Delegatable Functional Signatures

Michael Backes*†, Sebastian Meiser*, and Dominique Schröder*

* CISPA, Saarland University
† MPI-SWS

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. Technically, DFS unify several seemingly different signature primitives, including functional signatures and policy-based signatures (PKC'14), sanitizable signatures, identity based signatures, and blind signatures.
We characterize the instantiability of DFS with respect to the corresponding security notions of unforgeability and privacy. On the positive side we show that privacy-free DFS can be constructed from one-way functions. Furthermore, we show that unforgeable 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.

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 [31]. 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 and a controlled form of malleability would be desirable. Consider as an example a company S that wishes to outsource the computation on authenticated data to untrusted parties (resp. to parties that may further delegate the computation to sub-contractors), called the evaluators, without handing out any secret key. For each data, S chooses a set of allowed functions $\mathcal{F}$ that can be applied. The evaluators are organized hierarchically, where each evaluator receives an intermediate result and can compute

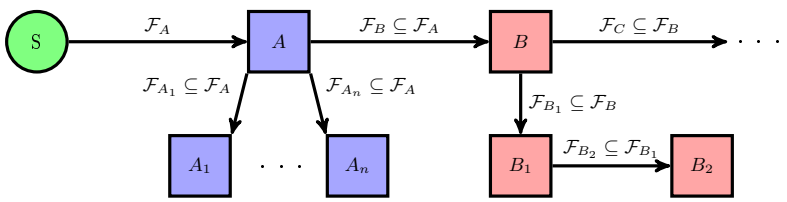

Fig. 1. Example hierarchical application of DFS

any function $f \in \mathcal{F}$ chosen by S, or delegate subsets $\mathcal{F}' \subseteq \mathcal{F}$ to other evaluators. We allow S to restrict the number of delegations, as well as the order in which functions can be applied. The chain computation should be publicly verifiable which means that everybody can verify that:

- The computation was based on the original data of S.
- Only the functions chosen by S were applied to the data (in the right order, if specified).
- Any delegation of computation by an evaluator A to another evaluator (a third party B or any sub-contractor A_i) has been authorized by S and A.

We refer to Figure 1 for an example structure of delegation. We consider the following security notions: *Unforgeability* says that malicious evaluators can only apply the functions(s) they were allowed to apply, e.g.: B_1 can only apply $\mathcal{F}_{B_1} \subseteq \mathcal{F}_B \subseteq \mathcal{F}_A$ and further delegate sets of functions $\mathcal{F}_{B_2} \subseteq \mathcal{F}_{B_1}$. *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): To an observer, the signature σ_{B_2} computed by B_2 for a message m_{B_2} is indistinguisable from a signature σ for m_{B_2} computed by S. Privacy is a useful and desirable property in many applications as it hides the business structure of the (sub-)contractor and still allows to verify the correctness of 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.

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 1 (Malleability). Suppose that a sender S wants to sign a document, but allow another entity A to fill in information in a few fields. S chooses f to describe the places where information can be added, as well as which information can be added (e.g., 16 characters) without harming the validity of the signature. A chooses the fields and the information it wishes to fill in by choosing the corresponding value for α, and he derives a signature on m', where $(\cdot, m') \leftarrow \mathcal{F}(f, \alpha, k, m)$.

Example 2 (Delegatability). Suppose that a sender S wants to restrict how A 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, but no further information can be added.[1] After filling in information, A may delegate the censoring to B, but impose restrictions on which part of the document may be censored by choosing the corresponding value for α. Then, A and delegates the corresponding capabilities f' to B, where $(f', \cdot) \leftarrow \mathcal{F}(f, \alpha, k_B, m)$.

Our definition also covers signing capabilities for fresh messages. If a sender S wants to give A the capability to sign certain messages in his name, he can simply generate a signature σ_{fresh} for a new (empty) message and use f to specify which capabilities A 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:

[1] If a PKI exists, S can additionally add descriptions of the evaluators that are allowed to do this second round of processing.

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 [15], policy-based signatures, which were recently introduced by Bellare and Fuchsbauer [11], 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., [16,17,21,22,10,23,20,28]. The main difference between delegatable anonymous credential schemes, such as [9,1], and our approach is that delegation is done by extending the proof chain (and thus leaking information about the chain). Restricting the properties of the issuer in a credential system has been considered in [8]. 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 (we refer to [24] for a nice overview). We group malleable signature schemes into three categories: publicly modifiable signatures, sanitizable signatures and proxy signatures. Publicly modifiable signatures 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 [38,33,37,18] 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 [33], linearly homomorphic signature schemes [29,14,5] and schemes that allow for evaluating polynomial functions [13,25]. Libert et al. combine linearly homomorphic signatures with structure preserving signatures [36]. However, known publicly modifiable signature schemes only consider static functions or predicates (one function or predicate for every scheme) and leave the signer little room for bounding a class of functions to a specific message. As the signatures can be

modified by everyone with access to public information, they do not allow for a concept of controlled delegation.

