Delegatable Functional Signatures

Michael Backes
MPI-SWS
Saarland University
Germany

Sebastian Meiser
Saarland University
Germany

Dominique Schröder
Saarland University
Germany

October 10, 2013

Abstract

We introduce *delegatable functional signatures* (DFS) which support the delegation of signing capabilities to another party, called the *evaluator*, with respect to a functionality $\mathcal{F}$. In a DFS, the signer of a message can choose an evaluator, specify how the evaluator can modify the signature without voiding its validity, allow additional input and decide how the evaluator can further delegate its capabilities.

The main contribution of this paper is twofold. First, we propose DFS, a novel cryptographic primitive that unifies several seemingly different signature primitives, including functional signatures as defined by Boyle, Goldwasser, and Ivan (eprint 2013/401), sanitizable signatures, identity based signatures, and blind signatures. To achieve this unification, we present several definitions of unforgeability and privacy. Finding appropriate and meaningful definitions in this context is challenging due to the natural mealleability of DFS and due to the multi-party setting that may involve malicious keys.

Second, we present a complete characterization of the instantiability of DFS under common assumptions, like the existence of one-way functions. Here, we present both positive and negative results. On the positive side we show that DFS not achieving our notion of privacy can be constructed from one-way functions. Furthermore, we show that unforgerable and private DFS can be constructed from doubly enhanced trapdoor permutations. On the negative side we show that the previous result is optimal regarding its underlying assumptions presenting an impossibility result for unforgeable private DFS from one-way permutations.

Contents

1 Introduction **1**
- 1.1 Our Contribution . . . 1
- 1.2 Related Work . . . 3
- 1.3 Independent Concurrent Work . . . 4

2 Delegatable Functional Signatures **5**
- 2.1 Formal Description of a DFS scheme 5

3 Security Notions for DFS **6**
- 3.1 Unforgeability . . . 7
- 3.2 Relations between the unforgeability notions . . . 9
- 3.3 Privacy . . . 10
- 3.4 Relations between the privacy notions . . . 11
- 3.5 Implications . . . 12

4 Possibility and Impossibility of DFS From OWFs **14**
- 4.1 Impossibility of DFS from OWPs . 14

5 Constructing DFS From Trapdoor Permutations **18**
- 5.1 Cryptographic Primitives . . . 18
- 5.2 Our scheme . . . 22
- 5.3 Security . . . 23

A Relation between DFS and FS **35**
- A.1 Constructing functional signatures (from DFSS) . . . 35
- A.2 Constructing delegatable functional signatures (from FS) . . . 36

B Blind signatures **38**
- B.1 Basic security notions for blind signatures . . . 39

1 Introduction

Digital signature schemes resemble the idea of a hand written signature in the sense that a signer signs messages with his private key sk_{sig} and anybody can check the validity of the signature using the corresponding public key pk_{sig}. The elementary security property is unforgeability under chosen message attacks which says that an adversary cannot compute a signature on a fresh message, even if he has observed q signatures on q messages of his choice [29]. This security definition models the idea of *non*-malleability for digital signatures: The adversary should not be able to modify any signature such that it verifies for a different message.

For many emerging applications, such as the delegation of computation on authenticated data, the basic notion is insufficient. Consider as an example the scenario where a user wants to outsource computation on authenticated data to untrusted parties, called the evaluators, without handing out her secret key. The evaluators are organized in a chain, where each evaluator receives the intermediate result and computes a specific function f chosen by the user on the intermediate result and its own input. The chain computation should be publicly verifiable which means that everybody can verify that:

- The computation was based on the original data of the user.
- Only the functions chosen by the user were applied to the data (in the correct order).
- Any delegation of computation by an evaluator to a third party has been authorized by the user.

For this scenario we consider malicious, non-colluding adversaries and the following security notions: *Unforgeability* says that malicious evaluators can only apply the functions(s) they were allowed to apply. *Privacy* says that given the result of a computation, it is not possible to gain information about the computed functions or their input (or the parties that did the computation). Traditional signature schemes as well as their malleable variants are not suitable in this setting. In this paper we close this gap by introducing the concept of *delegatable functional signatures*.

1.1 Our Contribution

Our main contributions are as follows. First, we introduce *delegatable functional signatures* (DFS). This primitive supports highly controlled, fine-grained delegation of signing capabilities to designated third parties and is general enough to cover several malleable signature schemes. Second, we present strong security notions for unforgeability and privacy that also take into account insider adversaries. Third, we provide a complete charaterization regarding the achievability of our security notions based on general complexity assumptions. In the following we discuss each contribution comprehensively.

Delegatable Functional Signatures. Delegatable functional signatures support the delegation of signing capabilities to another party, called the *evaluator*, with respect to a functionality $\mathcal{F}$. The evaluator may compute valid signatures on messages m' and delegate capabilities f' to another evaluator with key k whenever $(f', m') \leftarrow \mathcal{F}(f, \alpha, k, m)$ for a value α of the evaluators choice. Thus, the functionality describes how an evaluator can perform the following two tasks.

Malleability. The designated evaluator can derive a signature on m' from a signature on m, if $(f', m') \leftarrow \mathcal{F}(f, \alpha, k, m)$, where the evaluator picks α and k himself.

Example 1. *Suppose that Alice wants to allow Bob to fill in information in a few fields of a document that she signs. Her choice of f describes the places where information can be added, as well as which information can be added by Bob (e.g., 16 characters) without harming the validity of the signature. Bob chooses the fields and the information he wishes to fill in by choosing the corresponding value for α, and he derives a signature on m', where $(\cdot, m') \leftarrow \mathcal{F}(f, \alpha, k, m)$.*

Delegatability. The designated evaluator can delegate signing capabilities f' on his signature on m', to other parties, if $(f', m') \leftarrow \mathcal{F}(f, \alpha, k, m)$, where k is the key of another *evaluator* (or his own key, if he wants to apply several functions successively) and where the evaluator picks α himself.

Example 2. *Suppose that Alice wants to restrict how Bob can delegate further capabilities. Her choice of f additionally describes that after filling in information, certain parts of the document can be censored without harming the validity of the signature and she adds a description of the evaluators that are allowed to do this second round of processing. Bob chooses a key of the next evaluator in the chain as well as restrictions on which part may be censored by choosing the corresponding value for α and delegates the capabilities f', where $(f', \cdot) \leftarrow \mathcal{F}(f, \alpha, k, m)$.*

Our definition also covers signing capabilities for fresh messages. If Alice wants to give Bob the capability to sign certain messages in her name, she can simply generate a signature σ_{fresh} for a new (empty) message and use f to specify which capabilities Bob has, i.e., which signatures he can derive from σ_{fresh}.

Security Model for DFS. A central contribution of this paper are the formal definitions of unforgeability and privacy. On an abstract level, these notions resemble the well known intuition: Unforgeability means that no signatures can be forged, except on messages within a certain class. Privacy means that derived signatures are indistinguishable from fresh signatures. However, finding meaningful and achievable definitions for DFS is rather challenging, because the signatures are malleable by nature and we are also considering the multi-party setting:

Unforgeability In a DFS scheme the signer specifies for every signature the degree of malleability and how this malleability can be delegated. Unforgeability is then captured by a transitive closure that contains all messages that can trivially be derived.

Privacy Our notion of privacy follows the idea that all information about signatures should be hidden (except for the message). This is captured in an indistinguishability game where the adversary can hand in a signature of his own. Either this signature is treated exactly as the adversary specifies it (modified by evaluators of his choice, possibly under keys of the adversary possesses) or a new signature for the same (resulting) message is created.

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

Unifying signature primitives. Delegatable functional signatures are very versatile and imply several seemingly different signature primitives. These include functional signatures, which were recently introduced by Boyle, Goldwasser, and Ivan [14], blind signatures, identity based signatures, sanitizable signatures and redactable signatures.

Instantiability of DFS. We give a complete characterization of the instantiability of DFS from general complexity based assumptions presenting both positive and negative results.

Possibility of DFS. On the positive side we show that DFS can be constructed from one-way functions in a black-box way if one gives up privacy.

Theorem 1 (Possibility, informal)**.** *Unforgeable delegatable functional signatures exist if one-way functions exist.*

Furthermore, we show that unforgeable and private DFS schemes can be constructed from (doubly enhanced) trapdoor permutations in a black-box way. Our scheme shows that our strong definitions for unforgeability and privacy are achievable for arbitrary, efficiently computable, choices of $\mathcal{F}$.

Theorem 3 (Possibility, informal)**.** *Private unforgeable delegatable functional signatures exist if doubly enhanced trapdoor permutations exist.*

Impossibility of DFS. We show that the previous result is optimal w.r.t. the underlying assumptions. We show that unforgeable and private delegatable functional signatures cannot be constructed from one-way functions. The basic idea is to construct a blind signature scheme out of any functional signature scheme in a black-box way. Recently, Katz, Schröder, and Yerukhimovich have shown that blind signature schemes cannot be build from one-way permutations using black-box techniques only [34]. A construction of DFS based on OWFs would yield a black-box construction of blind signature schemes based on OWFs. However, this would directly contradict the result of [34].

Theorem 2 (Impossibility, informal)**.** *Private unforgeable delegatable functional signatures secure against insider adversaries cannot be constructed from one-way functions in a black-box way.*

1.2 Related Work

(Delegatable) Anonymous Credentials. In anonymous credential systems users can prove the possession of a credential 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., [15, 16, 20, 21, 8, 22, 39, 19, 24]. The main difference between delegatable anonymous credential schemes, such as [7, 2], 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 [6]. 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. A limited 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 [38, 32, 36, 17] 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 [32], linearly homomorphic signature schemes [27, 13] and schemes that allow for evaluating polynomial functions [12].

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 [4, 18] 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 [28] 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 to 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. [23]. 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.

The general framework by Ahn et al. [3] is versatile and, like delegatable functional signatures, unifies a variety of signature notions. A variety of instantiations can be captured in their framework using their predicate P to describe a complex functionality for deriving signatures. In fact, it seems possible to describe delegatable functional signatures in their framework by encoding the functionality in a complex predicate and by encoding the keys of the evaluators as specifically structured signatures. However, so far there exist no construction for their framework that is capable of dealing with such predicates (their constructions support single element sets $\mathcal{M}$, but to encode our scheme, at least sets of size two are required). Moreover, they do not explore the minimal computational assumptions.

1.3 Independent Concurrent Work

In a concurrent and independent work, Boyle, Goldwasser, and Ivan [14] introduced *functional digital signatures.* In their formulation the signer hands out keys sk_f for functions f to allow the recipient to sign all messages in the range of f. Similar to our contributions, they define notions of unforgeability and privacy (called function privacy) and present several constructions for functional digital signatures. One of their constructions also shows that functional signatures can be build from one-way functions provided that one is willing to give up privacy. They furthermore show how to construct one-round delegation schemes out of a functional digital signature scheme.

While our work is closely related, it differs in several aspects: First, we not only consider the controlled malleability of the signature, but also support the delegation of signing capabilities. Second, while we also show for our notions that unforgeable-only DFS schemes can be build from one-way functions, we additionally show that private DFS schemes can *not* be constructed from from one-way permutations (see Section 4). We believe that our impossibility result should also hold for their definition of functional signatures [14], because our impossibility result does not rely on the delegation property of our scheme. Furthermore, DFS signatures allow for authenticated chain computations. In Appendix A we compare delegatable functional signature schemes to functional digital signature schemes and show how to construct a functional digital signature scheme out of a delegatable functional signature scheme. Whether the converse holds is unknown (see Appendix A.2 for a discussion).

2 Delegatable Functional Signatures

Delegatable functional signatures support the delegation of signing capabilities to another party, called the *evaluator*, with respect to a functionality $\mathcal{F}$. The evaluator may compute valid signatures on messages m' and delegate capabilities f' to another evaluator with key k whenever $(f', m') \leftarrow \mathcal{F}(f, \alpha, k, m)$ for a value α of the evaluators choice.

Our definition of DFS limits the delegation capabilities of the evaluator. In particular, the signer specifies how an evaluator may delegate his signing rights.

2.1 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 $\mathcal{P}_f$ and $\mathcal{P}_\alpha$ is a signature scheme that additionally supports a controlled form of malleability and delegation. A DFS is described by a functionality $\mathcal{F} : \mathbb{N} \times \mathcal{P}_f \times \mathcal{P}_\alpha \times \mathcal{K} \times \mathcal{M} \to (\mathcal{P}_f \times \mathcal{M}) \cup \{\perp\}$ that specifies how messages can be changed and how capabilities can be delegated. Once the signer received a message-signature pair, it can compute signatures on messages of its choice (that are legitimate w.r.t. $\mathcal{F}$) and can partially delegate his signing capabilities to another evaluator. We model this property by introducing an algorithm $\mathsf{Eval}_\mathcal{F}$ for evaluating functions on signatures. This algorithm takes as input the parameter α that defines the evaluator's own input to the function f, the message m, and a key pk'_{ev}. The algorithm $\mathsf{Eval}_\mathcal{F}$ outputs a signature σ' on m', where $(f', m') \leftarrow \mathcal{F}(\lambda, f, \alpha, pk'_{ev}, m)$. This new signature σ' can be changed by an evaluator that owns a (possibly different) key sk'_{ev} and this evaluator can transform it further with the new capability f'.

Definition 1. *(Delegatable functional signatures). A* delegatable functional signature scheme DFSS *is a tuple of efficient algorithms* $\mathsf{DFSS} = (\mathsf{Setup}, \mathsf{KGen}_{sig}, \mathsf{KGen}_{ev}, \mathsf{Sig}, \mathsf{Eval}_\mathcal{F}, \mathsf{Vf})$ *defined as follows:*

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

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

$(\boldsymbol{sk}_{ev}, \boldsymbol{pk}_{ev}) \leftarrow \mathsf{KGen}_{ev}(\boldsymbol{pp}, \boldsymbol{msk})$**:** *The evaluation key generation algorithm KGen_{ev} outputs a secret evaluator key sk_{ev} and a public evaluator key pk_{ev}.*

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

$\hat{\sigma} \leftarrow \mathsf{Eval}_{\mathcal{F}}(\boldsymbol{pp}, \boldsymbol{sk}_{ev}, \boldsymbol{pk}_{sig}, \alpha, m, \boldsymbol{pk}'_{ev}, \sigma)$**:** *The evaluation algorithm outputs a derived signature $\hat{\sigma}$ for m' on the capability f', that can be modified using the evaluator key sk'_{ev} associated with pk'_{ev}, where $(f', m') \leftarrow \mathcal{F}(\lambda, f, \alpha, pk'_{ev}, m)$ (or an error symbol $\perp$).*

$b \leftarrow \mathsf{Vf}(\boldsymbol{pp}, \boldsymbol{pk}_{sig}, \boldsymbol{pk}_{ev}, m, \sigma)$**:** *The verification algorithm Vf outputs a bit $b \in \{0, 1\}$.*

A DFS is *correct* if the verification algorithm outputs 1 for all honestly generated signatures and for all valid transformations of honestly generated signatures.

