Operational Signature Schemes

Michael Backes
CISPA, Saarland University

Özgür Dagdelen*
TU Darmstadt

Marc Fischlin
TU Darmstadt

Sebastian Gajek
NEC Research Labs

Sebastian Meiser
CISPA, Saarland University

Dominique Schröder
CISPA, Saarland University

Abstract

Functional encryption, as introduced by Boneh, Sahai, and Waters (TCC'11), generalizes public-key encryption systems to include functional decryption capabilities. Recently, Boyle et al. as well as Bellare and Fuchsbauer (both PKC'14) formalized analogous notions for signature schemes. Here we discuss that both their notions are limited in terms of expressiveness in the sense that they cannot cast known signature schemes supporting operations on data in their frameworks. We therefore propose a notion of what we call, for sake of distinctiveness, operational signature schemes which captures functional signatures, policy-based signatures, sanitizable signatures, P-homomorphic signatures, ring signatures, aggregate signatures etc., and also their message authentication code counterparts.

We discuss possible instantiations for operational signatures. We give some positive result about achieving our general notion of operational signatures presenting a compact construction that relies on a new combination of indistinguishability obfuscation and random oracles. We then indicate that it is unlikely to be able to instantiate operational signature schemes in general using one-wayness and, under some circumstances, even using specific "non-interactive" assumptions like RSA.

Keywords: Foundations, functional cryptography, random oracle obfuscation, message authentication systems, existential relations

*Part of the research was conducted while interning at NEC Research Labs

1 Introduction

Functional encryption [SW05, BSW11] for functionality F allows one to encrypt messages m such that decryption, depending on a secret key $\mathsf{sk}_{\mathrm{id}}$ for identifier id, yields a function $F(\mathrm{id}, m)$ of this message. The notion is general enough to capture a vast number of "special" encryption schemes like identity-based encryption or attribute-based encryption. The original notion has been initially restricted to single ciphertexts and has therefore not subsumed, for example, homomorphic encryption. Nonetheless, the definition can be extended to also include (fully) homomorphic encryption schemes [ABF+13].

Recently, Boyle et al. [BGI14] transferred the idea of functional encryption to signatures. They basically say that, with knowledge of a signing key sk_f, one can sign any message $f(m)$. The related notion of policy-based signatures has been introduced by Bellare and Fuchsbauer [BF14], where a signer holding a key sk_P can sign messages m which conform to the respective policy P. Similar to the core case of functional encryption both notions, however, neglect schemes in which the signature is derived from a set of already signed messages (as required for example in the case of homomorphic signatures).

Some progress towards capturing homomorphic schemes in a general way has been achieved with the notion of P-homomorphic signatures, put forward by Ahn et al. [ABC+12]. There, one can deduce a signature for any message m for which one already holds signatures for messages $m_1, m_2, \ldots$ such that the predicate $\mathsf{P}(m, m_1, m_2, \ldots)$ is satisfied. Yet, such P-homomorphic schemes do not cover, for example, sanitizable signatures, because the predicate does not take into account any requirements on the keys. The same holds for approaches in a recent work by Chase et al. [CKLM13], transforming input message-signature pairs to a new signature for the transformed message all under the *same* signer's key.

The purpose of our work here is to bridge this gap between functional keys and malleable signatures. Our goal is to introduce a general notion which allows us to capture more broadly signature schemes (and also message authentication codes) supporting a rich class of operations, and to discuss the possibility to instantiate them. Since the term "functional signatures" has somewhat been set by the work of Boyle et al. for specific schemes, we call our schemes here *operational signature schemes* (OSS). In contrast to previous work, our notion is general enough to capture message authentication codes (MACs). On the constructive side, the general approach allows us to provide universal solutions, instead having to design dedicated schemes from scratch. At the same time, if defined properly, then any infeasibility result about operational signature schemes immediately tells us about the hardness of devising signature schemes for certain operations. Both features show up in our results.

