}(m^*, \sigma^*):}$
if $m^* \in \mathcal{M}$ then
  output 0
else
  retrieve (pp, sk, pk)
  $b \leftarrow \mathsf{Vf}_{\mathsf{SanS}}(pp, pk, m^*, \sigma^*)$
  output b

Figure 10: Unforgeability for sanitizable signature schemes

$\underline{\mathsf{Initialize}(\lambda):}$
$(pp, msk) \leftarrow \mathsf{Setup}(\lambda)$
store (pp, msk)
set $\mathcal{M} := \emptyset, \mathcal{U} := \emptyset, \mathcal{H} := \emptyset$
output (pp)

$\underline{\mathsf{Query[Init]}(\mathsf{ID}):}$
retrieve (pp, msk)
  $sk_{\mathsf{ID}} \leftarrow \mathsf{Extract}(pp, msk, \mathsf{ID})$
set $\mathcal{H} := \mathcal{H} \cup \{(\mathsf{ID}, sk_{\mathsf{ID}})\}$
output 1

$\underline{\mathsf{Query[Corrupt]}(\mathsf{ID}):}$
if $(\mathsf{ID}, sk_{\mathsf{ID}}) \in \mathcal{H}$
  set $\mathcal{U} := \mathcal{U} \cup \{\mathsf{ID}\}$
  output sk_{ID}
else
  output $\perp$

$\underline{\mathsf{Query[Sign]}(\mathsf{ID}, m):}$
retrieve (pp, msk)
if $(\mathsf{ID}, sk_{\mathsf{ID}}) \in \mathcal{H}$
  $\sigma \leftarrow \mathsf{Sig}(pp, sk_{\mathsf{ID}}, m)$
  set $\mathcal{M} := \mathcal{M} \cup \{(\mathsf{ID}, m)\}$
  output σ
else
  output $\perp$

$\underline{\mathsf{Finalize}(\mathsf{ID}^*, m^*, \sigma^*):}$
if $(\mathsf{ID}^*, m^*) \in \mathcal{M}$
or $\mathsf{ID}^* \in \mathcal{U}$
or $(\mathsf{ID}^*, \cdot) \notin \mathcal{H}$ then
  output 0
else
  retrieve (pp, msk)
  $b \leftarrow \mathsf{Vf}_{\mathsf{IBS}}(pp, \mathsf{ID}^*, m^*, \sigma^*)$
  output b

Figure 11: Unforgeability for identity based signature schemes

ORACLE $O_{pp=(CRS,pp_{\mathcal{S}},pp_{\mathcal{E}},\widetilde{ek}),msk,ssk_{\mathcal{S}}}$

Query[Sign]$(pk^*_{\mathcal{F}}, f, g, k, m)$:
$h_k := (f, g, m, pk^*_{\mathcal{F}}, k)$
$\sigma_k \leftarrow \mathsf{Sig}_{\mathcal{S}}(pp_{\mathcal{S}}, ssk_{\mathcal{S}}, h_k; r_{\mathcal{S}})$
$s_k \leftarrow \mathsf{Enc}(pp_{\mathcal{E}}, \widetilde{ek}, (\sigma_k, h_k); r_{s_k})$
For $i \in \{0, \ldots, n\} \setminus \{k\}$
  $\sigma_i := 0^{|\sigma_k|}$
  $h_i := (0^{\ell_p(\lambda)}, 0^{\ell_p(\lambda)}, 0^{\ell_m(\lambda)}, 0^{|pk^*_{\mathcal{F}}|}, 0)$
  $s_i \leftarrow \mathsf{Enc}(pp_{\mathcal{E}}, \widetilde{ek}, (\sigma_i, h_i); r_{s_i})$
$d \leftarrow \mathsf{Enc}(pp_{\mathcal{E}}, ek, (f, g, k, \sigma_k; r_d))$
$S := (s_0, \ldots, s_n)$
$x := (pp, pk_{sig}, pk^*_{\mathcal{F}}, S, d, m)$
$\omega := (f, g, n, r_d)$
with $\omega_k := (\sigma_k, r_{\mathcal{S}}, k)$
$\Pi \leftarrow \mathcal{P}(CRS, x, \omega)$
$\sigma := (S, d, \Pi)$
output σ