Formally, we define a *correctness set S* of all message/signature pairs for which Vf should output 1 as follows:

Definition 2. *($\mathcal{F}$-Correctness). Given a delegatable functional signature scheme* $\mathsf{DFSS} = (\mathsf{Setup}, \mathsf{KGen}_{sig}, \mathsf{KGen}_{ev}, \mathsf{Sig}, \mathsf{Eval}_{\mathcal{F}}, \mathsf{Vf})$*, let the correctness set $S_{\mathcal{F}}$ for a functionality $\mathcal{F}$ be a set such that:*

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

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

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

A delegatable functional signature scheme $\mathsf{DFSS} = (\mathsf{Setup}, \mathsf{KGen}_{sig}, \mathsf{KGen}_{ev}, \mathsf{Sig}, \mathsf{Eval}_{\mathcal{F}}, \mathsf{Vf})$ *is* $\mathcal{F}$-correct *for the functionality $\mathcal{F}$ if for all elements* $(pp, m, f, \sigma, pk_{sig}, pk_{ev}) \in S_{\mathcal{F}}$ *it holds that* $\mathsf{Vf}(pp, pk_{sig}, pk_{ev}, m, \sigma) = 1$.

3 Security Notions for DFS

In this section we define unforgeability and privacy for delegatable functional signatures. In both cases 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 delegatable 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 $G(\mathsf{DFSS}, \mathcal{F}, \mathcal{A}, \lambda)$ [11]. 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 $G(\mathsf{DFSS}, \mathcal{F}, \mathcal{A}, \lambda) = 1$.

3.1 Unforgeability

Intuitively, a delegatable 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 its knowledge. 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 and because several parties are involved (and they may even use malicious keys). 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_{ev}) and has oracle access to both the Sig and the $\mathsf{Eval}_{\mathcal{F}}$ algorithm. Our definition of unforgeability against outsider attacks resembles the traditional definition of unforgeability for signature schemes [30], 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 an honestly generated private key sk_{ev}. The second notion allows the adversary to choose its own private key(s) maliciously. Note, that the attacker might choose these keys adaptively. We refer to this notion as unforgeability against *strong* insider attacks.

We model our notions by giving the adversary access to three different KGen oracles. An adversary only using Query[KGenP] to retrieve public keys is considered an *outsider*, an adversary that additionally has access to the oracle Query[KGenS] to retrieve one or more secret evaluator keys is considered an *insider*. An adversary that additionally has access to the oracle Query[RegKey] to (adaptively) 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 set $\mathcal{K}_{\mathcal{C}}$ stores all key pairs, $K_{\mathcal{A}}$ contains all public for which the adversaries knows the private key, and Q stores $\mathcal{A}$'s queries to both Query[Sign] and Query[Eval]. To handle the information that an adversary can trivially deduce from its queries, we define the transitive closure for functionalities.

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 $(\lambda, (f, m))$ recursively as follows:*

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

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

Note that the transitive closure $\mathcal{F}^*$ on $(\lambda, (f, m))$ might not be efficiently computable (and thus a challenger for Unf might not be efficient).

Although it is not necessary to compute the closure explicitly in our case, one could require a DFSS to provide an efficient algorithm Check$-\mathcal{F}$ such that Check$-\mathcal{F}(\lambda, f, m, m^*) = 1$ iff $m \in \mathcal{F}^*(\lambda, (f, m))$.

```
proc Initialize $(\lambda)$:
$(pp, msk) \leftarrow \mathsf{Setup}(\lambda)$
$(sk_{sig}, pk_{sig}) \leftarrow \mathsf{KGen}_{sig}(pp, msk)$
store $(pp, msk, sk_{sig}, pk_{sig})$
set $\mathcal{K_C} := \emptyset, \mathcal{K_A} := \emptyset, \mathcal{Q} := \emptyset$
output $(pp, pk_{sig})$

proc Finalize$(m^*, \sigma^*, pk^*_{ev})$ :
if $\exists (f, m, pk_{ev}, \cdot) \in \mathcal{Q}$, s.t.
$pk_{ev} \in \mathcal{K_A} \wedge m^* \in \mathcal{F}^*(\lambda, (f, m))$
  output 0
else
  if $(\cdot, \cdot, m^*, \cdot, \cdot) \in \mathcal{Q}$
    output 0
  else
    retrieve $(pp, pk_{sig})$
    $b \leftarrow \mathsf{Vf}(pp, pk_{sig}, pk^*_{ev}, m^*, \sigma^*)$
    output $b$

proc Query[KGenP]() :
retrieve $(pp, msk)$
$(sk_{ev}, pk_{ev}) \leftarrow \mathsf{KGen}_{ev}(pp, msk)$
set $\mathcal{K_C} := \mathcal{K_C} \cup (sk_{ev}, pk_{ev})$
output $(pk_{ev})$

proc Query[KGenS]() :
retrieve $(pp, msk)$
$(sk_{ev}, pk_{ev}) \leftarrow \mathsf{KGen}_{ev}(pp, msk)$
set $\mathcal{K_C} := \mathcal{K_C} \cup \{(sk_{ev}, pk_{ev})\}$
set $\mathcal{K_A} := \mathcal{K_A} \cup \{pk_{ev}\}$
output $(sk_{ev}, pk_{ev})$

proc Query[RegKey]$(sk^*_{ev}, pk^*_{ev})$ :
set $\mathcal{K_C} := \mathcal{K_C} \cup \{(sk^*_{ev}, pk^*_{ev})\}$
set $\mathcal{K_A} := \mathcal{K_A} \cup \{pk^*_{ev}\}$

proc Query[Sign]$(pk^*_{ev}, f, m)$ :
retrieve $(pp, sk_{sig})$
if $(\cdot, pk^*_{ev}) \in \mathcal{K_C}$
  $\sigma \leftarrow \mathsf{Sig}(pp, sk_{sig}, pk^*_{ev}, f, m)$
  set $\mathcal{Q} := \mathcal{Q} \cup \{(f, m, pk^*_{ev}, \sigma)\}$
  output $\sigma$
else output$\perp$

proc Query[Eval]$(pk^*_{ev}, \alpha, m, pk'_{ev}, \sigma)$ :
retrieve $(pp, pk_{sig})$
if $(sk^*_{ev}, pk^*_{ev}) \in \mathcal{K_C} \wedge (\cdot, pk'_{ev}) \in \mathcal{K_C}$
  $\sigma' \leftarrow \mathsf{Eval}_{\mathcal{F}}(pp, sk^*_{ev}, pk_{sig}, \alpha, m, pk'_{ev}, \sigma)$
  if $\sigma' \neq \perp$
    extract $f$ from $\sigma$ using $sk^*_{ev}$
    let $(f', m') := \mathcal{F}(\lambda, f, \alpha, pk'_{ev}, m)$
    set $\mathcal{Q} := \mathcal{Q} \cup \{f', m', pk'_{ev}, \sigma'\}$
  output $\sigma'$
else output $\perp$
```

Figure 1: Unforgeability for delegatable functional signature schemes.

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

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

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

Remark: We assume implicitly that f can be extracted from σ using sk_{ev} from any valid query to $\mathsf{Eval}_{\mathcal{F}}$. We believe that this is a reasonable assumption, because the evaluator that transforms a signature should learn the value f, as it describes the capabilities of the evaluator. In fact, our construction (Section 5) satisfies this property.

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[Eval], the challenger only allows to delegate to keys that are "known" to it, which is formalized with the set $\mathcal{K_C}$.

The oracle $\mathsf{Query[Eval]}$ 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 $\mathsf{Query[Sign]}$ or $\mathsf{Query[Eval]}$, 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_{ev} for which the adversary knows the secret key sk_{ev}. Consequently, $\mathcal{K}_\mathcal{A}$ is initially empty and is only extended by $\mathsf{Query[KGenS]}$ and $\mathsf{Query[RegKey]}$. Whenever $\mathcal{A}$ delegates a signature to a key $pk'_{ev} \in \mathcal{K}_\mathcal{A}$, 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 $(\cdot, m^*, \cdot, \cdot) \neq \mathcal{Q}$. Observe that a different public key pk_{ev} 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_{ev} at all.

3.2 Relations between the unforgeability notions

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_{ev} 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_{ev}, 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 $\mathsf{Query[KGenS]}$, 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_{ev}, pk_{ev}) is used that is highly unlikely to be an output of KGen_{ev} (e.g., one of them is 0^λ).

Proposition 1 ($\mathsf{EU\text{-}X\text{-}A}$-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.*

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

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

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

3.3 Privacy

Our privacy notion for DFS says 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 $\mathsf{Eval}_{\mathcal{F}}$.
- a fresh signature on m', where $(\cdot, m') \leftarrow \mathcal{F}(\ldots)$ was computed via one or more applications of $\mathcal{F}$ to m.

This indistinguishability should hold even against an adversary with oracle access to KGen_{ev}, Sig and $\mathsf{Eval}_{\mathcal{F}}$ 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 that are defined analogously to Definition 19 in Section 3.1. In the following definition, the set $\mathcal{K}_{\mathcal{C}}$ stores all key pairs, $K_{\mathcal{A}}$ contains all public keys for which the adversaries knows the private key, and $\mathcal{K}_X$ stores the keys used in the challenge oracle $\mathsf{Query}[\mathsf{Sign}\text{-}\mathcal{F}]$. Note that is necessary to check that the adversary did not learn a private key for a key used by the challenge oracle, or it can simply revoke the privacy afterwards.

```
proc Initialize ($\lambda$):
$b \leftarrow \{0,1\}$
$(pp, msk) \leftarrow \mathsf{Setup}(\lambda)$
$(sk_{sig}, pk_{sig}) \leftarrow \mathsf{KGen}_{sig}(pp, msk)$
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 $(pp, sk_{sig}, pk_{sig})$

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[Eval]($pk^*_{ev}, \alpha, m, pk'_{ev}, \sigma$) :
retrieve $(pp, pk_{sig})$
if $(sk^*_{ev}, pk^*_{ev}) \in \mathcal{K}_{\mathcal{C}} \wedge (pk'_{ev}, \cdot) \in \mathcal{K}_{\mathcal{C}}$
  $\sigma' \leftarrow \mathsf{Eval}_{\mathcal{F}}(pp, sk^*_{ev}, pk_{sig}, \alpha, m, pk'_{ev}, \sigma)$
  output $\sigma'$
```

```
proc Query[KGenP]() :
retrieve $(pp, msk)$
$(sk_{ev}, pk_{ev}) \leftarrow \mathsf{KGen}_{ev}(pp, msk)$
set $\mathcal{K}_{\mathcal{C}} := \mathcal{K}_{\mathcal{C}} \cup (sk_{ev}, pk_{ev})$
output $(pk_{ev})$

proc Query[KGenS]() :
retrieve $(pp, msk, sk_{sig})$
$(sk_{ev}, pk_{ev}) \leftarrow \mathsf{KGen}_{ev}(pp, msk)$
set $\mathcal{K}_{\mathcal{C}} := \mathcal{K}_{\mathcal{C}} \cup \{(sk_{ev}, pk_{ev})\}$
set $\mathcal{K}_{\mathcal{A}} := \mathcal{K}_{\mathcal{A}} \cup \{pk_{ev}\}$
output $(sk_{ev}, pk_{ev})$

proc Query[Sign]($pk^*_{ev}, f, m$) :
retrieve $(pp, sk_{sig})$
if $(\cdot, pk^*_{ev}) \in \mathcal{K}_{\mathcal{C}}$
  $\sigma \leftarrow \mathsf{Sig}(pp, sk_{sig}, pk^*_{ev}, f, m)$
  output $\sigma$
```

```
proc Query[RegKey]($sk^*_{ev}, pk^*_{ev}$) :
set $\mathcal{K}_{\mathcal{C}} := \mathcal{K}_{\mathcal{C}} \cup \{(sk^*_{ev}, pk^*_{ev})\}$
set $\mathcal{K}_{\mathcal{A}} := \mathcal{K}_{\mathcal{A}} \cup \{pk^*_{ev}\}$

proc Query[Sign-$\mathcal{F}$]($[pk_{ev}, \alpha]^t_0, t, m_0, \sigma_0$) :
retrieve $(b, pp, sk_{sig}, pk_{sig})$
if $(\cdot, pk_{ev}[t]) \notin \mathcal{K}_{\mathcal{C}} \vee f \neq f_0$   output $\perp$
if $\mathsf{Vf}(pp, pk_{sig}, pk_{ev}[0], m_0, \sigma_0) \neq 1$ then output $\perp$
if $\neg\exists sk^*_{ev}.\ (sk^*_{ev}, pk_{ev}[0]) \in \mathcal{K}_{\mathcal{C}}$   output $\perp$
extract $f_0$ from $\sigma_0$ using $sk^*_{ev}$
for $i \in \{1, \ldots, t\}$
  if $\neg\exists sk^*_{ev}.\ (sk^*_{ev}, pk_{ev}[i-1]) \in \mathcal{K}_{\mathcal{C}}$
    output $\perp$
  $(f_i, m_i) := \mathcal{F}(\lambda, f_{i-1}, \alpha[i], pk_{ev}[i], m_{i-1})$
  $q_i := (pp, sk^*_{ev}, pk_{sig}, \alpha[i], m_{i-1}, pk_{ev}[i], \sigma_{i-1})$
  $\sigma_i \leftarrow \mathsf{Eval}_{\mathcal{F}}(q_i)$
set $\mathcal{K}_X := \mathcal{K}_X \cup \{pk_{ev}[t]\}$
if $b = 0 \wedge \sigma_t \neq \perp$
  $\sigma \leftarrow \mathsf{Sig}(pp, sk_{sig}, pk_{ev}[t], f_t, m_t)$
else
  $\sigma := \sigma_t$
output $\sigma$
```

Figure 2: Privacy under chosen functionality attacks CFA for delegatable functional signature schemes.