1.1 How to Define Operational Signature Schemes

To define operational signature schemes we consider an ensemble $\mathcal{P}$ of predicates P, describing the "admissible operations". These predicates operate on sets of pairs (id, m) consisting of key identifiers id, which one can view as a public handle for the corresponding secret (resp. public) key sk_{id} (resp. pk_{id}), and a message m. More precisely, a predicate takes as

input a sequence $(id_{\mathsf{in},i}, m_{\mathsf{in},i})_{1,2,\ldots}$, a pair $(id_{\mathsf{eval}}, m_{\mathsf{eval}})$, a pair $(id_{\mathsf{out}}, m_{\mathsf{out}})$ and returns 0 or 1, with the following interpretation:

> If for each i the tag of the input message $m_{\mathsf{in},i}$ verifies under the key for identifier $id_{\mathsf{in},i}$, and one evaluates these messages under the key for id_{eval} to obtain message m_{eval}, then the derived tag verifies for message m_{out} under the key for identifier id_{out}.

The key identifiers id are divided into secret and public keys, determining if the algorithm can access them, such that there is no need to differentiate between MACs and signatures with this approach. Note that we could also allow multiple messages-key pairs for evaluation and multiple output message-key pairs in our definition. Since we are not aware of any example, for sake of readability, we refrain from using this more general notion.

As an example, consider a sanitizable signature scheme [ACMT05] where a designated party, the sanitizer, can modify a given message-signature pair of another user with the help of a secret key into a signature for a new qualified message. For such a sanitizable signature scheme id_{eval} would correspond to the sanitizer's secret key and $id_{\mathsf{in}} = id_{\mathsf{out}} = id_{\mathrm{pub}}$ to the signer's public verification key, m_{in} would be the original message, $m_{\mathsf{eval}} = m_{\mathsf{out}}$ be the sanitized message, and the predicate P describes the sanitization process. In this sense it is understood that sanitizing m_{in} with a valid tag under the key id_{pub} to m_{eval} one derives a signature for $m_{\mathsf{eval}} = m_{\mathsf{out}}$ which is again valid under id_{pub}. Hence, the starting message may have been created under the original key, or may, recursively, have been a sanitized version itself. If we would like to implement for example a "sanitize only once" scheme then we can define id_{out} to be different from id_{pub} such that one can verify the derived signature under id_{out}, but one cannot use the derived signature as input again (because the predicate P evaluates to 1 only if the input messages verify under $id_{\mathsf{in}} = id_{\mathrm{pub}} \neq id_{\mathsf{out}}$).

As we explain formally later on, the notion of operational signature schemes comprises special notions of signatures supporting operations in the literature. This includes sanitizable signatures [ACMT05], redactable signatures [CLX09], P-homomorphic signatures [ABC+12], fully homomorphic message authenticators and signatures [GW13, CF13, BF11, CFGN14, CFW14], and even ring signatures [RST01a]. Our definition also captures the concepts of functional signatures introduced in [BGI14], policy-based signatures in [BF14], and delegatable functional signatures [BMS13], where parties can delegate signing capabilities to another party. Moreover, it covers aggregate signatures [BGLS03] where a public algorithm takes as input a set of message/signature pairs and outputs a single element that is as big as an ordinary signature and which verifies if all message/signature pairs were valid. A related result about generalizing aggregate signature schemes has very recently been proposed by Hohenberger et al. [HKW14].

Our notion of course has its limitations in terms of expressiveness. For example, while we are able to model ring signatures [RST01b] with our approach, group signatures [CH91] escape our framework. The reason is that group signatures usually include a group manager which can revoke anonymity of signers in case of a dispute. This, however, is clearly more than any *pure* signature functionality provides; we cannot hope to capture arbitrary multi-

party protocols which include signatures (or MACs). Still, it appears to us that we can express all known *signature-only* based schemes within our framework.[1]