Query[Trans]$(pk^*_{\mathcal{F}}, \alpha, \beta, m, pk_{\mathcal{F}}{}', \sigma)$:
find $sk_{\mathcal{F}} = (ssk_{\mathcal{F}}, dk)$ with $(pk^*_{\mathcal{F}}, sk_{\mathcal{F}}) \in \mathcal{K}_{\mathcal{C}}$
find $pk_{\mathcal{F}}{}' = (vk_{\mathcal{F}}, ek')$ with $pk_{\mathcal{F}}{}' \in \mathcal{K}_{\mathcal{C}}$
abort otherwise
parse $\sigma = (S, d, \Pi)$
$(f, g, i, \sigma_i) \leftarrow \mathsf{Dec}(pp_{\mathcal{E}}, dk, d)$
$x = (pp, pk_{sig}, pk_{\mathcal{F}}{}^*, S, d, m)$
abort if $\mathsf{Vf}_{\mathsf{NIZK}}(CRS, x, \Pi) \neq 1$
$\hat{m} := \mathcal{F}(\lambda, f, \alpha, m)$
$\hat{f} := g(f, \beta)$
$h_{i-1} := (\hat{f}, g, \hat{m}, pk_{\mathcal{F}}{}', i-1)$
$\hat{\sigma}_{i-1} \leftarrow \mathsf{Sig}_{\mathcal{S}}(pp_{\mathcal{S}}, ssk_{\mathcal{F}}, \hat{h}_{i-1}; \hat{r}_{\mathcal{S}})$
$\hat{s}_{i-1} \leftarrow \mathsf{Enc}(pp_{\mathcal{E}}, \widetilde{ek}, (\hat{\sigma}_{i-1}, h_{i-1}); \hat{r}_{s_{i-1}})$
$x = (pp, pk_{sig}, pk_{\mathcal{F}}{}', S, d, m)$
$\omega = (f, g, i-1, r_d)$
with $\omega_{i-1} = (\hat{\sigma}_{i-1}, r_{\hat{s}_{i-1}}, \Pi, vk_{\mathcal{F}}, pk_{\mathcal{F}}{}', m, f, \alpha, \beta)$
$\hat{\Pi} \leftarrow \mathcal{P}(CRS, x, \omega)$
$\hat{\sigma} := (S, d, \hat{\Pi})$
output $\hat{\sigma}$

Query[KGenP]() :
$(sk_{\mathcal{F}}, pk_{\mathcal{F}}) \leftarrow \mathsf{KGen}_{\mathcal{F}}(pp, msk)$
output $pk_{\mathcal{F}}$

Query[KGenS]() :
$(sk_{\mathcal{F}}, pk_{\mathcal{F}}) \leftarrow \mathsf{KGen}_{\mathcal{F}}(pp, msk)$
output $(sk_{\mathcal{F}}, pk_{\mathcal{F}})$

Query[RegisterKey]$(sk_{\mathcal{F}}{}^*, pk_{\mathcal{F}}{}^*)$:

GAME $\mathcal{G}_0$
001 $CRS \leftarrow \mathsf{KGen}_{\mathsf{NIZK}}(1^\lambda)$
002 $(msk_{\mathcal{S}}, pp_{\mathcal{S}}) \leftarrow \mathsf{Setup}_{\mathcal{S}}(1^\lambda)$
003 $(msk_{\mathcal{E}}, pp_{\mathcal{E}}) \leftarrow \mathsf{Setup}_{\mathcal{E}}(1^\lambda)$
004 $(\widetilde{dk}, \widetilde{ek}) \leftarrow \mathsf{KGen}_{\mathcal{E}}(pp_{\mathcal{E}}, msk_{\mathcal{E}})$
005 $pp := (CRS, pp_{\mathcal{S}}, pp_{\mathcal{E}}, \widetilde{ek})$
006 $msk := (msk_{\mathcal{S}}, msk_{\mathcal{E}})$
007 $(ssk_{\mathcal{S}}, vk_{\mathcal{S}}) \leftarrow \mathsf{KGen}_{\mathcal{S}}(pp_{\mathcal{S}}, msk_{\mathcal{S}})$
008 $pk_{sig} := vk_{\mathcal{S}}$
009 $sk_{sig} := (ssk_{\mathcal{S}}, pk_{sig})$
010 $c \leftarrow \mathcal{A}_1^{O_{pp,msk,ssk_{\mathcal{S}}}}(pp, pk_{sig}, pk_{\mathcal{F}})$
011 parse $c = (f_0, g, k, [pk_{\mathcal{F}}, \alpha, \beta]_0^t, t, m_0)$
012 if $pk_{\mathcal{F}}[t] \in \mathcal{K}_{\mathcal{A}} \vee k < t$
013 $out := \bot$
014 if $(\cdot, pk_{\mathcal{F}}[0]) \notin \mathcal{K}_{\mathcal{F}} \quad out := \bot$
015 $\sigma_0 \leftarrow \mathsf{Sig}(pp, sk_{sig}, pk_{\mathcal{F}}[0], f_0, g, k, m_0)$
016 for $i \in \{1, \ldots, t\}$
017 if $\neg\exists sk_{\mathcal{F}}{}^*.\ (sk_{\mathcal{F}}{}^*, pk_{\mathcal{F}}[i]) \in \mathcal{K}_{\mathcal{F}}$
018 $out := \bot$
019 $f_i := g(f_{i-1}, \beta[i])$
020 $m_i := \mathcal{F}(\lambda, f_{i-1}, \alpha[i], m_{i-1})$
021 $q_i := (pp, sk_{\mathcal{F}}{}^*, pk_{sig}, \alpha[i], \beta[i], m_{i-1}, pk_{\mathcal{F}}[i], \sigma_{i-1})$
022 $\sigma_i \leftarrow \mathsf{Trans}_{\mathcal{FG}}(q_i)$
023 if $\sigma_t \neq \bot$
024 $h_{k-t} := (f, g, m_t, pk_{\mathcal{F}}[t], k-t)$
025 $\sigma_{k-t} \leftarrow \mathsf{Sig}_{\mathcal{S}}(pp_{\mathcal{S}}, ssk_{\mathcal{S}}, h_{k-t}; r_{\mathcal{S}})$
026 $s_{k-t} \leftarrow \mathsf{Enc}(pp_{\mathcal{E}}, \widetilde{ek}, (\sigma_{k-t}, h_{k-t}); r_{s_{k-t}})$
027 For $j \in \{0, \ldots, n\} \setminus \{k\}$
028 $\sigma_j := 0^{|\sigma_{k-t}|}$
029 $h_j := (0^{\ell_p(\lambda)}, 0^{\ell_p(\lambda)}, 0^{\ell_m(\lambda)}, 0^{|pk_{\mathcal{F}}[t]|}, 0)$
030 $s_j \leftarrow \mathsf{Enc}(pp_{\mathcal{E}}, \widetilde{ek}, (\sigma_j, h_j); r_{s_i})$
031 $d \leftarrow \mathsf{Enc}(pp_{\mathcal{E}}, ek, (f, g, k, \sigma_{k-t}; r_d))$
032 $S := (s_0, \ldots, s_n)$
033 $x := (pp, pk_{sig}, pk_{\mathcal{F}}[t], S, d, m_t)$
034 $\omega := (f, g, n, r_d)$
035 with $\omega_{k-t} := (\sigma_{k-t}, r_{\mathcal{S}}, k)$
036 $\Pi \leftarrow \mathcal{P}(CRS, x, \omega)$
037 $\sigma := (S, d, \Pi)$
038 else
039 $\sigma := \sigma_t$
040 if $out \neq \bot$ then $out := \sigma$
041 $b^* \leftarrow \mathcal{A}_2(out)$