Sanitizable signature schemes [3,19] 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 [30] 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 was investigated by Chase et al. [27]. The main contribution is an efficient scheme based on malleable zero-knowledge proofs [26] and the question regarding the minimal assumptions was left open.

1.3 Closely Related Work

The general framework by Ahn et al. [2] 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). Attrapadung et al. [4] discuss and refine this framework. Both works do not explore the minimal computational assumptions.

The works of Boyle, Goldwasser, and Ivan [15] (introducing *functional digital signatures*) and of Bellare and Fuchsbauer [11] (introducing policy-based signatures) are closely related to our notion of DFS.

In a functional signature scheme, 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 to both works, 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 functional signatures [15], as well as for policy-based signatures [11], because our impossibility result does not rely on the delegation property of our scheme. Furthermore, DFS signatures allow for authenticated chain computations. In the extended version [6] we compare delegatable functional signature schemes to functional digital signature schemes and policy-based signature schemes and show how to construct them out of a delegatable functional signature scheme. Whether the converse is possible is unknown.

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:*

$(\boldsymbol{pp}, \boldsymbol{msk}) \leftarrow \mathbf{Setup}(\lambda)$**:** *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. We refer to the extended edition [6] for a formal definition of our correctness.

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 $G(\mathsf{DFSS}, \mathcal{F}, \mathcal{A}, \lambda)$ [12]. 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 [32], 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 that can only access $\mathsf{Query[KGenP]}$ to retrieve public keys is considered an *outsider*; an adversary that can access the oracle $\mathsf{Query[KGenS]}$ to retrieve one or more secret evaluator keys is considered an *insider*; an adversary that additionally can access the oracle $\mathsf{Query[RegKey]}$ to (adaptively) register its own (possibly malicious) evaluator keys is considered a *strong insider* (*S-Insider*). All adversaries have access to the honestly generated public signer key pk_{sig}. We keep track of the following sets: $\mathcal{K}_{\mathcal{C}}$ stores all key pairs, $K_{\mathcal{A}}$ stores all public keys for which the adversaries knows the private key, and Q stores $\mathcal{A}$'s queries to both $\mathsf{Query[Sign]}$ and $\mathsf{Query[Eval]}$. To handle the information that an adversary can trivially deduce from its queries, we define the transitive closure for functionalities.

Definition 2. *(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 $\mathsf{Check}{-}\mathcal{F}$ such that $\mathsf{Check}{-}\mathcal{F}(\lambda, f, m, m^*) = 1$ iff $m \in \mathcal{F}^*(\lambda, (f, m))$.

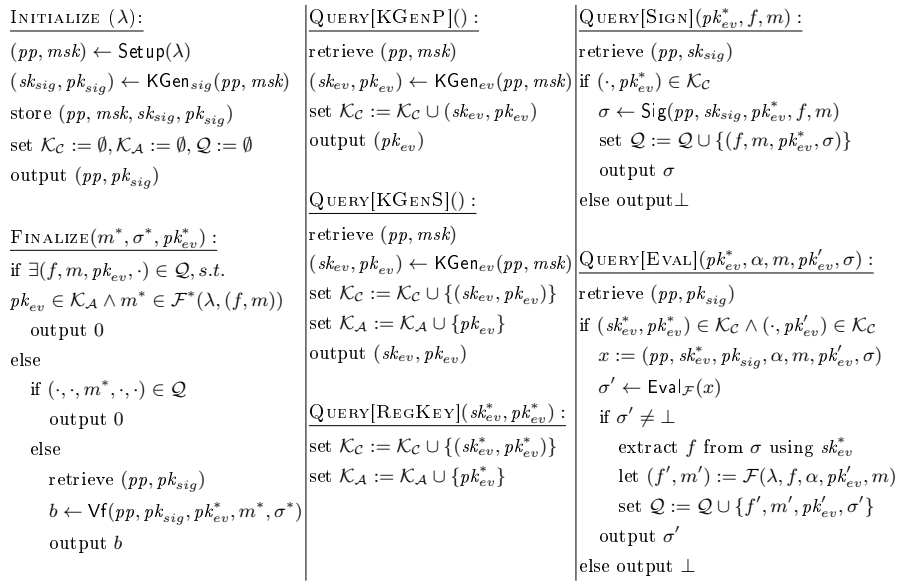

Fig. 2. Unforgeability for delegatable functional signature schemes.

Definition 3. *(Unforgeability Against $X \in \{$Outsider, Insider, S-Insider$\}$ Attacks). Let* $\mathsf{DFSS} = (\mathsf{Setup}, \mathsf{KGen}_{sig}, \mathsf{KGen}_{ev}, \mathsf{Sig}, \mathsf{Eval}_\mathcal{F}, \mathsf{Vf})$ *be a delegatable functional signature scheme. The definition uses the game* $\mathsf{Unf}(\mathsf{DFSS}, \mathcal{F}, \mathcal{A}, \lambda)$ *defined in Figure 2. 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. Within the oracles Query[Sign] and Query[Eval], the challenger only allows to delegate to known keys $k \in \mathcal{K}_\mathcal{C}$. Note that his does not restrict the adversary, but allows the challenger to distinguish between weak insider and strong insider. All messages m signed either by Query[Sign] or Query[Eval] are added to $\mathcal{Q}$, together with the

respective function f and the public key of the evaluator to whom the message was delegated.