1.2 Security Notions for Operational Signature Schemes

The very basic notion for (ordinary) signature schemes and MAC schemes is unforgeability, usually under adaptive chosen message attacks and providing existential unforgeability. Since our notion of operational signatures captures these ordinary schemes we adapt the idea behind unforgeability and generalize it for our approach. Basically, we require that an adversary can forge a signature for a "fresh" identifier-message pair (id^*, m^*), where freshness says that any direct signature (or MAC) query about a message and a key identifier renders this pair as "unfresh", and any pair $(id_{\mathsf{out}}, m_{\mathsf{out}})$ which can be derived recursively according to the rule from known input pairs $(id_{\mathsf{in},i}, m_{\mathsf{in},i})$ and id_{eval} above, is also unfresh. Again, known unforgeability requirements for schemes fall under our general notion, such as sanitizable signatures [BFF+09], redactable signatures [CLX09, BBD+10], and P-homomorphic signatures [ABC+12, ALP12].

Besides unforgeability, sometimes signature and message authentication schemes require additional privacy properties, e.g., that redacted parts of the original message cannot be recovered [BBD+10, BFLS10], or that the signature hides the message [Bel06]. Boyle et al. [BGI14] for their functional signatures also define a privacy notion which basically hides the function which has been used in the generation of the signatures. While the unforgeability requirement is nowadays undisputed, the concrete privacy requirement, if needed at all, for signatures and MACs varies significantly with the application. We have chosen to give a general definition for our operational signature schemes in the spirit of context hiding for P-homomorphic signatures [ABC+12, ALP12]. The notion basically says that one cannot distinguish between any two cases in which a tag for pair $(id_{\mathsf{out}}, m_{\mathsf{out}})$ could have been generated. We discuss that this definition captures for example some of the anonymity requirements for ring signatures [BKM09]. It also comprises the notion of [BGI14] for their functional signatures, and the privacy notion of delegatable functional signatures [BMS13].

1.3 Constructions, Techniques, and Results

We next discuss possible instantiations for operational signatures. We give some positive result about achieving our general notion of operational signatures. The universality of our construction comes at a price, though: it relies on a new combination of indistinguishability obfuscation (iO) [BGI+12, GGH+13, PST14] and random oracles [BR93].

Construction of operational signatures. The main idea of our construction is to push all evaluation steps into an obfuscated circuit. This circuit contains a master secret signing key, used for all users in the system, and issues signatures for identifier-message pair

[1] Note that similarly to policy-based signatures [BF14] one is able to construct a group signature scheme from operational signature schemes, if one adds the property of extractability. This extra property, however, seems to be much more than what is required for most of the operational schemes.

$(id_{\mathsf{out}}, m_{\mathsf{out}})$ if and only if the user supplies valid inputs $(id_{\mathsf{in},i}, m_{\mathsf{in},i})_{i=1,2...,}$, $(id_{\mathsf{eval}}, m_{\mathsf{eval}})$ and P, with the corresponding signatures and key to id_{eval}. That is, the circuit first checks the validity of the inputs and, if valid, then issues a signature under its master secret. General verification of such signatures is then carried out via the public key to the master secret.

The construction immediately comes with useful features. Since derived signatures are basically regular signatures under the master key, we obtain succinct operational signatures, enabling us to make recursive calls without having to worry about increasing signature lengths. Privacy holds in a strong sense as the final signature does not carry any information about the inputs.

Unforgeability of our construction should intuitively follow from the unforgeability of the master scheme. Alas, this property requires a more sophisticated argument. Since the scheme is operational it allows us to deduce further signatures for derivable messages. An adversary could thus forge a signature for some input messages and then execute a sequence of calls to the obfuscated circuit to cover its original forgery. We, however, need to be able to find the source for that final forgery. This is where the random oracle and its observability property turn out to be handy: If the adversary also needs to provide a (random oracle) hash value of its inputs, and we additionally let the obfuscated circuit check that hash value, then we can extract the source forgery from the adversary's queries to the random oracle. Indeed, we show that this approach works and allows us to argue unforgeability.[2]

Obfuscation and random oracles. At first glance using obfuscators —which require the code of some program or circuit— which take random-oracle based circuits —for which one cannot provide a short code— sounds provoking. In fact, it first and foremost requires us to define how such an obfuscator works in the oracle world. Once this technical issue has been addressed we can argue about the meaningfulness of the approach.