GAME $\mathcal{G}_1$
101 $(CRS, \xi) \leftarrow \mathsf{Sim}_{\mathsf{NIZK}}(1^\lambda)$
136 $\Pi \leftarrow \mathsf{Sim}_{\mathsf{NIZK}}(CRS, \xi, x)$

GAME $\mathcal{G}_2$
227 for $j \in \{0, \ldots, n\}$
231 $\delta := (0^{\ell_p(\lambda)}, 0^{\ell_p(\lambda)}, 0, 0^{|\sigma_{k-t}|})$
  $d \leftarrow \mathsf{Enc}(pp_{\mathcal{E}}, ek, \delta; r_d))$

GAME $\mathcal{G}_3$
316 for $j \in \{1, \ldots, t-1\}$
324 if $\neg\exists sk_{\mathcal{F}}{}^*.\ (sk_{\mathcal{F}}{}^*, pk_{\mathcal{F}}[t]) \in \mathcal{K}_{\mathcal{F}}$
  $out := \bot$
325 $f_t := g(f_{t-1}, \beta[t])$
$m_t := \mathcal{F}(\lambda, f_{t-1}, \alpha[t], m_{t-1})$
$q_t := (pp, sk_{\mathcal{F}}{}^*, pk_{sig}, \alpha[t], \beta[t], m_{t-1}, pk_{\mathcal{F}}[t], \sigma_{t-1})$
parse $sk_{\mathcal{F}}{}^* = (ssk_{\mathcal{F}}, dk)$
parse $pk_{\mathcal{F}}[t] = (vk'_{\mathcal{F}}, ek')$
parse $\sigma_{t-1} = (S', d', \Pi')$
$(f, g, i, \varsigma_i) \leftarrow \mathsf{Dec}(pp_{\mathcal{E}}, dk, d')$
$x' = (pp, pk_{sig}, pk_{\mathcal{F}}, S', d', m_{t-1})$
if $\mathsf{Vf}_{\mathsf{NIZK};Z_i}(CRS, x', \Pi) \neq 1$ then $out := \bot$
$h_{i-1} := (f_t, g, m_t, pk_{\mathcal{F}}[t] i-1)$
$\hat{\varsigma}_{i-1} \leftarrow \mathsf{Sig}_{\mathcal{S}}(pp_{\mathcal{S}}, ssk_{\mathcal{F}}, h_{i-1}; r_{\mathcal{S}})$
326 $s_{i-1} \leftarrow \mathsf{Enc}(pp_{\mathcal{E}}, \widetilde{ek}, (\hat{\varsigma}_{i-1}, h_{i-1}); r_s)$
334 $\omega = (f, g, i-1, r_d)$
335 with $\omega_{i-1} = (\varsigma_{i-1}, r_{\mathcal{S}}, \Pi', pk_{\mathcal{F}}, m_{t-1}, f, \alpha, \beta)$
336 $\Pi \leftarrow P_{\mathsf{NIZK}}(CRS, \xi, x)$
337 $\sigma := (S, d, \Pi)$

GAME $\mathcal{G}_4$
227 for $j \in \emptyset$
431 $d \leftarrow \mathsf{Enc}(pp_{\mathcal{E}}, ek', (f, g, i-1, \varsigma_{i-1}); r_d)$

GAME $\mathcal{G}_5$
501 $CRS \leftarrow \mathsf{KGen}_{\mathsf{NIZK}}(1^\lambda)$
536 $\Pi \leftarrow \mathcal{P}(CRS, x, \omega)$

Figure 12: Games for CFA proof of DFSS (Theorem 2)