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 $(\cdot, m^*, \cdot, \cdot) \neq \mathcal{Q}$. Moreover, for insider adversaries, the forgery must not be trivially deducible from previously issued signatures $(f, m, {pk_{ev}}^*, \sigma)$ for keys ${pk_{ev}}^* \in \mathcal{K}_{\mathcal{A}}$. 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 that leaks the secret signing key sk_{sig} with every signature, and that has only one valid public evaluator key pk_{ev}, that allows an insider with the respective secret key sk_{ev} to change messages inside signatures to arbitrary values. An insider that received sk_{ev} 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 $\mathsf{Eval}_{\mathcal{F}}$. Without invoking $\mathsf{Query}[\mathsf{KGenS}]$, 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 (EU-X-A-Implications). *Let* DFSS *be a functional signature scheme.*

(i) *For all PPT adversaries $\mathcal{A}_{Outsider}$ there exists a PPT adversary $\mathcal{A}_{Insider}$ s.t.* $\mathbf{Adv}^{\mathsf{EU\text{-}IA}}_{\mathsf{DFSS},\mathcal{F},\mathcal{A}_{Insider}} \geq \mathbf{Adv}^{\mathsf{EU\text{-}OA}}_{\mathsf{DFSS},\mathcal{F},\mathcal{A}_{Outsider}}$

(ii) *For all PPT adversaries $\mathcal{A}_{Insider}$ there exists a PPT adversary $\mathcal{A}_{S\text{-}Insider}$ s.t.* $\mathbf{Adv}^{\mathsf{EU\text{-}SIA}}_{\mathsf{DFSS},\mathcal{F},\mathcal{A}_{S\text{-}Insider}} \geq \mathbf{Adv}^{\mathsf{EU\text{-}IA}}_{\mathsf{DFSS},\mathcal{F},\mathcal{A}_{Insider}}$

Proof. The proposition follows trivially. For (i), the adversary $\mathcal{A}_{\mathrm{Insider}}$ runs a black-box simulation of $\mathcal{A}_{\mathrm{Outsider}}$ and makes no use of the additional oracle. For (ii) the proposition follows analogously.

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 3 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 $\mathsf{Query}[\mathsf{Sign}\text{-}\mathcal{F}]$.

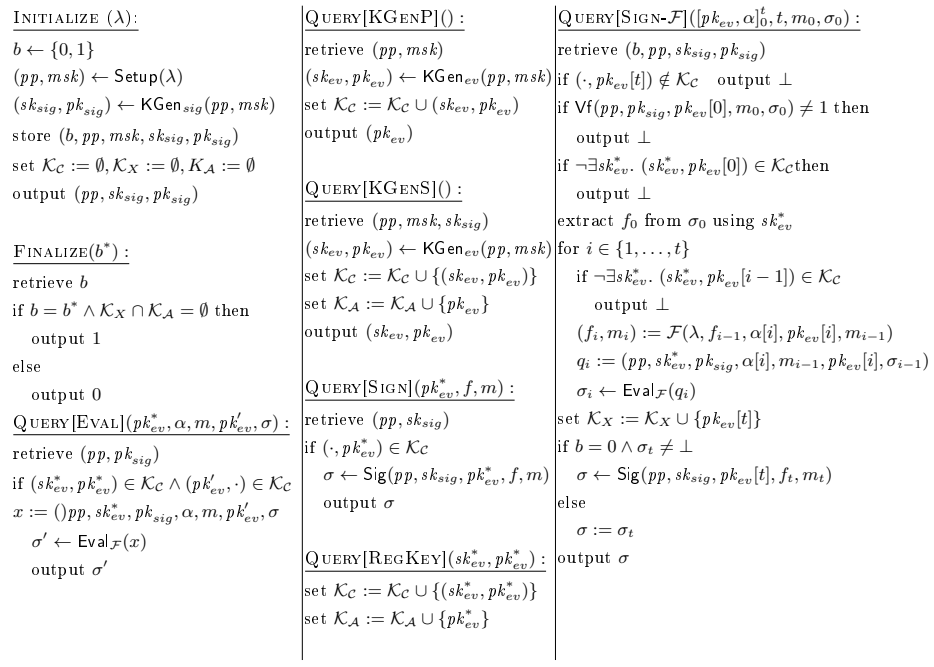

Fig. 3. Privacy under chosen functionality attacks CFA for delegatable functional signature schemes.