To define formally what obfuscation in the random oracle model means we assume that circuits passed to the obfuscator may now contain special random oracle gates. For our application we will use the random oracle only as an initial step. In this case we can think of the obfuscator of a circuit $C^H(\cdot)$ with access to random oracle H to be of the form $C^H(x) = C'(x, H(x))$ and we merely obfuscate the C' part. Indistinguishability of the obfuscator's output then requires indistinguishability of the C' part. Some additional care is required to make sure that an adversary cannot run C' on malformed inputs (x, h) to gain additional information.

The above approach corresponds to the mental model used in other random-oracle based constructions. The honest party can run the original program faithfully and the adversary can only access the "other code". The general idea of the random oracle method is that, eventually the random oracle is instantiated by a concrete hash function, simply putting in the function's code where oracle calls have been made before. For our obfuscator this means that it can then be run on the full code of the circuit. It will then hide the code of such evaluations within the whole circuit, and prevent that one can for example inject

[2]We also take advantage of some form of programmability of the random oracle, in order to be able to non-adaptively redirect outputs to pseudorandom values.

intermediate values "half way through the evaluations" of such steps, e.g., after the H evaluation. Note that secure obfuscators need provide these properties, and constructions such as in [GGH+13, PST14] actually address this.

Operational signature schemes imply blind signatures. We then indicate that it is unlikely to be able to instantiate operational signature schemes in general using one-wayness and, under some circumstances, even using specific "non-interactive" assumptions like RSA. More precisely, we show that unforgeable and private operational signature schemes allow one to build two-move blind signature schemes, such that we can then apply known impossibility results for such blind signatures [FS10, KSY11]. Since we determine exactly the predicate P for which our implication is satisfied, one can easily identify specific signature schemes for which the same limitations hold. Our result also underlines the difference to functional signatures as in [BGI14]. While in [BGI14], it was shown that succinct functional signatures cannot be build from one-way functions, our results supplements their result in the sense that demanding privacy (instead of succinctness) still makes constructions hard to be built out of one-way functions.

Historical note. Our contributions include and subsume the preliminary work called (Delegatable) Functional Signatures (DFS) by Backes, Meiser, and Schröder [BMS13]. Here we generalize their notions and results in several ways. Our definition covers both MACs and signature schemes and admits arbitrary input sequences (see Section 2), thereby covering a much larger class of known signature schemes (cf. Section 3). We also provide a construction for the more general notion in Section 4. We also adapt their impossibility result, which shows that constructing DFS requires blind signatures to the more general case of OSS in Section 5.

2 Operational Signature Schemes

We begin this section by giving a definition of operational signatures (and message authentication) schemes for predicates $\mathcal{P}$ along the line of some example instantiation of existing schemes.

2.1 Identifiers, Keys, and Predicates

Throughout the paper, let λ be the security parameter (but which we drop if it is clear from the context). Operational message authenticators will work over ensembles $\mathcal{ID} = \{\mathcal{ID}_\lambda\}_{\lambda\in\mathbb{N}}$ and $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ of finite sets of key identifiers $\mathcal{ID}_\lambda$ and messages spaces $\mathcal{M}_\lambda$. The set $\mathcal{ID}_\lambda$ specifies the (public) key identifiers id for security parameter λ which can be thought of as handles for the actual, possibly private cryptographic keys k_{id}. It is often also useful to identify a subset of key identifiers $\mathcal{ID}_{\text{pub}} \subseteq \mathcal{ID}_\lambda$ as the set of identifiers for public cryptographic keys.