Definition 5. *(Privacy under chosen function attacks (CFA)) against $X \in$ {Outsider, Insider, S-Insider}. Let* $\mathsf{DFSS} = (\mathit{Setup}, \mathit{KGen}_{sig}, \mathit{KGen}_{ev}, \mathit{Sig}, \mathit{Eval}_{\mathcal{F}}, \mathit{Vf})$ *be a delegatable functional signature* *scheme. The definition uses the game* $\mathsf{CFA}(\mathsf{DFSS}, \mathcal{F}, \mathcal{A}, \lambda)$ *defined in Figure 2. We say that* DFSS *is* privacy-preserving under chosen function attacks (X-CFA) *for the functionality* $\mathcal{F}$ *if for all PPT adversaries* $\mathcal{A}_X$

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

is negligible in λ*, where* $\mathcal{A}_{Outsider}$ *can neither invoke the oracles* Query[KGenS] *nor* Query[RegKey]*; the attacker* $\mathcal{A}_{Insider}$ *can 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 adversary may choose an arbitrary challenge message m_0, together with a capability f_0. The challenger constructs a respective signature σ_0. Furthermore the challenger repeatedly applies $\mathsf{Eval}_{\mathcal{F}}$ to σ_0 and allows the adversary to choose the parameters α_i that are input to $\mathsf{Eval}_{\mathcal{F}}$ and the key $pk_{ev}[i]$ 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 $\mathsf{Eval}_{\mathcal{F}}$ within the query Query[Sign-$\mathcal{F}$], 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 $pk_{ev}[t]$ can apply the capability f_t. The challenger adds the key $pk_{ev}[t]$ 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 ($\sigma_t = \perp$), Query[Sign-$\mathcal{F}$] 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 2.

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

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 $pk_{ev}[t]$ to which $\mathcal{A}$ knows the secret key ($pk_{ev}[t] \in \mathcal{K}_{\mathcal{A}} \cap \mathcal{K}_X$), 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 $pk^*_{ev} \in \mathcal{K}_{\mathcal{A}}$.

3.4 Relations between the privacy notions

For privacy, we have the same hierarchy as for unforgeability: A scheme that is secure against outsiders may be insecure against insiders, as the key sk_{ev} 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_{ev}, pk_{ev}) is used that is highly unlikely to be an output of KGen_{ev} (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.*

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

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

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

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.

□

3.5 Implications

In the following section we show that DFS imply several seemingly different signature primitives. Using only black-box access to a given delegatable functional signature scheme $\mathsf{DFSS} = (\mathsf{Setup}, \mathsf{KGen}_{sig}, \mathsf{KGen}_{ev}, \mathsf{Sig}, \mathsf{Eval}_{\mathcal{F}}, \mathsf{Vf})$, we construct (among others) identity based signature schemes, sanitizable signature, and redactable signature schemes.

Signature Schemes. As a warm-up, we show that DFS 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 messages of its choice.

We use the following, straight-forward functionality:

$$\mathcal{F}_S(\lambda, f, \alpha, pk_{ev}, m) := (\bot, \bot)$$

Proposition 3 (Signatures)**.** *If unforgeable delegatable functional signatures exist, then unforgeable signatures exist.*

Unforgeability against outsider adversaries (UnfO) suffices for regular signatures.

Identity Based Signatures. In an identity based signature scheme (IBS), publicly known identifiers ID serve as verification keys for signatures that were generated by the entity with identifier ID. We follow the general construction of [10], in which a trusted party generates a master secret key *msk* and extracts a signing key sk_{ID} from (msk,ID) for each entity with identifier ID. Formally, an IBS scheme is a tuple $\mathsf{IBS} = (\mathsf{Setup}_{\mathsf{IBS}}, \mathsf{Extract}_{\mathsf{IBS}}, \mathsf{Sig}_{\mathsf{IBS}}, \mathsf{Vf}_{\mathsf{IBS}})$ of efficient algorithms.

```
$\underline{\mathsf{Setup}_{\mathsf{IBS}}(\lambda):}$
$(pp, msk) \leftarrow \mathsf{Setup}(\lambda)$
$(sk_{sig}, pk_{sig}) \leftarrow \mathsf{KGen}_{sig}(pp, msk)$
$(sk_{ev}, pk_{ev}) \leftarrow \mathsf{KGen}_{ev}(pp, msk)$
set $\mathsf{IBS}.pp := (pp, pk_{sig}, pk_{ev})$
set $\mathsf{IBS}.msk := (msk, sk_{sig}, sk_{ev})$
output $(\mathsf{IBS}.msk, \mathsf{IBS}.pp)$

$\underline{\mathsf{Extract}_{\mathsf{IBS}}(pp, msk, \mathsf{ID}):}$
parse $\mathsf{IBS}.pp = (pp_{\mathcal{F}}, pk_{sig}, pk_{ev})$
parse $\mathsf{IBS}.msk = (msk_{\mathcal{F}}, sk_{sig}, sk_{ev})$
$\sigma_{\mathsf{ID}} \leftarrow \mathsf{Sig}(pp, sk_{sig},$
        $pk_{ev}, 1, (\mathsf{ID}, \perp))$
set $sk_{\mathsf{ID}} := (sk_{ev}, \sigma_{\mathsf{ID}}, \mathsf{ID})$
output $sk_{\mathsf{ID}}$

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

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

Figure 3: Construction of an IBS out of a DFSS

For our implication we derive a secret signing key sk_{sig} that is never given to any entity except for the trusted party. With this signing key, we generate secret keys by signing the individual entity's ID. This signature σ_{ID} constitutes the secret that a participant can use in order to sign their own messages. Signing means modifying σ_{ID} in a controlled way: We add the (real) message to the (signed) ID. (see Figure 8).

We define the functionality $\mathcal{F}_{\mathsf{IBS}}$ as follows.

$$\mathcal{F}_{\mathsf{IBS}}(\lambda, 1, \alpha, pk_{ev}, (\mathsf{ID}, \perp)) := (0, (\mathsf{ID}, \alpha))$$

Naturally, $\mathcal{F}_{\mathsf{IBS}}(\lambda, f, \cdot, \cdot, \cdot) := \perp$ for all $f \neq 1$. The security of the scheme comes from the fact that σ_{ID} cannot be generated by any entity but the trusted party.

Proposition 4 (Identity based signatures)**.** *If unforgeable private delegatable functional signatures exist, then unforgeable identity based signatures exist.*

Unforgeability against insider adversaries (UnfI) and privacy against insider adversaries is necessary for this construction. The fact that privacy is necessary might be somewhat surprising at first glance. However, since we use signatures as keys, the original signature σ_{ID} has to be kept secret. If it can be extracted from a derived signature, other participants can sign in the name of ID.

Sanitizable Signatures. Sanitizable signature schemes as in [18] allow a signature on m to be enriched with certain admissions adm. All possible modifications mod such that $\mathsf{adm}(\mathsf{mod}) = 1$ can be applied to the signature to derive signatures on $m' = \mathsf{mod}(m)$. The sanitizable signature scheme we create does not have a notion of accountability. However, extending the scheme to an accountable sanitizable signature scheme is an interesting future work. We construct a functionality $\mathcal{F}$ as follows, where $\mathsf{adm}_{\perp}$ always returns 0.

$$\mathcal{F}(\lambda, \mathsf{adm}, \mathsf{mod}, pk_{ev}, m) := \begin{cases} (\mathsf{adm}_{\perp}, \mathsf{mod}(m)) \text{ if } \mathsf{adm}(\mathsf{mod}) = 1 \\ \perp \text{ otherwise} \end{cases}$$

Proposition 5. *If unforgeable, private delegatable signature schemes exist, then unforgeable, immutable, private sanitizable signature schemes exist.*

Redactable Signatures. Redactable signatures as in [17] allow signatures that can be modified in specific ways by anyone that holds access to the signature. Intuitively we can construct a redactable signature scheme RSS from a functional signature scheme by making the secret evaluator key sk_{ev} public. This allows everyone to modify signatures according to a functionality $\mathcal{F}$ that implements the desired redaction predicate P as follows.

$$\mathcal{F}_P(\lambda, f, \alpha, pk_{ev}, m) := \begin{cases} (0, \alpha) \text{ if } P(m, \alpha) = 1 \wedge f = 1 \\ \perp \text{ otherwise} \end{cases}$$

Proposition 6 (Redactable signatures)**.** *If unforgeable, private delegatable signatures exist then unforgeable private redactable signatures exist.*

Blind signatures. Delegatable functional signatures also imply blind signatures. We explore this subject in detail in Section 4.1.

4 Possibility and Impossibility of DFS From OWFs

In this section we investigate the instantiability of DFS. In particular, we are interested in understanding which security property is "harder" to achieve. If we counter-intuitively are not interested in unforgeability, naturally we can construct a delegatable functional signature scheme DFSS unconditionally. A signature on m simply consists of the string "this is a signature for m". Obviously, this construction satisfies privacy against strong insiders in the sense of Definition 5 but not even unforgeability against outsiders.

Similarly, if we are not interested in privacy, DFS schemes can easily be constructed similarly to the construction in [14]. We assume a signature scheme S that is based on any one-way function. Now, the idea is that the signer simply signs a tuple consisting of the message together with the capability f and the public verification key of an evaluator. When evaluating, a changer adds his own signature on the previous signature together with α and the key of the following evaluator to the original signature. The verification procedure only accepts a signature if the signed trace of evaluations and delegations is legitimate w.r.t. the functionality $\mathcal{F}$. This scheme trivially satisfies unforgeability against strong insiders (cf. Definition 4) but none of our privacy notions. Thus, we obtain the following simple result:

Theorem 1. *If one-way functions exist, then there exists a unforgeable delegatable functional signature scheme.*

4.1 Impossibility of DFS from OWPs

In this section we prove an impossibility result showing that (D)FS cannot be constructed from OWP in a black-box way. The basic idea of our impossibility is to build a blind signature scheme in a black-box way. Since it is known that blind signature cannot be constructed from OWP only using black-box techniques [35], this implies that (D)FS cannot be constructed from OWF as well.

Blind Signatures and Their Security A blind signature scheme is an interactive protocol between a signer $\mathcal{S}$, holding a secret key sk_{BS} and a user $\mathcal{U}$ who wishes to obtain a signature on a message m such that the user cannot create any additional signatures and such that $\mathcal{S}$ remains oblivious about this message. We refer to Section 5.1 for formal definitions of commitment schemes and to Appendix B for formal definitions of blind signatures.

Building Blind Signatures from (D)FS The basic idea of our construction is as follows. The user chooses a message m, commits to the message and sends the commitment c on m to the signer. The signer signs the commitment, using a delegatable functional signature scheme and sends the signature σ_c back to the user. The user then calls $\mathsf{Eval}_{\mathcal{F}}$ with the open information o_m to derive a signature on m.

Given a commitment scheme $\mathsf{C} = (\mathsf{Commit}, \mathsf{Open})$ and a delegatable functional signature scheme $\mathsf{DFSS} = (\mathsf{Setup}, \mathsf{KGen}_{sig}, \mathsf{KGen}_{ev}, \mathsf{Sig}, \mathsf{Eval}_{\mathcal{F}}, \mathsf{Vf})$ for functionality $\mathcal{F}_C$, where $\mathcal{F}_C(\lambda, 1, \alpha, pk_{ev}, m) := (0, \mathsf{Open}(\alpha, m))$, we construct a blind signature scheme $\mathsf{BS} = (\mathsf{KG}_{\mathsf{BS}}, \langle \mathcal{S}, \mathcal{U} \rangle, \mathsf{Vf}_{\mathsf{BS}})$ as follows.[1]

$(\boldsymbol{sk}_{\mathsf{BS}}, \boldsymbol{pk}_{\mathsf{BS}}) \leftarrow \mathsf{KG}_{\mathsf{BS}}(1^\lambda)$**.** The key generation algorithm $\mathsf{KG}_{\mathsf{BS}}(1^\lambda)$ performs the following steps:

$$(msk, pp) \leftarrow \mathsf{Setup}(\lambda)$$
$$(sk_{sig}, pk_{sig}) \leftarrow \mathsf{KGen}_{sig}(pp, msk)$$
$$(sk_{ev}, pk_{ev}) \leftarrow \mathsf{KGen}_{ev}(pp, msk)$$
$$(sk_{\mathsf{BS}}, pk_{\mathsf{BS}}) \leftarrow (sk_{sig}, (pp, pk_{sig}, pk_{ev}, sk_{ev}))$$

Signing. The protocol for $\mathcal{U}$ to obtain a signature on message m is depicted in Figure 4 and consists of the following steps:

$\mathcal{U} \to \mathcal{S}$ The user sends a commitment c to the signer, where $(c, o_m) := \mathsf{Commit}(\lambda, m)$.

$\mathcal{S} \to \mathcal{U}$ The signer signs c together with the capability f and the public evaluator key of the user, obtaining $\sigma_c \leftarrow \mathsf{Sig}(sk_{sig}, pk_{ev}, 1, c)$. It sends σ_c to $\mathcal{U}$. The user calls $\mathsf{Eval}_{\mathcal{F}}$ with the open information o_m to derive a signature on m, as $\sigma_m \leftarrow \mathsf{Eval}_{\mathcal{F}}(sk_{ev}, pk_{sig}, o_m, pk'_{ev}, \sigma_c)$ and outputs (m, σ').

$b \leftarrow \mathsf{Vf}_{\mathsf{BS}}(\boldsymbol{pk}_{\mathsf{BS}}, \sigma, m)$**.** Verification $\mathsf{Vf}_{\mathsf{BS}}(pk_{\mathsf{BS}}, \sigma, m)$ returns $\mathsf{Vf}(pp, pk_{sig}, pk_{ev}, m, \sigma)$.

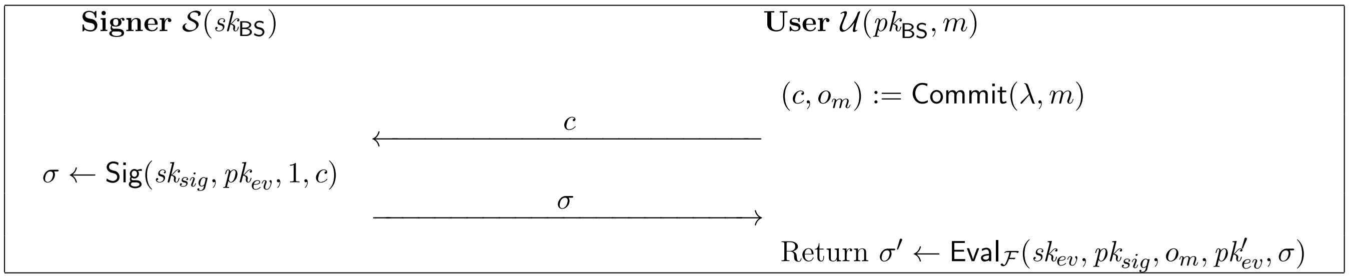
Signer $\mathcal{S}(sk_{\mathsf{BS}})$
User $\mathcal{U}(pk_{\mathsf{BS}}, m)$
$(c, o_m) := \mathsf{Commit}(\lambda, m)$
c
$\sigma \leftarrow \mathsf{Sig}(sk_{sig}, pk_{ev}, 1, c)$
σ
Return $\sigma' \leftarrow \mathsf{Eval}_{\mathcal{F}}(sk_{ev}, pk_{sig}, o_m, pk'_{ev}, \sigma)$

Figure 4: Issue protocol of the two move blind signature scheme.

[1]$\mathcal{F}_C$ outputs $\perp$ whenever $f \neq 1$.

Theorem 2. *If* $\mathsf{DFSS} = (\mathit{Setup}, \mathit{KGen}_{sig}, \mathit{KGen}_{ev}, \mathit{Sig}, \mathit{Eval}_{\mathcal{F}}, \mathit{Vf})$ *is an unforgeable and private delegatable signature scheme (both against insider attacks) and* $\mathsf{C} = (\mathsf{Commit}, \mathsf{Open})$ *is a commitment scheme which is both computationally binding and hiding, then the interactive signature scheme* $\mathsf{BS} = (\mathsf{KG}_{\mathsf{BS}}, \langle \mathcal{S}, \mathcal{U} \rangle, \mathsf{Vf}_{\mathsf{BS}})$ *as defined above is unforgeable and blind w.r.t. Definitions 19 and 20.*

Intuitively, unforgeability holds because the user can only obtain a signature on m if he calls $\mathsf{Eval}_{\mathcal{F}}$ on an authenticated commitment. This follows from the binding of the commitment scheme and from the unforgeability of the DFS. Blindness follows directly from the hiding property of the commitment scheme and from the privacy of our DFS. Note that the impossibility result of [34] rules out blind signature schemes that are secure against semi-honest adversaries.

We prove this theorem with the following two propositions.

Proposition 7. *If* $\mathsf{DFSS} = (\mathit{Setup}, \mathit{KGen}_{sig}, \mathit{KGen}_{ev}, \mathit{Sig}, \mathit{Eval}_{\mathcal{F}}, \mathit{Vf})$ *is an unforgeable delegatable signature scheme and* $\mathsf{C} = (\mathsf{Commit}, \mathsf{Open})$ *is a commitment scheme which is binding, then the interactive signature scheme* $\mathsf{BS} = (\mathsf{KG}_{\mathsf{BS}}, \langle \mathcal{S}, \mathcal{U} \rangle, \mathsf{Vf}_{\mathsf{BS}})$ *as defined above is unforgeable.*

Proof. Assume there is an efficient algorithm $\mathcal{A}$ that forges a signature for BS. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ against the unforgeability of DFSS. First, $\mathcal{B}$ receives (pp, pk_{sig}) from the $\mathsf{Initialize}$ algorithm of the UnfI challenger. Then it calls $\mathsf{Query[KGenS]}()$ to receive an evaluator key pair (sk_{ev}, pk_{ev}), sets $pk_{\mathsf{BS}} := (pp, pk_{sig}, pk_{ev}, sk_{ev})$ and simulates $\mathcal{A}(pk_{\mathsf{BS}})$. Whenever $\mathcal{A}$ interacts with its signature oracle with a (blinded) message c_i, then $\mathcal{B}$ calls $\mathsf{Query[Sign]}(pk_{ev}, f, c_i)$ and returns the resulting signature σ_i to $\mathcal{A}$.

Eventually, $\mathcal{A}$ stops, outputting $k+1$ message-signature pairs (m_i^*, σ_i^*) after k successful interactive signing procedures, $\mathcal{B}$ chooses one of them at random and outputs it as a (possible) forgery.

For the analysis observe that the functionality $\mathcal{F}_C$ only applies the Open algorithm to the signed message and only if $f = 1$ has been set. So for every signature σ_i that $\mathcal{A}$ received, only one application of $\mathsf{Eval}_{\mathcal{F}}$ is allowed and only the Open algorithm can be applied. Since C is a binding commitment scheme, every commitment can only be opened to a unique message, except for a negligible error probability. However, this means that one of the signatures σ_i^* must be a forgery for DFSS as it is either a signature on a new message, or an evaluation of a function that has not been allowed (and thus is not in the transitive hull $\mathcal{F}^*$ of any message that has been signed via $\mathsf{Query[Sign]}$).

If $\mathcal{A}$ constructs a forgery, $\mathcal{B}$ chooses the right message-signature pair (m_i^*, σ_i^*) with probability at least $\frac{1}{k}$, so the probability that $\mathcal{B}$ constructs a forgery is at least $\frac{1}{k}$ times the probability that $\mathcal{A}$ constructs a forgery. □

Remark. Note that f is necessary to ensure that $\mathsf{Eval}_{\mathcal{F}}$ is only called once, otherwise there is a simple attack against the scheme: The adversary $\mathcal{A}$ picks a message m and computes $(c_1, o_1) \leftarrow \mathsf{Commit}(m)$. Then $\mathcal{A}$ computes $(c_2, o_2) \leftarrow \mathsf{Commit}(c_1)$ and sends c_2 to the signer. Upon receiving a signature σ_{c_2}, the algorithm $\mathcal{A}$ uses $\mathsf{Eval}_{\mathcal{F}}$ to get a signature on $c_1 = \mathsf{Open}(c_2, o_2)$. Now $\mathcal{A}$ uses $\mathsf{Eval}_{\mathcal{F}}$ again to derive a signature on $m = \mathsf{Open}(c_1, o_1)$ and it outputs both signatures. Since $\mathcal{A}$ outputs two valid message-signature pairs (with two distinct message) after one successful interaction it breaks the unforgeability of BS.

Proposition 8. *If* DFSS = (*Setup*, *KGen*$_{sig}$, *KGen*$_{ev}$, *Sig*, *Eval*$_{\mathcal{F}}$, *Vf*) *is a private delegatable signature scheme and* C = (Commit, Open) *is a commitment scheme which is hiding, then the interactive signature scheme* BS = $(\mathsf{KG}_{\mathsf{BS}}, \langle\mathcal{S},\mathcal{U}\rangle, \mathsf{Vf}_{\mathsf{BS}})$ *as defined above is blind.*

Proof. We will show this proposition via a game-based proof. We will start with the original game $\mathsf{Blind}^{\mathsf{BS}}_{\mathcal{S}^*}(\lambda)$ for blindness (see Definition 20) and modify it until we reach a game in which the adversary can not observe any information that might help him in guessing the bit b.

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 34th 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 $\mathcal{G}_5$ 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 $\mathcal{G}_0$
– –$\mathsf{KG}_{\mathsf{BS}}$ – –
001 $(msk, pp) \leftarrow \mathsf{Setup}(\lambda)$
002 $(sk_{sig}, pk_{sig}) \leftarrow \mathsf{KGen}_{sig}(pp, msk)$
003 $(sk_{ev}, pk_{ev}) \leftarrow \mathsf{KGen}_{ev}(pp, msk)$
004 $(sk_{\mathsf{BS}}, pk_{\mathsf{BS}}) \leftarrow (sk_{sig}, (pp, pk_{sig}, pk_{ev}, sk_{ev}))$
– –find– –
005 $(m_0, m_1, state) \leftarrow \mathcal{A}(\mathsf{find}, sk_{\mathsf{BS}}, pk_{\mathsf{BS}})$
006 $b \leftarrow \{0,1\}$
– –issue– –
007 $(state, \sigma_{c_b}) \leftarrow \mathcal{A}(\mathsf{issue}state)^{\langle\cdot,U_b\rangle^1,\langle\cdot,U_{1-b}\rangle^1}$
– where $U_b$ computes –
008 $(c_b, o_b) \leftarrow \mathsf{Commit}(m_b)$
– and $U_{1-b}$ computes –
009 $(c_{1-b}, o_{1-b}) \leftarrow \mathsf{Commit}(m_{1-b})$
– unblind–
010 $\sigma_{m_0} \leftarrow \mathsf{Eval}_{\mathcal{F}}(pp, sk_{ev}, pk_{sig}, o_0, c_0, pk_{ev}, \sigma)$
011 $\sigma_{m_1} \leftarrow \mathsf{Eval}_{\mathcal{F}}(pp, sk_{ev}, pk_{sig}, o_1, c_1, pk_{ev}, \sigma)$
012 if $\sigma_{m_0} = \perp \vee \sigma_{m_1} = \perp$ then
013     $(\sigma_{m_0}, \sigma_{m_1}) := (\perp, \perp)$
– –guess– –
014 $b^* \leftarrow \mathcal{A}(\mathsf{quess}, state, \sigma_{m_0}, \sigma_{m_1})$
```

```
GAME $\mathcal{G}_1$
110 $\sigma_{m_0} \leftarrow \mathsf{Sig}(pp, sk_{sig}, pk_{ev}, 0, m_0, pk_{ev})$
111 $\sigma_{m_1} \leftarrow \mathsf{Sig}(pp, sk_{sig}, pk_{ev}, 0, m_1, pk_{ev})$

GAME $\mathcal{G}_2$
208 $(c_x, o_x) \leftarrow \mathsf{Commit}(0^n)$
209 $(c_y, o_y) \leftarrow \mathsf{Commit}(0^n)$
```