Definition 4. *(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 3. 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 signature σ_0 that allows the capability f_0 (technically the adversary can invoke PROC QUERY[SIGN] for computing σ_0), and a list of t public keys of evaluators together with evaluator inputs α. The chosen keys must be known to the challenger to distinguishing between outsiders, insiders and strong insiders. The challenger repeatedly applies $\mathsf{Eval}_{\mathcal{F}}$ to σ_0, using the specified parameters α_i keys $pk_{ev}[i]$. Additionally, $\mathcal{C}$ computes the derived valued m_i and f_i for the resulting signature via direct application of $\mathcal{F}$. If all transformations succeed,[2] $\mathcal{C}$ yields a signature σ_t, on a message m_t delegated to $pk_{ev}[t]$ with capability f_t. Depending on the bit b, $\mathcal{C}$ either sends σ_t or a fresh signature on m_t delegated to $pk_{ev}[t]$ with capability f_t to $\mathcal{A}$ and adds the key $pk_{ev}[t]$ to the set $\mathcal{K}_X$ that contains all keys to which the signaturews in challenges were (finally) delegated. If at the end of the game $\mathcal{K}_{\mathcal{A}} \cap \mathcal{K}_X \neq \emptyset$, the challenger outputs 0. This way we allow a scheme to leak some information to the evaluator to which a signature is delegated. For security against insider attacks only "local" information is allowed. After one delegation, this information has to vanish, since an insider adversary $\mathcal{A}$ can delegate the signature σ_t to a key $pk^*_{ev} \in \mathcal{K}_{\mathcal{A}}$ by using the Query[Eval] oracle.

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 $\mathcal{K}_X \cap \mathcal{K}_{\mathcal{A}} \neq \emptyset$).

Analogously to the hierarchy for unforgeability, an S-Insider is stronger an insider or an outsider. A scheme can leak information about delegation if a

[2] If one of the transformations failed Query[Sign-$\mathcal{F}$] outputs $\perp$ independently from the value of b, as we only want to give guarantees for valid signatures and not extend the notion of correctness.

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 analogously to the proof for Proposition 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 4 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 [15]. 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, an evaluator 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 3) but none of our privacy notions. Thus, we obtain the following simple result:

Theorem 1. *If one-way functions exist, then there exists an 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.

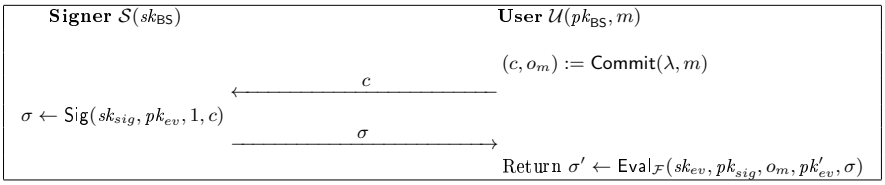

Fig. 4. Issue protocol of the two move blind signature scheme.

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 the extended edition [6] for formal definitions of commitment schemes and 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.[3]

$(\boldsymbol{sk}_{\mathsf{BS}}, \boldsymbol{pk}_{\mathsf{BS}}) \leftarrow \mathsf{KG}_{\mathsf{BS}}(1^\lambda)$**.** 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').

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

[3] $\mathcal{F}_C$ 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.*

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 3. *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 Unfl 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

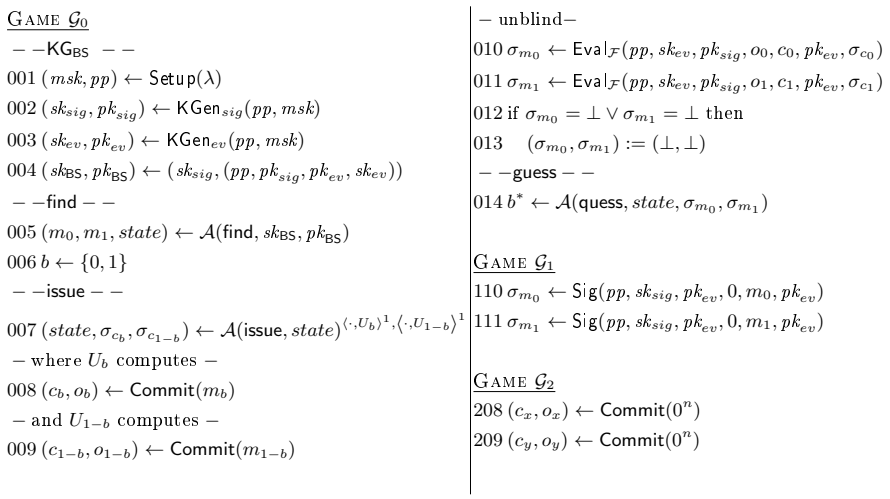

Fig. 5. The games for our proof for Proposition 4

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 messages) after one successful interaction it breaks the unforgeability of BS.

Proposition 4. *If* $\mathsf{DFSS} = (\mathit{Setup}, \mathit{KGen}_{sig}, \mathit{KGen}_{ev}, \mathit{Sig}, \mathit{Eval}_{\mathcal{F}}, \mathit{Vf})$ *is a private delegatable signature scheme and* $\mathsf{C} = (\mathsf{Commit}, \mathsf{Open})$ *is a commitment scheme which is 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 blind.*

Proof. We show this proposition via a game-based proof. We start with the original game $\mathsf{Blind}^{\mathsf{BS}}_{\mathcal{S}^*}(\lambda)$ for blindness 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.