The admissible operations are given by an ensemble $\mathcal{P} = \{\mathcal{P}_\lambda\}_{\lambda\in\mathbb{N}}$ of finite collections $\mathcal{P}_\lambda$ of predicates. The input to such an n-ary predicate $P \in \mathcal{P}_\lambda$ consists of n pairs (id, m) from $\mathcal{ID}_\lambda \times \mathcal{M}_\lambda$. It is convenient to designate the first $n-2$ input pairs $(id_{\mathsf{in},i}, m_{\mathsf{in},i}))_{i=1,2,\ldots,n-2}$ as the input pairs, and the final pairs $(id_{\mathsf{eval}}, m_{\mathsf{eval}})$, $(id_{\mathsf{out}}, m_{\mathsf{out}})$ as the evaluation and the output pair, respectively. Below we sometimes abbreviate $\big((id_{\mathsf{in},i}, m_{\mathsf{in},i})_{i=1,2,\ldots,n-2}, (id_{\mathsf{eval}}, m_{\mathsf{eval}}), (id_{\mathsf{out}}, m_{\mathsf{out}})\big)$ by $(\mathbf{id}, \mathbf{m})$.

Intuitively, the predicate P upon input $(id_{\mathsf{in},i}, m_{\mathsf{in},i})_{i=1,2,\ldots,n-2}$, $(id_{\mathsf{eval}}, m_{\mathsf{eval}})$, and $(id_{\mathsf{out}}, m_{\mathsf{out}})$ should return 1 if one holds authenticators for each $m_{\mathsf{in},i}$ which verify under the cryptographic keys to $id_{\mathsf{in},i}$, and if one evaluates these messages under key id_{eval} to m_{eval}, then one derives an authenticator for m_{out} which verifies under the key to identifier id_{out}. An example would be homomorphic signatures where the predicate evaluates to 1 if the output message is the sum of the input messages, independently of m_{eval}:

$$P(\mathbf{id}, \mathbf{m}) = 1 \iff \sum_{i=1}^{n-2} m_{\mathsf{in},i} = m_{\mathsf{out}}.$$

Note that the predicate P only operates on key-message pairs (id, m) and not signatures, of course. The actual transformation of signatures will be carried out by an algorithm Eval which takes tuples $(id_{\mathsf{in},i}, m_{\mathsf{in},i}, \sigma_{\mathsf{in},i})_i$ including authenticators $\sigma_{\mathsf{in},i}$ and a pair $(\mathsf{k}_{id_{\mathsf{eval}}}, m_{\mathsf{eval}})$ with the actual cryptographic key $\mathsf{k}_{id_{\mathsf{eval}}}$ for identifier id_{eval} as input, and returns an authenticator for m_{out} under identifier id_{out}. For sake of simplicity we often write

$$(\mathbf{id}[\mathsf{k}_{\mathsf{eval}} \to \mathbf{id}_{\mathsf{eval}}], \mathbf{m}, \sigma)$$

for the input $((id_{\mathsf{in},i}, m_{\mathsf{in},i}, \sigma_{\mathsf{in},i})_i, (\mathsf{k}_{id_{\mathsf{eval}}}, m_{\mathsf{eval}}), (id_{\mathsf{out}}, m_{\mathsf{out}}))$ to Eval, where Eval takes the key $\mathsf{k}_{id_{\mathsf{eval}}}$ instead of the index.

2.2 Syntax

We begin by describing the interfaces of operational signature schemes.

Definition 2.1 (Operational Signature Scheme)**.** *An operational signature scheme* OSS *for predicates* $\mathcal{P}$ *consists of four PPT algorithms* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Verify})$ *such that*

Setup: *The probabilistic parameter generation algorithm* Setup *takes as input a security parameter* 1^λ *and outputs a master secret* MSK *and some public parameters* PP*. We assume that all algorithms get* PP *as input, but we drop it to simplify the notations.*

Key Generation: *The probabilistic key generation algorithm* KeyGen *takes as input a master secret* MSK *and a key identifier* $id \in \mathcal{ID}$*. It outputs an operational key* k_{id}*.*