GAME $\mathcal{G}_0 \Rightarrow$ GAME $\mathcal{G}_1$: Since DFSS is private, we can create new signatures instead of calling $\mathsf{Eval}_{\mathcal{F}}$ on the signature of $\mathcal{A}$.

Claim 1. GAME $\mathcal{G}_0$ *and* GAME $\mathcal{G}_1$ *are computationally indistinguishable.*

Proof. Assume there is an efficient, malicious signer $\mathcal{A}$, which is able to distinguish both games. We show how to use $\mathcal{A}$ to build a distinguisher $\mathcal{B}$ that breaks the privacy property of DFSS. The algorithm $\mathcal{B}$ simulates GAME $\mathcal{G}_0$, but instead of calling $\mathsf{Eval}_{\mathcal{F}}$ in lines 10 and 11, it queries Query[Sign-$\mathcal{F}$]

$((pk_{ev}, \bot), (pk_{ev}, o_0), 1, c_0, \sigma_{c_0})$ and $\mathsf{Query[Sign\text{-}\mathcal{F}\ ]}((pk_{ev}, \bot), (pk_{ev}, o_1), 1, c_1, \sigma_{c_1})$ respectively. If $\mathcal{A}$ distinguishes Game $\mathcal{G}_0$ and Game $\mathcal{G}_1$ with probability noticeably larger than $\frac{1}{2}$, then $\mathcal{B}$ breaks the privacy property of DFSS by guessing the bit b of the challenger for CFA with probability noticeably larger than $\frac{1}{2}$.

□

Game $\mathcal{G}_1 \Rightarrow$ Game $\mathcal{G}_2$: In Game $\mathcal{G}_1$ the signature of the adversary is not used anymore and thus the commitment is not opened anymore. Consequently we can replace the commitments by commitments to zero.

Claim 2. *Game $\mathcal{G}_1$ and Game $\mathcal{G}_2$ are computationally indistinguishable.*

Proof. This follows from the hiding property of the commitment scheme. If there is an efficient, malicious signer $\mathcal{A}$, which is able to distinguish the two games, then we can use it to break the hiding property of C. □

In Game $\mathcal{G}_2$ the bit b is never used. The commitments and all signatures are completely independent of b. Thus, the probability that $\mathcal{A}$ guesses $b^* = b$ in Game $\mathcal{G}_2$ is exactly $\frac{1}{2}$. Since the games are (pairwise) computationally indistinguishable, the proposition holds. □

Combining Theorem 2 and the impossibility result of [35] (which is based on the work of Barak and Mahmoody [5]), we obtain the following result.

Corollary 1. *(Delegatable) functional signature schemes that are unforgeable and private against insider adversaries cannot be build from one-way permutations in a black-box way.*

Remark. Since this construction did not use the delegation property of the delegatable functional signature scheme, it should be possible to construct blind signatures from functional signatures, as defined by [14]. A DFS that is unforgeable and private against outsider adversaries is not ruled out by this corollary. Exploring whether the impossibility also holds in this case would be interesting.

5 Constructing DFS From Trapdoor Permutations

In this section we construct a delegatable functional signature scheme DFSS as defined in Section 2.1. Our construction is based on (regular) unforgeable signature schemes, a public-key encryption scheme, and a non-interactive zero-knowledge proof system. It is well known that these primitives can be constructed from (doubly enhanced) trapdoor permutations. 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 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 $|\sigma_1 \leftarrow \mathsf{Sig}_{\mathcal{S}}(pp, sk, m_1)| = |\sigma_2 \leftarrow \mathsf{Sig}_{\mathcal{S}}(pp, sk, m_2)|$

```
Initialize(λ):
  $(pp, msk) \leftarrow \mathsf{Setup}(\lambda)$
  $(sk, pk) \leftarrow \mathsf{KGen}_S(pp, msk)$
  store $(pp, sk, pk)$
  set $\mathcal{M} := \emptyset$
  output $(pp, pk)$

Query[Sign](m):
  retrieve $(pp, sk, pk)$
  $\sigma \leftarrow \mathsf{Sig}_S(pp, sk, m)$
  set $\mathcal{M} := \mathcal{M} \cup \{m\}$
  output $\sigma$

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

```
Initialize(λ):
  $b \xleftarrow{R} \{0,1\}$
  $(pp, msk) \leftarrow \mathsf{Setup}_{\mathcal{E}}(\lambda)$
  $(dk, ek) \leftarrow \mathsf{KGen}_{\mathcal{E}}(pp, msk)$
  store $(pp, ek, b)$
  output $(pp, ek)$

Query[Challenge]($m_0, m_1$):
  (only once)
  retrieve $(pp, ek, b)$
  $c \leftarrow \mathsf{Enc}_{\mathcal{E}}(pp, ek, m_b)$
    output $c$

Finalize($b^*$):
  if $b^* = b$
    output 1
  else
    output 0
```

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

Definition 6. *(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 7. *(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 8. *(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 9. *(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 10. *(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 11. *(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 [25, 31].

Definition 12. *(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 13. *(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 14. *(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 $D = (D_0, D_1)$:*

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

Commitment schemes. Here we recall the formal definitions of commitment schemes. A commitment scheme $\mathsf{C} = (\mathsf{Commit}, \mathsf{Open})$ is a tuple of efficient algorithms, where

Commit **(λ,m)** on input m (and a security parameter λ) returns both a commitment c and open information o.

Open **(c,o)** either returns a message m, if or an error symbol $\perp$

Definition 15. *A commitment scheme* $\mathsf{C} = (\mathsf{Commit}, \mathsf{Open})$ *is correct if for all messages m and for all* $(c, o) \in [\mathsf{Commit}(m)]$ *it holds that*

$$\mathsf{Open}(c, o) = m$$

Definition 16. *A commitment scheme* $\mathsf{C} = (\mathsf{Commit}, \mathsf{Open})$ *is binding, if for all PPT adversaries $\mathcal{A}$ the following probability is negligible in λ.*

$$Pr[m_0 \neq m_1 \wedge m_0 \neq \perp \neq m_1; m_0 := \mathsf{Open}(c, o_0), m_1 := \mathsf{Open}(c, o_1), (c, o_0, o_1) \leftarrow \mathcal{A}(\lambda)]$$

Definition 17. *A commitment scheme* $\mathsf{C} = (\mathsf{Commit}, \mathsf{Open})$ *is hiding, if for all PPT adversaries $\mathcal{A}$ and for all messages m_0, m_1 the following is negligible in λ.*

$$|Pr[m_0 \leftarrow \mathcal{A}(c); (c, o) \leftarrow \mathsf{Commit}(\lambda, m_0)] - Pr[m_0 \leftarrow \mathcal{A}(c); (c, o) \leftarrow \mathsf{Commit}(\lambda, m_1)]|$$

5.2 Our scheme

Our construction follows the encrypt and proof strategy and is completely general with respect to efficiently computable functionalities $\mathcal{F}$ with the exception that $\mathcal{F}$ may allow only for up to n applications of $\mathsf{Eval}_{\mathcal{F}}$. We let the signer choose how many applications he allows by defining f as a tuple $(f', k) \in \mathcal{P}_f \times \{0, \ldots, n\}$. We achieve the strong notion of privacy under chosen function attack (CFA) according to Definition 5 by applying the following idea : If the signer choses a number of k possible applications of $\mathsf{Eval}_{\mathcal{F}}$, 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 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 $\mathsf{Eval}_{\mathcal{F}}$ that are still allowed. Figure 6 shows the construction in more detail.