In the following proof, as well as in subsequent proofs, we assign a number to each line where the first digit marks the game and the remaining digits the line in this game (e.g., 234 marks the 34^{th} line of game 2). All lines that are not explicitly stated are as they were defined in the last game that defined them. We refer to Figure 5 for a description of the games.

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. 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 respectively queries $\mathsf{Query}[\mathsf{Sign}\text{-}\mathcal{F}\,]((pk_{ev}, \perp), (pk_{ev}, o_0), 1, c_0, \sigma_{c_0})$ and $\mathsf{Query}[\mathsf{Sign}\text{-}\mathcal{F}\,]((pk_{ev}, \perp), (pk_{ev}, o_1), 1, c_1, \sigma_{c_1})$.

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. 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 [7]), 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 [15]. 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 Bounded DFS From Trapdoor Permutations

In this section we construct a bounded delegatable functional signature scheme DFSS as defined in Section 2.1, where we put an *a-priori* bound on the number of evaluations. 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. Formal definition of the underlying primitives can be found in the extended edition of this paper [6].

5.1 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

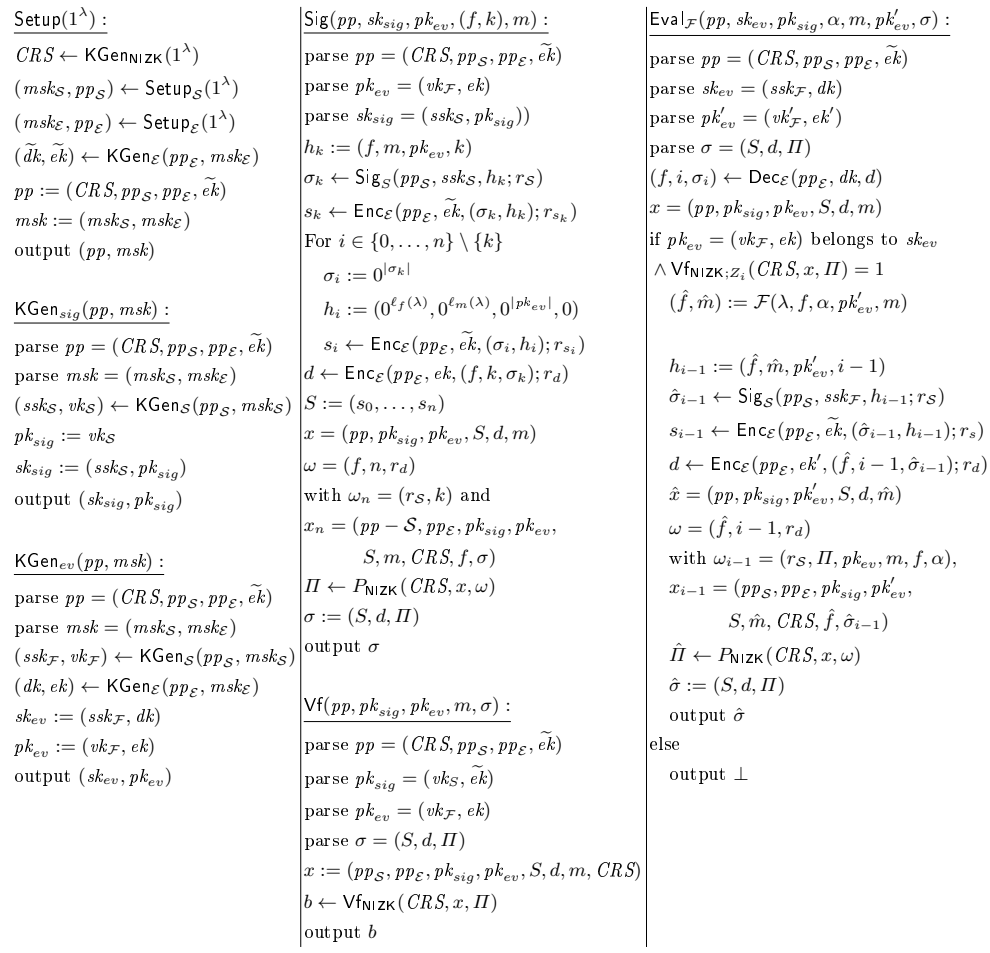

Fig. 6. Construction of a DFSS

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 4 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.