(If queried again on $id \in \mathcal{ID}$*, the algorithm returns the same* k_{id}*. Similarly, we assume that* KeyGen *could output related keys for similar identifiers such as the corresponding secret and public key for identifiers* id_{sig} *and* id_{pub}*. Thereby, one can, for example, fetch the public key via* KeyGen *only, without learning the secret key.)*

Evaluation: *The (probabilistic) signature algorithm* Eval *takes as input a sequence* $(\mathbf{id}[\mathsf{k}_{eval} \to \mathbf{id}_{eval}], \mathbf{m}, \sigma)$ *consisting of triples* $(id_{in,i}, m_{in,i}, \sigma_{in,i})_{i=1}^{n-2}$*, the cryptographic key* k_{eval} *for evaluating message* m_{eval}*, the pair of target values* (id_{out}, m_{out})*, and (the description of) a predicate* P*. The algorithm outputs a* signature σ *in a set* $\mathcal{S}$*, or a special symbol* $\perp \notin \mathcal{S}$*.*

Verification: *The (deterministic) verification algorithm* Verify *takes as input an operational key* k_{out}*, a message* $m_{out} \in \mathcal{M}$*, and a signature* σ*. It outputs a bit* $b \in \{0, 1\}$*.*

An operational signature scheme for predicates $\mathcal{P}$ *is correct, if for any* $P \in \mathcal{P}_\lambda$*, and all message tuples* $\mathbf{m} = (m_{in,1}, \ldots, m_{in,n-2}, m_{eval}, m_{out}) \in \mathcal{M}_\lambda^n$*, all key identifiers* $\mathbf{id} = (id_{in,1}, \ldots, id_{in,n-2}, id_{eval}, id_{out}) \in \mathcal{ID}_\lambda^n$*, and all signatures* $\sigma = (\sigma_1, \ldots, \sigma_{n-2}) \in \mathcal{S}_\lambda^{n-2}$*, where* $n \geq 2$ *is the arity of* P*, we have*

$$Pr\left[\mathsf{Verify}(\mathsf{k}_{out}, m_{out}, \sigma_{out}) = P(\mathbf{id}, \mathbf{m}) \;\middle|\; \begin{array}{c} (\mathsf{MSK}, \mathsf{PP}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{k}_{id})_{id \in \mathbf{id}} \leftarrow \mathsf{KeyGen}(\mathsf{MSK}, id)_{id \in \mathbf{id}} \\ \sigma_{out} \leftarrow \mathsf{Eval}((\mathbf{id}[\mathsf{k}_{eval} \to \mathbf{id}_{eval}], \mathbf{m}, \sigma), P) \\ \forall i \in [n-2] : \mathsf{Verify}(\mathsf{k}_{id_{in,i}}, m_{in,i}, \sigma_i) = 1 \end{array}\right] = 1 - \varepsilon(\lambda)$$

where the probability is taken over the coin tosses of Setup*,* KeyGen*, and* Eval*.*

Note that our notion, in particular, is a generalization of functional signatures [BGI14], policy-based signatures [BF14], sanitizable signatures [ACMT05], redactable signatures [CLX09], aggregate signatures [BGLS03], homomorphic signatures/MACs [GW13, CF13, BF11, CFGN14, CFW14]. We discuss several of these examples in Appendix A. In the forthcoming section we show how to cast regular signatures and MACs in our framework.

2.3 Signatures and MACs as Operational Schemes

To cover regular signature schemes we note that we can take up again the idea of designating some key identifiers $\mathcal{ID}_{\text{pub}} \subseteq \mathcal{ID}$ as public keys, which are given to all parties (and which also the scheme's algorithms like Eval can take as additional input or access via KeyGen). In this sense, for a digital signature scheme in the single-user setting the key space is $\mathcal{ID} = \{id_{\text{sig}}, id_{\text{pub}}\}$ with $id_{\text{sig}} \in \mathcal{ID} \setminus \mathcal{ID}_{\text{pub}}$ being the secret signing key and $id_{\text{pub}} \in \mathcal{ID}_{\text{pub}}$ being the verification key. Simply consider the predicate

$$P_{\text{sig}}(\mathbf{id}, \mathbf{m}) = \begin{cases} 1 & \text{if } id_{\mathsf{eval}} = id_{\text{sig}} \in \mathcal{ID},\ id_{\mathsf{out}} = id_{\text{pub}} \in \mathcal{ID}_{\text{pub}},\ m_{\mathsf{eval}} = m_{\mathsf{out}} \in \mathcal{M} \\ 0 & \text{otherwise.} \end{cases}$$