```
$\underline{\mathsf{Setup}(1^\lambda):}$
$CRS \leftarrow \mathsf{KGen}_{\mathsf{NIZK}}(1^\lambda)$
$(msk_{\mathcal{S}}, pp_{\mathcal{S}}) \leftarrow \mathsf{Setup}_{\mathcal{S}}(1^\lambda)$
$(msk_{\mathcal{E}}, pp_{\mathcal{E}}) \leftarrow \mathsf{Setup}_{\mathcal{E}}(1^\lambda)$
$(\widetilde{dk}, \widetilde{ek}) \leftarrow \mathsf{KGen}_{\mathcal{E}}(pp_{\mathcal{E}}, msk_{\mathcal{E}})$
$pp := (CRS, pp_{\mathcal{S}}, pp_{\mathcal{E}}, \widetilde{ek})$
$msk := (msk_{\mathcal{S}}, msk_{\mathcal{E}})$
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 $(sk_{sig}, pk_{sig})$

$\underline{\mathsf{KGen}_{ev}(pp, msk):}$
parse $pp = (CRS, pp_{\mathcal{S}}, pp_{\mathcal{E}}, \widetilde{ek})$
parse $msk = (msk_{\mathcal{S}}, msk_{\mathcal{E}})$
$(ssk_{\mathcal{F}}, vk_{\mathcal{F}}) \leftarrow \mathsf{KGen}_{\mathcal{S}}(pp_{\mathcal{S}}, msk_{\mathcal{S}})$
$(dk, ek) \leftarrow \mathsf{KGen}_{\mathcal{E}}(pp_{\mathcal{E}}, msk_{\mathcal{E}})$
$sk_{ev} := (ssk_{\mathcal{F}}, dk)$
$pk_{ev} := (vk_{\mathcal{F}}, ek)$
output $(sk_{ev}, pk_{ev})$
```

```
$\underline{\mathsf{Sig}(pp, sk_{sig}, pk_{ev}, (f, k), m):}$
parse $pp = (CRS, pp_{\mathcal{S}}, pp_{\mathcal{E}}, \widetilde{ek})$
parse $pk_{ev} = (vk_{\mathcal{F}}, ek)$
parse $sk_{sig} = (ssk_{\mathcal{S}}, pk_{sig}))$
$h_k := (f, m, pk_{ev}, k)$
$\sigma_k \leftarrow \mathsf{Sig}_{\mathcal{S}}(pp_{\mathcal{S}}, ssk_{\mathcal{S}}, h_k; r_{\mathcal{S}})$
$s_k \leftarrow \mathsf{Enc}_{\mathcal{E}}(pp_{\mathcal{E}}, \widetilde{ek}, (\sigma_k, h_k); r_{s_k})$
For $i \in \{0, \ldots, n\} \setminus \{k\}$
    $\sigma_i := 0^{|\sigma_k|}$
    $h_i := (0^{\ell_f(\lambda)}, 0^{\ell_m(\lambda)}, 0^{|pk_{ev}|}, 0)$
    $s_i \leftarrow \mathsf{Enc}_{\mathcal{E}}(pp_{\mathcal{E}}, \widetilde{ek}, (\sigma_i, h_i); r_{s_i})$
$d \leftarrow \mathsf{Enc}_{\mathcal{E}}(pp_{\mathcal{E}}, ek, (f, k, \sigma_k); r_d)$
$S := (s_0, \ldots, s_n)$
$x = (pp, pk_{sig}, pk_{ev}, S, d, m)$
$\omega = (f, n, r_d)$
with $\omega_k = (\sigma_k, r_{\mathcal{S}}, k)$
$\Pi \leftarrow P_{\mathsf{NIZK}}(CRS, x, \omega)$
$\sigma := (S, d, \Pi)$
output $\sigma$

$\underline{\mathsf{Vf}(pp, pk_{sig}, pk_{ev}, m, \sigma):}$
parse $pp = (CRS, pp_{\mathcal{S}}, pp_{\mathcal{E}}, \widetilde{ek})$
parse $pk_{sig} = (vk_{\mathcal{S}}, \widetilde{ek})$
parse $pk_{ev} = (vk_{\mathcal{F}}, ek)$
parse $\sigma = (S, d, \Pi)$
$x := (pp_{\mathcal{S}}, pp_{\mathcal{E}}, pk_{sig}, pk_{ev}, S, d, m, CRS)$
$b \leftarrow \mathsf{Vf}_{\mathsf{NIZK}}(CRS, x, \Pi)$
output $b$
```

```
$\underline{\mathsf{Eval}_{\mathcal{F}}(pp, sk_{ev}, pk_{sig}, \alpha, m, pk'_{ev}, \sigma):}$
parse $pp = (CRS, pp_{\mathcal{S}}, pp_{\mathcal{E}}, \widetilde{ek})$
parse $sk_{ev} = (ssk_{\mathcal{F}}, dk)$
parse $pk'_{ev} = (vk'_{\mathcal{F}}, ek')$
parse $\sigma = (S, d, \Pi)$
$(f, i, \sigma_i) \leftarrow \mathsf{Dec}_{\mathcal{E}}(pp_{\mathcal{E}}, dk, d)$
$x = (pp, pk_{sig}, pk_{ev}, S, d, m)$
if $pk_{ev} = (vk_{\mathcal{F}}, ek)$ belongs to $sk_{ev}$
$\wedge\, \mathsf{Vf}_{\mathsf{NIZK};Z_i}(CRS, x, \Pi) = 1$
    $(\hat{f}, \hat{m}) := \mathcal{F}(\lambda, f, \alpha, pk'_{ev}, m)$