Given a signature scheme $S = (\mathsf{Setup}_{\mathcal{S}}, \mathsf{KGen}_{\mathcal{S}}, \mathsf{Sig}_{\mathcal{S}}, \mathsf{Vf}_{\mathcal{S}})$ with a simple key generation algorithm and with signatures of equal length, an encryption scheme $\mathcal{E} = (\mathsf{Setup}_{\mathcal{E}}, \mathsf{KGen}_{\mathcal{E}}, \mathsf{Enc}_{\mathcal{E}}, \mathsf{Dec}_{\mathcal{E}})$ and a zero-knowledge scheme $\mathsf{NIZK} = (\mathsf{KGen}_{\mathsf{NIZK}}, P_{\mathsf{NIZK}}, \mathsf{Vf}_{\mathsf{NIZK}})$ for languages in NP we construct a delegatable functional signature scheme DFSS as follows: We define a recursive class of languages L_i, where $L_n : \ x_n = (pp_{\mathcal{S}}, pp_{\mathcal{E}}, pk_{sig}, pk_{ev}, S, m, CRS, f, \sigma) \in L_n$ means that there exists a witness $\omega_n = (r, k)$ 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 $0 \leq i < n : \ x_i = (pp_{\mathcal{S}}, pp_{\mathcal{E}}, pk_{sig}, pk_{ev}, S, m, CRS, f, \sigma) \in L_i$ if there exists a witness $\omega_i = (r, \Pi, pk'_{ev}, m', f', \alpha)$ s.t. $pk_{sig} = (vk_{\mathcal{S}}, \widetilde{ek})$ and

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

The signer proves that $x = (pp = (CRS, pp_{\mathcal{S}}, pp_{\mathcal{E}}), pk_{sig}, \ pk_{ev} = (vk_{\mathcal{F}}, ek), S, d, m) \in L$, where L contains tuples for which there exists a witness $\omega = (f, i, r_d)$ such that $\mathsf{Vf}_{\mathsf{NIZK}_i}(CRS, (pp_{\mathcal{S}}, pp_{\mathcal{E}}, pk_{sig}, pk_{ev}, S, m, CRS, f, \sigma)) = 1$
$\wedge \ d \leftarrow \mathsf{Enc}_{\mathcal{E}}(pp_{\mathcal{E}}, ek, (f, i, \sigma); r_d)$.

5.2 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 that is zero knowledge, 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 1 and Lemma 2.

Lemma 1. *If $\mathcal{E}$ is a public key encryption scheme, $\mathcal{S}$ a length preserving unforgeable signature scheme with a simple key generation (i.e., not requiring a master secret key), and NIZK is a sound non-interactive proof scheme, then the construction* DFSS *presented in Section 5 is* unforgeable against outsider and (strong) insider attacks *according to Definition 3.*

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

Proof (for Claim 5.2). We investigate the simulation of all oracles and local computations.

Simulation of Initialize: Observe that by construction and by the fact that $\mathcal{S}$ 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 $\mathcal{S}$ does not require a master secret key, 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 $\mathcal{S}$ does not require a master secret key , 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.

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

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

Fig. 7. A simulated version of Finalize for our construction DFSS

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

Proof (Proof of Claim 5.2). 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.[4]

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

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

hull $\mathcal{F}^*(\lambda,(f,m))$. Also, there was no query to $\mathsf{Query[Eval]}(pk_{ev},\alpha,m,pk'_{ev},\sigma')$ 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 $\mathsf{Query[Sig]}$ with parameters $(pk_{ev},(f,k),m)$. Thus, $\mathcal{B}$ never sent h_k to $\mathsf{Query[Sig]}_{\mathcal{C}}$. and thus, S contains a signature $\sigma_x = \sigma_k$ that verifies with pk_{sig} for the message h_k.

$0 < i < k$: There was a call to $\mathsf{Query[Sig]}$ with parameters $(pk_{ev},(f,k),m)$. And for all $0 < j \leq i$ there was a call to $\mathsf{Query[Eval]}$ 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 $\mathsf{Query[Eval]}$ 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 $\mathsf{Query[Sig]}_{\mathcal{C}}$ and thus, σ_i and h_i fulfill our claim.

$i=k$: There was a call to $\mathsf{Query[Sig]}$ with parameters $(pk_{ev},(f,k),m)$. 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^*,σ^*,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 5.2, $\mathcal{B}$ 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 5.2, 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 $p(\lambda)+1$. Since $\mathcal{B}$ is an efficient algorithm, this concludes the proof.

Lemma 2. *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, then the construction* DFSS *presented in Section 5 is* secure against chosen function attacks (CFA) *as in Definition 4.*

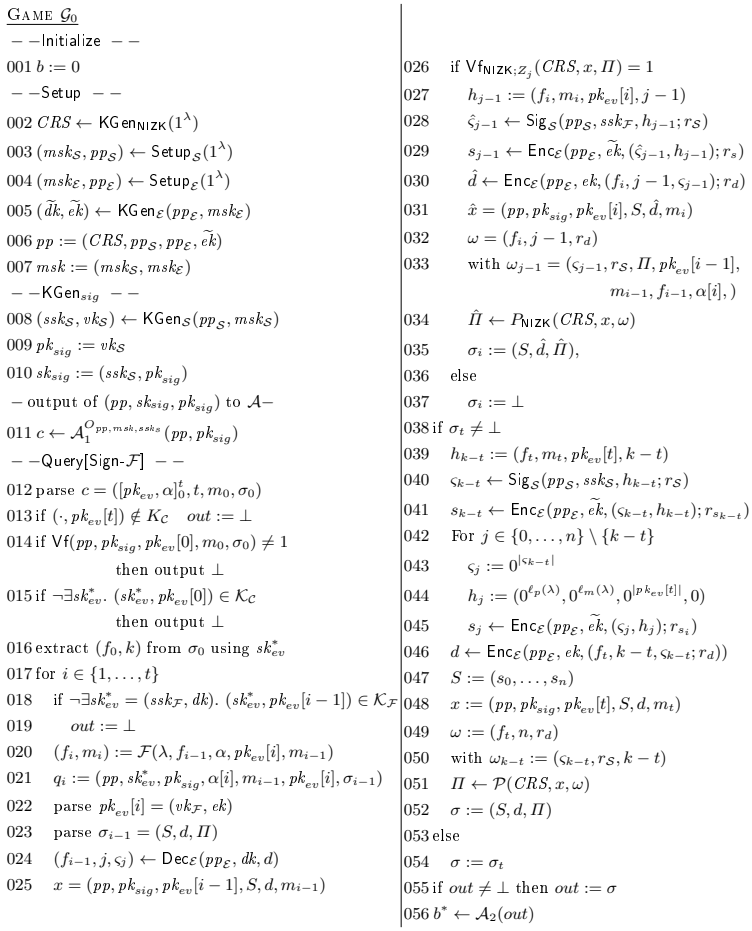

Fig. 8. Definition of GAME $\mathcal{G}_0$ for Section 5.2

For showing this lemma we will first give a game-based proof for an adversary that only uses the oracle $\mathsf{Query}[\mathsf{Sign}\text{-}\mathcal{F}]$ once. We proceed using a hybrid argument that shows that the existence of a successful adversary that makes polynomially many calls to $\mathsf{Query}[\mathsf{Sign}\text{-}\mathcal{F}]$ 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 4 with non negligible advantage. For simplicity we will write $\mathcal{A}$ for $\mathcal{A}_{\text{S-Insider}}$ in this proof.

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

Proof (Proof for Claim 5.2). The challenger uses the uniformly distributed value b only when Query[Sign-$\mathcal{F}$] is called. Thus, if $\mathcal{A}$ does not call Query[Sign-$\mathcal{F}$], the advantage of $\mathcal{A}$ is 0.

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

Let Game $\mathcal{G}_0$ be the original game from Definition 4 where $b = 0$, as defined in Figure 8. As by our claim $\mathcal{A}$ calls Query[Sign-$\mathcal{F}$] only once we will simplify the notation of the game by making the call to Query[Sign-$\mathcal{F}$] explicit. Moreover we make the invokation of Initialize explicit as we will modify it in the following games. The oracles that $\mathcal{A}$ can access (aside from Query[Sign-$\mathcal{F}$]) are as they are formalized in Definition 4. As before, we annotate each line with the game (first digit) and the line within the game (remaining digits).

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 *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 $\mathcal{G}_0$ $\Rightarrow$ 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. Game $\mathcal{G}_0$ and Game $\mathcal{G}_1$ are computationally indistinguishable.

Proof. The indistinguishability follows from the fact that NIZK is zero knowledge. 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 $\mathcal{G}_1$ $\Rightarrow$ 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 $j \in \{0, \ldots, n\}$ 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 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. 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. 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. 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. GAME $\mathcal{G}_4$ and GAME $\mathcal{G}_5$ are computationally indistinguishable.

Proof. As for Claim 5.2, 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.

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 5.2

Via hybrid argument we reduce the case in which the adversary might make polynomially many calls to Query[Sign-$\mathcal{F}$] to the case of Claim 5.2 where the adversary makes at most one call to Query[Sign-$\mathcal{F}$]. We can simulate the calls to Query[Sign-$\mathcal{F}$] both for $b = 0$ and for $b = 1$ using the oracle access to Sig and to $\mathsf{Eval}_{\mathcal{F}}$.

Acknowledgments. We would like the Marc Fischlin, Özgür Dagdelen, and Sebastian Gajek for the helpful discussions and their encouragement to submit our work. We also thank the reviewers for the valuable comments. This work was supported by the German Federal Ministry of Education and Research (BMBF) through funding for the Center for IT-Security, Privacy and Accountability (CISPA – `www.cispa-security.org`) and the project PROMISE. Moreover, it was supported by the Initiative for Excellence of the German federal and state governments through funding for the Saarbrücken Graduate School of Computer Science and the DFG MMCI Cluster of Excellence. Part of this work was also supported by the German research foundation (DFG) through funding for the collaborative research center 1223. Dominique Schröder was also supported by an Intel Early Career Faculty Honor Program Award.

References

1. T. Acar and L. Nguyen. Revocation for delegatable anonymous credentials. In , *PKC 2011*, volume 6571 of *LNCS*, pages 423–440. Springer, Heidelberg, Mar. 2011.
2. J. H. Ahn, D. Boneh, J. Camenisch, S. Hohenberger, a. shelat, and B. Waters. Computing on authenticated data. In , *TCC 2012*, volume 7194 of *LNCS*, pages 1–20. Springer, Heidelberg, Mar. 2012.
3. G. Ateniese, D. H. Chou, B. de Medeiros, and G. Tsudik. Sanitizable signatures. In , *ESORICS 2005*, volume 3679 of *LNCS*, pages 159–177. Springer, Heidelberg, Sept. 2005.
4. N. Attrapadung, B. Libert, and T. Peters. Computing on authenticated data: New privacy definitions and constructions. In *Advances in Cryptology–ASIACRYPT 2012*, pages 367–385. Springer, 2012.