    $h_{i-1} := (\hat{f}, \hat{m}, pk'_{ev}, i-1)$
    $\hat{\sigma}_{i-1} \leftarrow \mathsf{Sig}_{\mathcal{S}}(pp_{\mathcal{S}}, ssk_{\mathcal{F}}, h_{i-1}; r_{\mathcal{S}})$
    $s_{i-1} \leftarrow \mathsf{Enc}_{\mathcal{E}}(pp_{\mathcal{E}}, \widetilde{ek}, (\hat{\sigma}_{i-1}, h_{i-1}); r_s)$
    $d \leftarrow \mathsf{Enc}_{\mathcal{E}}(pp_{\mathcal{E}}, ek', (\hat{f}, i-1, \sigma_{i-1}); r_d)$
    $\hat{x} = (pp, pk_{sig}, pk'_{ev}, S, d, \hat{m})$
    $\omega = (\hat{f}, i-1, r_d)$
    with $\omega_{i-1} = (\sigma_{i-1}, r_{\mathcal{S}}, \Pi, pk_{ev}, m, f, \alpha)$
    $\hat{\Pi} \leftarrow P_{\mathsf{NIZK}}(CRS, x, \omega)$
    $\hat{\sigma} := (S, d, \Pi)$
    output $\hat{\sigma}$
else
    output $\perp$
```

Figure 6: Construction of a DFSS

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 (cf. Section 5.1) 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, where $L_n :\ x = (pp_{\mathcal{S}}, pp_{\mathcal{E}}, pk_{sig}, pk_{ev}, S, m, CRS, f, \sigma) \in L_n$ means that there exists a witness $\omega = (r, k)$ such that

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

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

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

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

$$\mathsf{Vf}_{\mathsf{NIZK}_i}(CRS, (pp_{\mathcal{S}}, pp_{\mathcal{E}}, pk_{sig}, pk_{ev}, S, m, CRS, f, \sigma)) = 1 \ \wedge\ d \leftarrow \mathsf{Enc}_{\mathcal{E}}(pp_{\mathcal{E}}, ek, (f, i, \sigma); r_d).$$

5.3 Security

Concerning security, we show the following theorem.

Theorem 3. *If $\mathcal{E}$ is a public key encryption scheme that is secure against chosen ciphertext attacks (CCA-2), $\mathcal{S}$ a length preserving unforgeable signature scheme with a simple key generation, and NIZK is a sound non-interactive proof scheme (Definition 13) that is zero knowledge (Definition 14), the construction presented in this section is* unforgeable against outsider and (strong) insider attacks *and* secure against chosen function attacks (CFA) against outsiders and (strong) insiders

Proof. The theorem follows directly from Lemma 2 and Lemma 4. □

Lemma 1. *If $\mathcal{E}$ is a 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 13), then the construction* DFSS *presented in Section 5 is* unforgeable against outsider and (strong) insider attacks *according to Definition 4.*

Lemma 2. *If $\mathcal{E}$ is a 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 13), then the construction* DFSS *presented in Section 5 is* unforgeable against outsider and (strong) insider attacks *according to Definition 4.*

Given an adversary $\mathcal{A}$ that breaks the unforgeability of our construction we construct an efficient adversary $\mathcal{B}$ that breaks the underlying signature scheme .

Proof. By Proposition 1 it suffices to show unforgeability against an S-Insider adversary. Assume towards contradiction that DFSS is not unforgeable against strong insider attacks. Then there exists an efficient adversary $\mathcal{A} := \mathcal{A}_{\text{S-Insider}}$ that makes at most $p(\lambda)$ many steps for a polynomial p

and that wins the game $\mathsf{Unf}(\mathsf{DFSS}, \mathcal{F}, \mathcal{A}_{\text{S-Insider}}, \lambda)$, formalized in Definition 4, with non-negligible probability. Since $\mathcal{A}$ makes at most $p(\lambda)$ many steps, $\mathcal{A}$ invokes the oracle $\mathsf{Query[KgenP]}$ 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 $\mathsf{Unf}(\mathsf{DFSS}, \mathcal{F}, \mathcal{A}, \lambda)$. 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 inputs of these oracles are provided by $\mathcal{A}$ upon calling them and are thus sent to $\mathcal{B}$. The algorithm $\mathcal{B}$ handles the oracle queries from $\mathcal{A}$ as follows:

Query[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 $(sk_{ev}, pk_{ev}) \leftarrow \mathsf{KGen}_{\mathcal{S}}(pp_{\mathcal{C}}, \varepsilon)$, stores this pair, and sends pk_{ev} to $\mathcal{A}$.

Query[Sign] $(\boldsymbol{pk^*_{ev}}, f, g, m)$**:** If $z \neq 0$, the algorithm $\mathcal{B}$ computes all necessary values locally exactly as a challenger for $\mathsf{Unf}(\mathsf{DFSS}, \mathcal{F}, \mathcal{A}, \lambda)$ 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^*_{ev}, f 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 $h_k := (f, m, pk_{ev}, k)$ and invokes $\mathsf{Query[Sig]}_{\mathcal{C}}(h_k)$. It sets σ_k to the output of the challenger and otherwise proceeds as above.

Query[Eval] $(\boldsymbol{pk^*_{ev}}, \alpha, m, \boldsymbol{pk'_{ev}}, \sigma)$**:** Parse $pk^*_{ev} = (vk, ek)$. $\mathcal{B}$ behaves differently depending on the value of vk.

If the key pk^*_{ev} is a key for which $\mathcal{B}$ knows a secret key (in particular it does not contain the challenge key $vk_{\mathcal{C}}$), $\mathcal{B}$ computes all necessary values locally exactly as a challenger for $\mathsf{Unf}(\mathsf{DFSS}, \mathcal{F}, \mathcal{A}, \lambda)$ would. For computing the values locally, $\mathcal{B}$ needs to know pp (publicly known), a value for sk^*_{ev} corresponding to pk^*_{ev} (discussed below), pk_{sig} (known to $\mathcal{B}$) and the values for α, m, pk'_{ev} and σ (provided by $\mathcal{A}$). There are four cases for sk^*_{ev}. If pk^*_{ev} 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 $sk^*_{ev} = (ssk_{\mathcal{F}}, dk)$ itself. The same applies if pk^*_{ev} was output by $\mathsf{Query[KGenS]}$. If pk^*_{ev} was registered by $\mathcal{A}$ via $\mathsf{Query[RegKey]}$, $\mathcal{B}$ uses the corresponding (registered) key sk^*_{ev}. If none of the three cases applies, then the key pk^*_{ev} is unknown and $\mathcal{B}$ returns $\perp$ instead.

If the key pk^*_{ev} is the key in which $\mathcal{B}$ has embedded its own challenge key ($vk = vk_{\mathcal{C}}$), a corresponding value $ssk_{\mathcal{F}}$ (the first part of the secret key sk^*_{ev} corresponding to pk^*_{ev}) is not known to $\mathcal{B}$. This key is necessary to sign the value $h = (\hat{f}, \hat{m}, pk'_{ev}, k-1)$. 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($m^*, \sigma^*, \boldsymbol{pk}^*_{ev}$): Eventually, $\mathcal{A}$ invokes $\mathsf{Finalize}$ on a tuple $(m^*, \sigma^*, pk^*_{ev})$, 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 $\mathsf{Vf}(pp, pk_{sig}, pk^*_{ev}, m^*, \sigma^*)$. 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 3. *The algorithm $\mathcal{B}$ is efficient.*

Proof for Claim 3. The algorithm $\mathcal{B}$ simulates a challenger for $\mathsf{Unf}(\mathsf{DFSS}, \mathcal{F}, \mathcal{A}, \lambda)$ 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 $\mathsf{Unf}(\mathsf{DFSS}, \mathcal{F}, \mathcal{A}, \lambda)$ 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 4. *The algorithm $\mathcal{B}$ perfectly simulates a challenger for* $\mathsf{Unf}(\mathsf{DFSS}, \mathcal{F}, \mathcal{A}, \lambda)$.

Proof for Claim 4. 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 8 the values pp and msk are identically distributed to values for pp and msk generated by a challenger for $\mathsf{Unf}(\mathsf{DFSS}, \mathcal{F}, \mathcal{A}, \lambda)$. 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 $\mathsf{Unf}(\mathsf{DFSS}, \mathcal{F}, \mathcal{A}, \lambda)$.

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 8), 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 $\mathsf{Unf}(\mathsf{DFSS}, \mathcal{F}, \mathcal{A}, \lambda)$ 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 8), 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 $\mathsf{Unf}(\mathsf{DFSS}, \mathcal{F}, \mathcal{A}, \lambda)$. On them it performs a perfect simulation of Query[KGenS]. Thus, the resulting keys have the same distribution as the keys output by Query[KGenS] of the challenger.

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

Simulation of Query[Sign] and Query[Eval]: $\mathcal{B}$ 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 $\mathsf{Unf}(\mathsf{DFSS}, \mathcal{F}, \mathcal{A}, \lambda)$ sends to $\mathcal{A}$, the algorithm $\mathcal{B}$ perfectly simulates a challenger for $\mathsf{Unf}(\mathsf{DFSS}, \mathcal{F}, \mathcal{A}, \lambda)$. □

Claim 5. *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 5. First we show the following statement: Whenever $\mathcal{A}$ produces a forgery $(m^*, \sigma^*, pk^*_{ev})$, then σ^* is of the form $\sigma^* = (S, d, \pi)$. Moreover, $S = (s_0, \ldots, s_{n+1})$ contains the encryption s_x of a signature σ_x such that:

- σ_x verifies for a message m_x under a key vk^*
- vk^* either equals pk_{sig} or that has been sent to $\mathcal{A}$ as an answer to an oracle query Query[KGenP]
- m_x a message that has not been sent to Query[Sign] or achieved as result of Query[Eval].

Assume that $\mathcal{A}$ invokes Finalize with $(m^*, \sigma^*, pk^*_{ev})$ such that $(m^*, \sigma^*, pk^*_{ev})$ constitutes a forgery for DFSS. Technically: If our algorithm $\mathcal{B}$ would simulate the Finalize algorithm (as in Figure 7), it would output 1.[2]

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

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

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

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

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 $h_k = (f, m, pk_{ev}, k)$. The NIZK makes sure that m^* is in the transitive hull $\mathcal{F}^*(\lambda, (f, m))$ and that all transformations are legitimized by the previous ones (depending on the intermediate α's).

We distinguish the following cases:

$i = 0$: There was no call to Query[Sig] with parameters $(pk_{ev}, (f, k), m)$. Thus, $\mathcal{B}$ never sent h_k to 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 Query[Sig] with parameters $(pk_{ev}, (f, k), m)$. And for all $0 < j \leq i$ there was a call to Query[Eval] with parameters $(pk_{evj}, \alpha_j, m_j, pk'_{evj}, \sigma'_j)$, such that $h_{k-j} = (f_j, m_j, pk'_{evj}, k - j)$ with $(f_j, m_j) = \mathcal{F}(\lambda, f_{j-1}, \alpha_j, pk'_{evj}, m_{j-1})$, but there was no call to Query[Eval] with parameters $(pk_{evi}, \alpha_i, m_i, pk'_{evi}, \sigma'_i)$, such that $h_{k-i} = (f_i, m_i, pk'_{evi}, k - i)$ with $(f_i, m_i) = \mathcal{F}(\lambda, f_{i-1}, \alpha_i, pk'_{evi}, m_{i-1})$, where $f_0 = f$ and $m_0 = m$.

Thus, $\mathcal{B}$ never sent h_i to 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 $(pk_{ev}, (f, k), m)$. And for all $0 < j \leq k$ there was a call to $\mathsf{Query[Eval]}$ with parameters $(pk_{evj}, \alpha_j, \beta_j, m_j, pk'_{evj}, \sigma'_j)$, such that $h_{k-j} = (f_j, m_j, pk'_{evj}, k - j)$ with $(f_j, m_j) = \mathcal{F}(\lambda, f_{j-1}, \alpha_j, pk'_{evj}, m_{j-1}$. The NIZK makes sure that at most k transformations of the original message exist. Thus, all transformations have been done via calls to $\mathsf{Query[Eval]}$, which means that $(m^*, \sigma^*, pk^*_{ev})$ 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 $\mathsf{Query[KGenP]}$. Note that if, by chance, $vk^* = vk_{\mathcal{C}}$, then σ_x is a valid forgery for the message h_x. By Claim 4, $\mathcal{B}$ performs a perfect simulation of a challenger for $\mathsf{Unf}(\mathsf{DFSS}, \mathcal{F}, \mathcal{A}, \lambda)$ (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 5, 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 3, $\mathcal{B}$ is an efficient algorithm. This concludes the proof.

□

Lemma 3. *If $\mathcal{E}$ is a public key encryption scheme that is secure against chosen ciphertext attacks (CCA-2), and the interactive proof scheme NIZK is zero knowledge (Definition 14), then the construction* DFSS *presented in Section 5 is* secure against chosen function attacks (CFA) *as in Definition 5.*

Lemma 4. *If $\mathcal{E}$ is a public key encryption scheme that is secure against chosen ciphertext attacks (CCA-2), and the interactive proof scheme NIZK is zero knowledge (Definition 14), then the construction* DFSS *presented in Section 5 is* secure against chosen function attacks (CFA) *as in Definition 5.*

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

Proof. Let $\mathsf{DFSS} = (\mathsf{Setup}, \mathsf{KGen}_{sig}, \mathsf{KGen}_{ev}, \mathsf{Sig}, \mathsf{Eval}_{\mathcal{F}}, \mathsf{Vf})$ 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 $\mathsf{CFA}(\mathsf{DFSS}, \mathcal{F}, \mathcal{A}_{\text{S-Insider}}, \lambda)$ from Definition 5 with non negligible advantage. For simplicity we will write $\mathcal{A}$ for $\mathcal{A}_{\text{S-Insider}}$ in this proof.

Claim 6. *If $\mathcal{A}$ invokes the challenge oracle $\mathsf{Query[Sign\text{-}\mathcal{F}]}$ at most once, then the advantage of $\mathcal{A}$ is negligible.*

Proof for Claim 6. The challenger uses the uniformly distributed value b only when $\mathsf{Query[Sign\text{-}\mathcal{F}]}$ is called. Thus, if $\mathcal{A}$ does not call $\mathsf{Query[Sign\text{-}\mathcal{F}]}$, the advantage of $\mathcal{A}$ is 0.

For the case that $\mathcal{A}$ calls $\mathsf{Query[Sign\text{-}\mathcal{F}]}$ 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 5 where $b = 0$. As by our claim $\mathcal{A}$ calls $\mathsf{Query[Sign\text{-}\mathcal{F}]}$ only once we will simplify the notation of the game by making the call to $\mathsf{Query[Sign\text{-}\mathcal{F}]}$ explicit. Moreover we make the invokation of $\mathsf{Initialize}$ explicit as we will modify it in the following games. The oracles that $\mathcal{A}$ can access (aside from $\mathsf{Query[Sign\text{-}\mathcal{F}]}$) are as they are formalized in Definition 5.

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 34th 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 $\mathcal{G}_5$ 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 $msk := (msk_\mathcal{S}, msk_\mathcal{E})$
 – –$\mathsf{KGen}_{sig}$ – –
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 $(pp, sk_{sig}, pk_{sig})$ to $\mathcal{A}$–
011 $c \leftarrow \mathcal{A}_1^{O_{pp,msk,ssk_s}}(pp, pk_{sig})$
 – –Query[Sign-$\mathcal{F}$] – –
012 parse $c = ([pk_{ev}, \alpha]_0^t, t, m_0, \sigma_0)$
013 if $(\cdot, pk_{ev}[t]) \notin K_\mathcal{C}$   $out := \bot$
014 if $\mathsf{Vf}(pp, pk_{sig}, pk_{ev}[0], m_0, \sigma_0) \neq 1$
            then output $\bot$
015 if $\neg\exists sk^*_{ev}.\ (sk^*_{ev}, pk_{ev}[0]) \in \mathcal{K}_\mathcal{C}$
            then output $\bot$
016 extract $(f_0, k)$ from $\sigma_0$ using $sk^*_{ev}$
017 for $i \in \{1, \ldots, t\}$
018     if $\neg\exists sk^*_{ev} = (ssk_\mathcal{F}, dk).\ (sk^*_{ev}, pk_{ev}[i-1]) \in \mathcal{K}_\mathcal{F}$
019         $out := \bot$
020     $(f_i, m_i) := \mathcal{F}(\lambda, f_{i-1}, \alpha, pk_{ev}[i], m_{i-1})$

021     $q_i := (pp, sk^*_{ev}, pk_{sig}, \alpha[i], m_{i-1}, pk_{ev}[i], \sigma_{i-1})$
022     parse $pk_{ev}[i] = (vk_\mathcal{F}, ek)$
023     parse $\sigma_{i-1} = (S, d, \Pi)$
024     $(f_{i-1}, j, \varsigma_j) \leftarrow \mathsf{Dec}_\mathcal{E}(pp_\mathcal{E}, dk, d)$
025     $x = (pp, pk_{sig}, pk_{ev}[i-1], S, d, m_{i-1})$
026     if $\mathsf{Vf}_{\mathsf{NIZK};Z_j}(CRS, x, \Pi) = 1$
027         $h_{j-1} := (f_i, m_i, pk_{ev}[i], j-1)$
028         $\hat{\varsigma}_{j-1} \leftarrow \mathsf{Sig}_\mathcal{S}(pp_\mathcal{S}, ssk_\mathcal{F}, h_{j-1}; r_\mathcal{S})$
029         $s_{j-1} \leftarrow \mathsf{Enc}_\mathcal{E}(pp_\mathcal{E}, \widetilde{ek}, (\hat{\varsigma}_{j-1}, h_{j-1}); r_s)$
030         $\hat{d} \leftarrow \mathsf{Enc}_\mathcal{E}(pp_\mathcal{E}, ek, (f_i, j-1, \varsigma_{j-1}); r_d)$
031         $\hat{x} = (pp, pk_{sig}, pk_{ev}[i], S, \hat{d}, m_i)$
032         $\omega = (f_i, j-1, r_d)$
033         with $\omega_{j-1} = (\varsigma_{j-1}, r_\mathcal{S}, \Pi, pk_{ev}[i-1],$
                            $m_{i-1}, f_{i-1}, \alpha[i], )$
034         $\hat{\Pi} \leftarrow P_{\mathsf{NIZK}}(CRS, x, \omega)$
035         $\sigma_i := (S, \hat{d}, \hat{\Pi})$
036     else
037         $\sigma_i := \bot$
038 if $\sigma_t \neq \bot$
039     $h_{k-t} := (f_t, m_t, pk_{ev}[t], k-t)$
040     $\varsigma_{k-t} \leftarrow \mathsf{Sig}_\mathcal{S}(pp_\mathcal{S}, ssk_\mathcal{S}, h_{k-t}; r_\mathcal{S})$
041     $s_{k-t} \leftarrow \mathsf{Enc}_\mathcal{E}(pp_\mathcal{E}, \widetilde{ek}, (\varsigma_{k-t}, h_{k-t}); r_{s_{k-t}})$
042     For $j \in \{0, \ldots, n\} \setminus \{k-t\}$
043         $\varsigma_j := 0^{|\varsigma_{k-t}|}$
044         $h_j := (0^{\ell_p(\lambda)}, 0^{\ell_m(\lambda)}, 0^{|pk_{ev}[t]|}, 0)$
045         $s_j \leftarrow \mathsf{Enc}_\mathcal{E}(pp_\mathcal{E}, \widetilde{ek}, (\varsigma_j, h_j); r_{s_i})$
046     $d \leftarrow \mathsf{Enc}_\mathcal{E}(pp_\mathcal{E}, ek, (f_t, k-t, \varsigma_{k-t}; r_d))$
047     $S := (s_0, \ldots, s_n)$
048     $x := (pp, pk_{sig}, pk_{ev}[t], S, d, m_t)$
049     $\omega := (f_t, n, r_d)$
050     with $\omega_{k-t} := (\varsigma_{k-t}, r_\mathcal{S}, k-t)$
051     $\Pi \leftarrow \mathcal{P}(CRS, x, \omega)$
052     $\sigma := (S, d, \Pi)$
053 else
054     $\sigma := \sigma_t$
055 if $out \neq \bot$ then $out := \sigma$
056 $b^* \leftarrow \mathcal{A}_2(out)$
```

Game $\mathcal{G}_1$

```
102 $(CRS, state) \leftarrow \mathcal{S}_0(1^\lambda)$
134 $\hat{\Pi} \leftarrow \mathcal{S}_1(state, x)$
151 $\Pi \leftarrow \mathcal{S}_1(state, x)$
```

Game $\mathcal{G}_2$

```
229         $s_{j-1} \leftarrow \mathsf{Enc}_{\mathcal{E}}(pp_{\mathcal{E}}, \widetilde{ek}, (0^{|\varsigma_{j-1}|}, (0^{\ell_p(\lambda)}, 0^{\ell_m(\lambda)}, 0^{|pk_{ev}[t]|}, 0)); r_s)$
230         $\hat{d} \leftarrow \mathsf{Enc}_{\mathcal{E}}(pp_{\mathcal{E}}, ek, (0^{|f_t|}, 0, 0^{|\varsigma_{j-1}|}); r_d)$
241     $s_{k-t} \leftarrow \mathsf{Enc}_{\mathcal{E}}(pp_{\mathcal{E}}, \widetilde{ek}, (0^{|\varsigma_{k-t}|}, (0^{\ell_p(\lambda)}, 0^{\ell_m(\lambda)}, 0^{|pk_{ev}[t]|}, 0); r_{s_{k-t}})$
246     $d \leftarrow \mathsf{Enc}_{\mathcal{E}}(pp_{\mathcal{E}}, ek, (0^{|f_t|}, 0, 0^{|\varsigma_{k-t}|}); r_d)$
```

Game $\mathcal{G}_3$

```
301 $b := 1$
338 if $\mathit{false}$
```

Game $\mathcal{G}_4$

```
429         $s_{j-1} \leftarrow \mathsf{Enc}_{\mathcal{E}}(pp_{\mathcal{E}}, \widetilde{ek}, (\varsigma_{j-1}, h_{j-1}); r_s)$
430         $\hat{d} \leftarrow \mathsf{Enc}_{\mathcal{E}}(pp_{\mathcal{E}}, ek, (f_t, j-1, \varsigma_{j-1}); r_d)$
```

Game $\mathcal{G}_5$

```
501 $CRS \leftarrow \mathsf{KGen}_{\mathsf{NIZK}}(1^\lambda)$
534 $\Pi \leftarrow \mathcal{P}(CRS, x, \omega)$
```

Game $\mathcal{G}_0$ ⇒ Game $\mathcal{G}_1$: Since NIZK is zero knowledge, there exists an 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 $\mathsf{Setup}_{\mathsf{NIZK}}$. The simulator is allowed to keep state from $\mathcal{S}_0$ to $\mathcal{S}_1$. 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 7. 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 14). 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 and d contain only descriptions of zero-strings: we put encryptions of zero strings in all s_j for $j \in \{0, \dots, n\}$ instead of leaving an encryption of a signature ς_{k-t} together with its message

h_{k-t} at position $k-t$ and in an encryption of a zero string in d instead of an encryption of ς_{k-t} together with f_t and $k-t$.