5. N. Attrapadung, B. Libert, and T. Peters. Efficient completely context-hiding quotable and linearly homomorphic signatures. In *Public-Key Cryptography–PKC 2013*, pages 386–404. Springer, 2013.
6. M. Backes, S. Meiser, and D. Schröder. Delegatable functional signatures. Cryptology ePrint Archive, Report 2013/408, 2013. `http://eprint.iacr.org/`.
7. B. Barak and M. Mahmoody-Ghidary. Lower bounds on signatures from symmetric primitives. In *48th FOCS*, pages 680–688. IEEE Computer Society Press, Oct. 2007.
8. 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.
9. 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. Springer, Heidelberg, Aug. 2009.
10. 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. Springer, Heidelberg, Mar. 2008.
11. M. Bellare and G. Fuchsbauer. Policy-based signatures. In *Public-Key Cryptography–PKC 2014*, pages 520–537. Springer, 2014.
12. 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. Springer, Heidelberg, May / June 2006.
13. D. Boneh and D. M. Freeman. Homomorphic signatures for polynomial functions. In , *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 149–168. Springer, Heidelberg, May 2011.
14. 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. Springer, Heidelberg, Mar. 2011.
15. E. Boyle, S. Goldwasser, and I. Ivan. Functional signatures and pseudorandom functions, 2014.
16. S. Brands. Restrictive blinding of secret-key certificates. In , *EUROCRYPT'95*, volume 921 of *LNCS*, pages 231–247. Springer, Heidelberg, May 1995.
17. S. A. Brands. *Rethinking public key infrastructures and digital certificates: building in privacy*. The MIT Press, 2000.
18. 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. Springer, Heidelberg, June 2010.
19. C. Brzuska, M. Fischlin, A. Lehmann, and D. Schröder. Unlinkability of sanitizable signatures. In , *PKC 2010*, volume 6056 of *LNCS*, pages 444–461. Springer, Heidelberg, May 2010.
20. J. Camenisch and T. Groß. Efficient attributes for anonymous credentials. In , *ACM CCS 08*, pages 345–356. ACM Press, Oct. 2008.
21. 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. Springer, Heidelberg, May 2001.
22. J. Camenisch and A. Lysyanskaya. A signature scheme with efficient protocols. In , *SCN 02*, volume 2576 of *LNCS*, pages 268–289. Springer, Heidelberg, Sept. 2003.

23. J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In , *CRYPTO 2004*, volume 3152 of *LNCS*, pages 56–72. Springer, Heidelberg, Aug. 2004.
24. D. Catalano. Homomorphic signatures and message authentication codes. In *Security and Cryptography for Networks - 9th International Conference, SCN 2014, Amalfi, Italy, September 3-5, 2014. Proceedings*, pages 514–519, 2014.
25. D. Catalano, D. Fiore, and B. Warinschi. Homomorphic signatures with efficient verification for polynomial functions. In *Advances in Cryptology–CRYPTO 2014*, pages 371–389. Springer, 2014.
26. M. Chase, M. Kohlweiss, A. Lysyanskaya, and S. Meiklejohn. Succinct malleable NIZKs and an application to compact shuffles. In , *TCC 2013*, volume 7785 of *LNCS*, pages 100–119. Springer, Heidelberg, Mar. 2013.
27. M. Chase, M. Kohlweiss, A. Lysyanskaya, and S. Meiklejohn. Malleable signatures: New definitions and delegatable anonymous credentials. pages 199–213, 2014.
28. 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. Springer, Heidelberg, Mar. 2009.
29. D. M. Freeman. Improved security for linearly homomorphic signatures: A generic framework. In , *PKC 2012*, volume 7293 of *LNCS*, pages 697–714. Springer, Heidelberg, May 2012.
30. G. Fuchsbauer and D. Pointcheval. Anonymous proxy signatures. In , *SCN 08*, volume 5229 of *LNCS*, pages 201–217. Springer, Heidelberg, Sept. 2008.
31. 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.
32. 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.
33. R. Johnson, D. Molnar, D. X. Song, and D. Wagner. Homomorphic signature schemes. In , *CT-RSA 2002*, volume 2271 of *LNCS*, pages 244–262. Springer, Heidelberg, Feb. 2002.
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. Springer, Heidelberg, Mar. 2011.
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. B. Libert, T. Peters, M. Joye, and M. Yung. Linearly homomorphic structure-preserving signatures and their applications. *Des. Codes Cryptography*, 77(2-3):441–477, 2015.
37. 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.
38. R. Steinfeld, L. Bull, and Y. Zheng. Content extraction signatures. In , *ICISC 01*, volume 2288 of *LNCS*, pages 285–304. Springer, Heidelberg, Dec. 2002.