To compensate for the loss of information in d, we store the tuple $(f_t, k-t, \varsigma_{k-t})$ together with the (supposed) ciphertext d. Whenever $\mathsf{Query[Eval]}$ is called and within the call one of the ciphertexts d is placed, we look up the values $(f_t, k-t, \varsigma_{k-t})$ instead of decrypting d. The same applies to the decryption in line 22 of our game.

Claim 8. *Game $\mathcal{G}_1$ and Game $\mathcal{G}_2$ are computationally indistinguishable.*

Proof. If the games could be distinguished by a PPT distinguisher, then we could construct an efficient distinguisher that breaks the CCA-2 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 or d (as in Game $\mathcal{G}_1$) and an S or d that consists only of encryptions of zero-strings (as in Game $\mathcal{G}_2$) can also be used to break the CCA-2 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 CCA-2 security of $\mathcal{E}$.

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

□

Game $\mathcal{G}_2 \Rightarrow$ Game $\mathcal{G}_3$: In Game $\mathcal{G}_3$, the bit b is set to 1 instead of 0. However, b is never used explicitly in the game. Moreover we always use the signature generated by $\mathsf{Eval}_{\mathcal{F}}$ (from line 33) instead of the fresh signature (from line 50).

Claim 9. *Game $\mathcal{G}_2$ and Game $\mathcal{G}_3$ are computationally indistinguishable.*

Proof. In both cases S and d are encryptions of zero strings (under the same keys) and in both cases Π is a proof generated by $\mathcal{S}_1$ for the same statement $x = (pp, pk_{sig}, pk_{ev}[t], S, d, m_t)$. Since S_1 does not receive a witness, the proofs are based on the same arguments. □

Game $\mathcal{G}_3 \Rightarrow$ Game $\mathcal{G}_4$: The game Game $\mathcal{G}_4$ is identical to Game $\mathcal{G}_3$ except for the fact that S and d are "normal" encryptions again (not encryptions of zero strings).

Claim 10. *Game $\mathcal{G}_3$ and Game $\mathcal{G}_4$ are computationally indistinguishable.*

Proof. The same argument as for Game $\mathcal{G}_1$ and Game $\mathcal{G}_2$ applies here. If the games could be distinguished, we could construct an efficient distinguisher for the encryption scheme.

Note that we do not need to revert the encryptions in lines 39 and 44 as they are within the "*if false*"-block. □

Game $\mathcal{G}_4 \Rightarrow$ Game $\mathcal{G}_5$: In Game $\mathcal{G}_5$ we replace the simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$ with the original $\mathsf{Setup}_{\mathsf{NIZK}}$ and $\mathcal{P}$ algorithms again.

Claim 11. *Game $\mathcal{G}_4$ and Game $\mathcal{G}_5$ are computationally indistinguishable.*

Proof. As for Claim 7, the indistinguishability again follows from the fact that NIZK is zero knowledge. If a PPT distinguisher could distinguish between GAME $\mathcal{G}_4$ and GAME $\mathcal{G}_5$, we could construct an efficient distinguisher for NIZK. $\square$

As we have shown, the games GAME $\mathcal{G}_0$ and GAME $\mathcal{G}_5$ are computationally indistinguishable. However, GAME $\mathcal{G}_0$ perfectly models the case, where an adversary plays against a challenger for CFA when $b = 0$, whereas GAME $\mathcal{G}_5$ perfectly models the case, where an adversary plays against a challenger for CFA when $b = 1$. Since the games are (pairwise) computationally distinguishable, the cases are also computationally indistinguishable and thus the advantage of $\mathcal{A}$ is negligible. This concludes the proof for Claim 6 $\square$

Via hybrid argument we reduce the case in which the adversary might make polynomially many calls to $\mathsf{Query}[\mathsf{Sign}\text{-}\mathcal{F}\ ]$ to the case of Claim 6 where the adversary makes at most one call to $\mathsf{Query}[\mathsf{Sign}\text{-}\mathcal{F}\ ]$. We can simulate the calls to $\mathsf{Query}[\mathsf{Sign}\text{-}\mathcal{F}\ ]$ both for $b = 0$ and for $b = 1$ using the oracle access to Sig and to $\mathsf{Eval}_{\mathcal{F}}$.

$\square$

References

[1] M. Abdalla, C. Namprempre, and G. Neven. On the (im)possibility of blind message authentication codes. In , *CT-RSA 2006*, volume 3860 of *LNCS*, pages 262–279, San Jose, CA, USA, Feb. 13–17, 2006. Springer, Berlin, Germany.

[2] 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.

[3] 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.

[4] 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.

[5] B. Barak and M. Mahmoody-Ghidary. Lower bounds on signatures from symmetric primitives. In *48th FOCS*, pages 680–688, Providence, USA, Oct. 20–23, 2007. IEEE Computer Society Press.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] E. Boyle, S. Goldwasser, and I. Ivan. Functional signatures and pseudorandom functions. Cryptology ePrint Archive, Report 2013/401, 2013. `http://eprint.iacr.org/`.

[15] 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.

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

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

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

[24] 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.

[25] 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.

[26] M. Fischlin. Round-optimal composable blind signatures in the common reference string model. In , *CRYPTO 2006*, volume 4117 of *LNCS*, pages 60–77, Santa Barbara, CA, USA, Aug. 20–24, 2006. Springer, Berlin, Germany.

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

[28] 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.

[29] 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, 1988.

[30] 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.

[31] 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.

[32] 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.

[33] A. Juels, M. Luby, and R. Ostrovsky. Security of blind digital signatures (extended abstract). In , *CRYPTO'97*, volume 1294 of *LNCS*, pages 150–164, Santa Barbara, CA, USA, Aug. 17–21, 1997. Springer, Berlin, Germany.

[34] J. Katz, D. Schröder, and A. Yerukhimovich. Impossibility of blind signatures from one-way permutations. In , *TCC 2011*, volume 6597 of *LNCS*, pages 615–629, Providence, RI, USA, Mar. 28–30, 2011. Springer, Berlin, Germany.

[35] J. Katz, D. Schröder, and A. Yerukhimovich. Impossibility of blind signatures from one-way permutations. In *Theory of Cryptography*, pages 615–629. Springer, 2011.

[36] 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.

[37] D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, 2000.

[38] 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.

[39] 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 Relation between DFS and FS

In an independent work, Boyle, Goldwasser, and Ivan [14] presented *functional signatures*, a concept that is closely related to a weaker variant of delegatable functional signatures In this section we investigate the relation of their work with delegatable functional signatures. First we show how to build functional signatures from delegatable functional signatures. Furthermore we argue why building delegatable functional signatures our of functional signatures alone seems hard.

A.1 Constructing functional signatures (from DFSS)

Given a DFS scheme $\mathsf{DFSS} = (\mathsf{Setup}, \mathsf{KGen}_{sig}, \mathsf{KGen}_{ev}, \mathsf{Sig}, \mathsf{Eval}_{\mathcal{F}}, \mathsf{Vf})$, we construct a functional signature scheme $\mathsf{FSS} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Sig}, \mathsf{Vf})$ satisfying both unforgeability and function privacy as defined in [14].

The setup algorithm generates two key-pairs of the DFS, where we use (sk_{sig}, pk_{sig}) for signing and (sk_{ev}, pk_{ev}) for evaluating functions on the messages. The signing key sk_f for a function f is modeled as a signature (on an empty message) that can be transformed into all messages in the range of f. This means that we consider the following functionality: $\mathcal{F}(\lambda, f, \alpha, pk_{ev}, m) := f(\alpha)$. We give a formal description of the algorithm in the following.

$\mathsf{FSS}.\mathbf{Setup}(\lambda)$:

$$\begin{aligned}
(pp, msk) &\leftarrow \mathsf{DFSS}.\mathsf{Setup}(\lambda)\\
(sk_{sig}, pk_{sig}) &\leftarrow \mathsf{DFSS}.\mathsf{KGen}_{sig}(pp, msk)\\
(sk_{ev}, pk_{ev}) &\leftarrow \mathsf{DFSS}.\mathsf{KGen}_{ev}(pp, msk)\\
\mathsf{FSS}.msk &:= (pp, sk_{sig}, sk_{ev}, pk_{sig}, pk_{ev})\\
\mathsf{FSS}.mvk &:= (pp, pk_{sig}, pk_{ev})\\
\mathit{output} &\ \ (\mathsf{FSS}.msk, \mathsf{FSS}.mvk)
\end{aligned}$$

$\mathsf{FSS}.\mathbf{KGen}(\mathsf{FSS}.\boldsymbol{msk}, f)$:

$$\begin{aligned}
\mathsf{FSS}.msk &= (pp, sk_{sig}, sk_{ev}, pk_{sig}, pk_{ev})\\
\sigma_f &\leftarrow \mathsf{DFSS}.\mathsf{Sig}(pp, sk_{sig}, pk_{ev}, f, \perp)\\
\mathsf{FSS}.sk_f &:= (pp, sk_{ev}, \sigma_f, pk_{sig})\\
\mathit{output} &\ \ \mathsf{FSS}.sk_f
\end{aligned}$$

$\mathsf{FSS}.\mathbf{Sig}(\mathsf{FSS}.sk_f, m)$:

```
$\mathsf{FSS}.sk_f = (pp, sk_{ev}, \sigma_f, pk_{sig})$
$\mathsf{FSS}.\sigma \leftarrow \mathsf{DFSS}.\mathsf{Eval}_{\mathcal{F}}(pp, sk_{ev}, pk_{sig}, m, \perp, \perp, \sigma_f)$
$output\ \ \mathsf{FSS}.\sigma$
```

$\mathsf{FSS}.\mathbf{Vf}(\mathsf{FSS}.mvk, m^*, \mathsf{FSS}.\sigma)$:

```
$\mathsf{FSS}.mvk = (pp, pk_{sig}, pk_{ev})$
$\mathsf{FSS}.b \leftarrow \mathsf{DFSS}.\mathsf{Vf}(pp, pk_{sig}, pk_{ev}, m^*, \mathsf{FSS}.\sigma)$
$output\ \ \mathsf{FSS}.b$
```

DFSS.Unfl implies FS.Unforgeability

The unforgeability notion of a functional signature scheme is closely related to our notion of unforgeability against insider attacks. Given an adversary $\mathcal{A}$ that is able to forge a signature for a message that is not in the range of any function f for which it has been given a key sk_f we build an adversary $\mathcal{B}$ against Unfl as follows. $\mathcal{B}$ simulates the algorithm from Section A.1, but uses Query[KGenS] to acquire a key sk_{ev} and Query[Sign] to acquire signatures σ_f for functions f whenever $\mathcal{A}$ asks for a key sk_f.

Eventually $\mathcal{A}$ produces a forgery (m^*, σ^*), i.e., σ^* is a signature for m^* and m^* is not in the range of any function f for which $\mathcal{A}$ has asked for a key sk_f. By construction m^* is not in the transitive closure $\mathcal{F}^*(\lambda, (f, m))$ for any message m that was signed for $\mathcal{B}$ together with functionality f and thus (m^*, σ^*) also constitutes a forgery for Unfl.

DFSS.CFA implies FS.FunctionPrivacy

Given an adversary $\mathcal{A}$ against FS.FunctionPrivacy, we construct an adversary $\mathcal{B}$ against DFSS.CFA. $\mathcal{B}$ translates the queries as follows. First, $\mathcal{B}$ uses Query[KGenS] to acquire a key sk_{ev} and initializes $\mathcal{A}$. Eventually, $\mathcal{A}$ outputs a tuple $m_0.m_1, f_0, f_1$ s.t. $f_0(m_0) = f_1(m_1)$. Now $\mathcal{B}$ flips a coin $b_{\mathcal{B}} \leftarrow \{0, 1\}$ and sends Query[Sign-$\mathcal{F}$]$(f_b, (pk_{ev}, \perp), (pk_{ev\perp}, m_b), 1, m_\perp)$ to its challenger for DFSS.CFA. It forwards the signature that the challenger sends to $\mathcal{A}$. This signature now either is constructed exactly as in the game of FS.FunctionPrivacy (if the signature has been generated via $\mathsf{Eval}_{\mathcal{F}}$) or a fresh signature on $m' = f_0(m_0) = f_1(m_1)$. In the latter case, $\mathcal{A}$ can only guess the bit $b_{\mathcal{B}}$ with probability exactly $\frac{1}{2}$. In the first case, however, by assumption $\mathcal{A}$ guesses the bit with probability noticeably higher than $\frac{1}{2}$. Thus, If $\mathcal{A}$ guesses the bit b correctly, $\mathcal{B}$ outputs 1, otherwise it outputs 0. This way the (noticeable) advantage of $\mathcal{A}$ translates to an advantage of $\mathcal{B}$ that is at least half the advantage of $\mathcal{A}$ and thus still noticeable.

A.2 Constructing delegatable functional signatures (from FS)

A DFS scheme provides one significant property, which a functional signature scheme does (trivially) not provide: delegation. Without delegation, i.e., if we disallow an evaluator to delegate its capabilities, delegatable functional signature schemes are closely related to functional signature schemes. One could encode the functionality $\mathcal{F}$ together with an identity ID of an evaluator into the functions $f_{\mathcal{F},\mathsf{ID}}$ for which keys sk_f are generated. The (original) signer then signs the message together with capabilities $\mathsf{DFSS}.f$ and the identity of the evaluator. The main idea lies in defining

$f_{\mathcal{F},\mathsf{ID}}$ such that it generates a new message m' only on input of all the relevant values: A signature on $(m, \mathsf{DFSS}.f, \mathsf{ID})$ and the value α. If furthermore the functional signature scheme has the property that whenever an adversary can create a signature on $f' = f(x)$, there is an adversary that also can output x, we can prove both unforgeability and privacy against chosen function attack both against insider adversaries. Obtaining delegation, however, appears to be a more complicated task and we currently do not see how to achieve it. In the following section we show that DFS imply several seemingly different signature primitives. Using only black-box access to a given delegatable functional signature scheme $\mathsf{DFSS} = (\mathsf{Setup}, \mathsf{KGen}_{sig}, \mathsf{KGen}_{ev}, \mathsf{Sig}, \mathsf{Eval}_{\mathcal{F}}, \mathsf{Vf})$, we construct (among others) identity based signature schemes, sanitizable signature, and redactable signature schemes.
Signature Schemes. As a warm-up, we show that DFS 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 messages of its choice.

We use the following, straight-forward functionality:

$$\mathcal{F}_S(\lambda, f, \alpha, pk_{ev}, m) := (\perp, \perp)$$

Proposition 9 (Signatures)**.** *If unforgeable delegatable functional signatures exist, then unforgeable signatures exist.*

Unforgeability against outsider adversaries (UnfO) suffices for regular signatures.

Identity Based Signatures. In an identity based signature scheme (IBS), publicly known identifiers ID serve as verification keys for signatures that were generated by the entity with identifier ID. We follow the general construction of [10], in which a trusted party generates a master secret key msk and extracts a signing key sk_{ID} from (msk, ID) for each entity with identifier ID. Formally, an IBS scheme is a tuple $\mathsf{IBS} = (\mathsf{Setup}_{\mathsf{IBS}}, \mathsf{Extract}_{\mathsf{IBS}}, \mathsf{Sig}_{\mathsf{IBS}}, \mathsf{Vf}_{\mathsf{IBS}})$ of efficient algorithms.

```
$\mathsf{Setup}_{\mathsf{IBS}}(\lambda)$ :
$(pp, msk) \leftarrow \mathsf{Setup}(\lambda)$
$(sk_{sig}, pk_{sig}) \leftarrow \mathsf{KGen}_{sig}(pp, msk)$
$(sk_{ev}, pk_{ev}) \leftarrow \mathsf{KGen}_{ev}(pp, msk)$
set $\mathsf{IBS}.pp := (pp, pk_{sig}, pk_{ev})$
set $\mathsf{IBS}.msk := (msk, sk_{sig}, sk_{ev})$
output $(\mathsf{IBS}.msk, \mathsf{IBS}.pp)$
```

```
$\mathsf{Extract}_{\mathsf{IBS}}(pp, msk, \mathsf{ID})$ :
parse $\mathsf{IBS}.pp = (pp_{\mathcal{F}}, pk_{sig}, pk_{ev})$
parse $\mathsf{IBS}.msk = (msk_{\mathcal{F}}, sk_{sig}, sk_{ev})$
$\sigma_{\mathsf{ID}} \leftarrow \mathsf{Sig}(pp, sk_{sig},$
        $pk_{ev}, 1, (\mathsf{ID}, \perp))$
set $sk_{\mathsf{ID}} := (sk_{ev}, \sigma_{\mathsf{ID}}, \mathsf{ID})$
output $sk_{\mathsf{ID}}$
```

```
$\mathsf{Sig}_{\mathsf{IBS}}(pp, sk_{\mathsf{ID}}, m)$ :
parse $\mathsf{IBS}.pp = (pp, pk_{sig}, pk_{ev})$
parse $sk_{\mathsf{ID}} = (sk_{ev}, \sigma_{\mathsf{ID}}, \mathsf{ID})$
$\sigma \leftarrow \mathsf{Eval}_{\mathcal{F}}(pp_{\mathcal{F}}, sk_{ev}, pk_{sig},$
        $m, (\mathsf{ID}, \perp), \sigma_{\mathsf{ID}})$
output $\sigma$
$\mathsf{Vf}_{\mathsf{IBS}}(pp, \mathsf{ID}, m, \sigma)$ :
parse $\mathsf{IBS}.pp = (pp, pk_{sig}, pk_{ev})$
$b \leftarrow \mathsf{Vf}(pp, pk_{sig}, pk_{ev}, (\mathsf{ID}, m), \sigma)$
output $b$
```

Figure 8: Construction of an IBS out of a DFSS

For our implication we derive a secret signing key sk_{sig} that is never given to any entity except for the trusted party. With this signing key, we generate secret keys by signing the individual entity's ID. This signature σ_{ID} constitutes the secret that a participant can use in order to sign their own messages. Signing means modifying σ_{ID} in a controlled way: We add the (real) message to the (signed) ID. (see Figure 8).

We define the functionality $\mathcal{F}_{\mathsf{IBS}}$ as follows.

$$\mathcal{F}_{\mathsf{IBS}}(\lambda, 1, \alpha, pk_{ev}, (\mathsf{ID}, \perp)) := (0, (\mathsf{ID}, \alpha))$$

Naturally, $\mathcal{F}_{\mathsf{IBS}}(\lambda, f, \cdot, \cdot, \cdot) := \perp$ for all $f \neq 1$. The security of the scheme comes from the fact that σ_{ID} cannot be generated by any entity but the trusted party.

Proposition 10 (Identity based signatures)**.** *If unforgeable private delegatable functional signatures exist, then unforgeable identity based signatures exist.*

Unforgeability against insider adversaries (UnfI) and privacy against insider adversaries is necessary for this construction. The fact that privacy is necessary might be somewhat surprising at first glance. However, since we use signatures as keys, the original signature σ_{ID} has to be kept secret. If it can be extracted from a derived signature, other participants can sign in the name of ID.

Sanitizable Signatures. Sanitizable signature schemes as in [18] allow a signature on m to be enriched with certain admissions adm. All possible modifications mod such that $\mathsf{adm}(\mathsf{mod}) = 1$ can be applied to the signature to derive signatures on $m' = \mathsf{mod}(m)$. The sanitizable signature scheme we create does not have a notion of accountability. However, extending the scheme to an accountable sanitizable signature scheme is an interesting future work. We construct a functionality $\mathcal{F}$ as follows, where $\mathsf{adm}_\perp$ always returns 0.

$$\mathcal{F}(\lambda, \mathsf{adm}, \mathsf{mod}, pk_{ev}, m) := \begin{cases} (\mathsf{adm}_\perp, \mathsf{mod}(m)) \text{ if } \mathsf{adm}(\mathsf{mod}) = 1 \\ \perp \text{ otherwise} \end{cases}$$

Proposition 11. *If unforgeable, private delegatable signature schemes exist, then unforgeable, immutable, private sanitizable signature schemes exist.*

Redactable Signatures. Redactable signatures as in [17] allow signatures that can be modified in specific ways by anyone that holds access to the signature. Intuitively we can construct a redactable signature scheme RSS from a functional signature scheme by making the secret evaluator key sk_{ev} public. This allows everyone to modify signatures according to a functionality $\mathcal{F}$ that implements the desired redaction predicate P as follows.

$$\mathcal{F}_P(\lambda, f, \alpha, pk_{ev}, m) := \begin{cases} (0, \alpha) \text{ if } P(m, \alpha) = 1 \wedge f = 1 \\ \perp \text{ otherwise} \end{cases}$$

Proposition 12 (Redactable signatures)**.** *If unforgeable, private delegatable signatures exist then unforgeable private redactable signatures exist.*

Blind signatures. Delegatable functional signatures also imply blind signatures. We explore this subject in detail in Section 4.1.

B Blind signatures

We follow the notation and the definitions from [35]. Blind signature schemes are, in contrast to traditional signature schemes, interactive. Thus we define interactive executions of two algorithms

X and Y as $(x, y) \leftarrow \langle \mathsf{X}(a), \mathsf{Y}(b) \rangle$, where x is the output of X and y is the output of Y. These outputs are not necessarily public. We write $\mathsf{X}^{\langle O(a),\cdot \rangle}$ if X has access to an interactive oracle O with input a. In such an oracle interaction, X does not receive the private output of O and vice versa. If X may only interact with O once, we write $\mathsf{X}^{\langle O(a),\cdot \rangle^1}$. Note that if X has access to several oracles, it does not have to invoke all of them and might invoke them in an arbitrary order.

Definition 18 (Interactive signature scheme)**.** *We define an interactive signature scheme as a tuple of efficient algorithms* $\mathsf{BS} = (\mathsf{KG}_{\mathsf{BS}}, \langle \mathcal{S}, \mathcal{U} \rangle, \mathsf{Vf}_{\mathsf{BS}})$*, consisting of the key-generation algorithm* $\mathsf{KG}_{\mathsf{BS}}$*, the signer* $\mathcal{S}$*, the user* $\mathcal{U}$*, and the verification algorithm* $\mathsf{Vf}_{\mathsf{BS}}$*, where*

$\mathsf{KG}_{\mathsf{BS}}(\lambda)$**:** *The key generation for parameter* λ *generates a key pair* $(sk_{\mathsf{BS}}, pk_{\mathsf{BS}})$.

$\langle \mathcal{S}, \mathcal{U} \rangle$**:** *The signature issuing, consisting of the execution of algorithm* $\mathcal{S}(sk_{\mathsf{BS}})$ *and algorithm* $\mathcal{U}(pk_{\mathsf{BS}}, m)$ *for message* $m \in \{0,1\}^*$ *generates an output* σ *of the user, and some possibly empty output out for the signer (such as a status message like* ok *or* $\perp$*),* $(out, \sigma) \leftarrow \langle \mathcal{S}(sk_{\mathsf{BS}}), \mathcal{U}(pk_{\mathsf{BS}}, m) \rangle$.

$\mathsf{Vf}_{\mathsf{BS}}(\boldsymbol{pk}_{\mathsf{BS}}, m, \sigma)$**:** *The verification algorithm* $\mathsf{Vf}_{\mathsf{BS}}$ *outputs a bit* $b \in \{0,1\}$.

We call an interactive signature scheme complete*, if for any function* f*, with overwhelming probability in* $\lambda \in \mathbb{N}$ *the following holds: when executing* $(sk_{\mathsf{BS}}, pk_{\mathsf{BS}}) \leftarrow \mathsf{KG}_{\mathsf{BS}}(\lambda)$*, setting* $m := f(\lambda, pk_{\mathsf{BS}}, sk_{\mathsf{BS}})$*, and letting* σ *be the output of* $\mathcal{U}$ *in the joint execution of* $\mathcal{S}(sk_{\mathsf{BS}})$ *and* $\mathcal{U}(pk_{\mathsf{BS}}, m)$*, then we have* $\mathsf{Vf}_{\mathsf{BS}}(pk_{\mathsf{BS}}, m, \sigma) = 1$.

B.1 Basic security notions for blind signatures

Security of blind signature schemes is defined by unforgeability and blindness [33, 37].

Unforgeability. An adversary $\mathcal{U}^*$ against unforgeability tries to generate $k+1$ valid message/signatures pairs with different messages after at most k completed interactions with the honest signer, where the number of executions is adaptively determined by $\mathcal{U}^*$ during the attack. To identify completed sessions we assume that the honest signer returns a special symbol ok when having sent the final protocol message in order to indicate a completed execution (from its point of view). We remark that this output is "atomically" connected to the final transmission to the user.

Definition 19 (Unforgeability)**.** *An interactive signature scheme* $\mathsf{BS} = (\mathsf{KG}_{\mathsf{BS}}, \langle \mathcal{S}, \mathcal{U} \rangle, \mathsf{Vf}_{\mathsf{BS}})$ *is called* unforgeable *if for any efficient algorithm* $\mathcal{A}$ *(the malicious user) the probability that experiment* $\mathsf{Unforge}_{\mathcal{A}}^{\mathsf{BS}}(\lambda)$ *evaluates to 1 is negligible (as a function of* λ*) where*

```
***Experiment*** $\mathsf{Unforge}_{\mathcal{A}}^{\mathsf{BS}}(\lambda)$
    $(sk_{\mathsf{BS}}, pk_{\mathsf{BS}}) \leftarrow \mathsf{KG}_{\mathsf{BS}}(\lambda)$
    $((m_1^*, \sigma_1^*), \ldots, (m_{k+1}^*, \sigma_{k+1}^*)) \leftarrow \mathcal{A}^{\langle \mathcal{S}(sk_{\mathsf{BS}}),\cdot \rangle^\infty}(pk_{\mathsf{BS}})$
    Return 1 iff
        $m_i^* \neq m_j^*$ for all $i, j$ with $i \neq j$, and
        $\mathsf{Vf}_{\mathsf{BS}}(pk_{\mathsf{BS}}, m_i^*, \sigma_i^*) = 1$ for all $i$, and
        $\mathcal{S}$ has returned ok in at most $k$ interactions.
```

Blindness. The blindness condition says that it should be infeasible for a malicious signer $\mathcal{S}^*$ to decide which of two messages m_0 and m_1 has been signed first in two executions with an honest user $\mathcal{U}$. The traditional definition considers maliciously generated keys by the $\mathcal{S}^*$ [1, 26]. For our purpose, however, it is sufficient to consider honestly generated keys. If one of these executions has returned $\perp$ then the signer is not informed about the other signature (Otherwise the signer could trivially identify one session by making the other abort.).

Definition 20 (Blindness). *A blind signature scheme* $\mathsf{BS} = (\mathsf{KG}_{\mathsf{BS}}, \langle \mathcal{S},\mathcal{U} \rangle, \mathsf{Vf}_{\mathsf{BS}})$ *is called* blind *if for any efficient algorithm* $\mathcal{S}^*$ *(working in modes* find, issue *and* guess*) the probability that the following experiment* $\mathsf{Blind}^{\mathsf{BS}}_{\mathcal{S}^*}(\lambda)$ *evaluates to* 1 *is negligibly bigger than* 1/2*, where*

```
***Experiment*** $\mathsf{Blind}^{\mathsf{BS}}_{\mathcal{S}^*}(\lambda)$
    $(sk_{\mathsf{BS}}, pk_{\mathsf{BS}}) \leftarrow \mathsf{KG}_{\mathsf{BS}}(\lambda)$
    $(m_0, m_1, \mathit{st}_{\mathit{find}}) \leftarrow \mathcal{S}^*(\mathsf{find}, sk_{\mathsf{BS}}, pk_{\mathsf{BS}})$
    $b \leftarrow \{0,1\}$
    $\mathit{st}_{\mathit{issue}} \leftarrow \mathcal{S}^{*\langle \cdot, \mathcal{U}(pk_{\mathsf{BS}}, m_b)\rangle^1, \langle \cdot, \mathcal{U}(pk_{\mathsf{BS}}, m_{1-b})\rangle^1}(\mathit{issue}, \mathit{st}_{\mathit{find}})$
        and let $\sigma_b, \sigma_{1-b}$ denote the (possibly undefined) local outputs
        of $\mathcal{U}(pk_{\mathsf{BS}}, m_b)$ resp. $\mathcal{U}(pk_{\mathsf{BS}}, m_{1-b})$.
    set $(\sigma_0, \sigma_1) = (\perp, \perp)$ if $\sigma_0 = \perp$ or $\sigma_1 = \perp$
    $b^* \leftarrow \mathcal{S}^*(\mathsf{guess}, \sigma_0, \sigma_1, \mathit{st}_{\mathit{issue}})$
    return 1 iff $b = b^*$.
```