In our terminology we formally thus operate on an empty sequence of input pairs $(id_{\mathsf{in},i}, m_{\mathsf{in},i})_i$ and allow any tag produced for message $m_{\mathsf{eval}} = m_{\mathsf{out}}$ under id_{sig} to be publicly verifiable under id_{pub}. In fact, this allows us to spare a formal definition of the signature generation algorithm, but we can view this as a special case of the Eval algorithm.

Analogously, to devise the notion of message authentication schemes, let $\mathcal{ID} = \{id_{\text{mac}}\}$ be the key space consisting of a symmetric key id_{mac} only, and let $\mathcal{ID}_{\text{pub}} = \emptyset$. A message authentication scheme is a special case of an operational scheme for predicate

$$P_{\text{mac}}(\mathbf{id}, \mathbf{m}) = \begin{cases} 1 & \text{if } id_{\mathsf{eval}} = id_{\mathsf{out}} = id_{\text{mac}} \in \mathcal{ID},\ m_{\mathsf{eval}} = m_{\mathsf{out}} \in \mathcal{M} \\ 0 & \text{otherwise} \end{cases}$$

In other words, P_{mac} is identical to the signature functionality P_{sig} with the exception that key id_{mac} is used for evaluation and verification.

Looking ahead, many operational signature systems (cf. Appendix A) comprise either of the two predicates. We then call the system well-formed.

Definition 2.2 (Well-Formedness)**.** *We call a predicate ensemble $\mathcal{P}$* well-formed *if the basic signature (or, depending on the setting, MAC) predicates are included, that is, if $P_{mac} \in \mathcal{P}$ or $P_{sig} \in \mathcal{P}$.*

3 Security Models

We now turn to the description of the security properties.

3.1 Unforgeability

To define unforgeability we need to specify the set of messages for which the adversary can trivially deduce a valid signature (or MAC, for that matter). In the case of single-user ordinary signatures and standard unforgeability, this set is simply the set of queried messages to the Eval oracle, implementing the signing process. In the multi-user case, this set contains all pairs (id, m) for which the adversary has made such a query. However, the adversary should be considered successful if it manages to create a signature for a pair (id', m) where it may have queried the signing oracle for (id, m) for a different key. In the case of non-trivial operations, we also need to take into account any messages for which the adversary can deduce tags trivially.

In the course of the attack, the adversary will be allowed to ask for keys associated to identifiers, in addition to the public ones for key identifiers in $\mathcal{ID}_{\text{pub}}$ which it can fetch without punishment at any point, to derive signatures via Eval for $(\mathbf{id}, \mathbf{m}, P)$, and to verify messages via Verify. (We call the corresponding oracles $\mathsf{Eval}', \mathsf{Verify}'$ as they partly operate on key identifiers instead of actual keys.) To define messages m for which the adversary can trivially compute a MAC or signature under some identifier id, we keep track of the keys $\mathcal{QID}$ which the adversary requested from KeyGen (where $\mathcal{QID}$ initially contains the public keys $\mathcal{ID}_{\text{pub}}$), all queries $\mathcal{QE}$ of the form $((id_{\mathsf{in},i}, m_{\mathsf{in},i})_i, (id_{\mathsf{eval}}, m_{\mathsf{eval}}), P)$ made to Eval', where the pairs $(id_{\mathsf{in},i}, m_{\mathsf{in},i})$ also come with signatures $\sigma_{\mathsf{in},i}$, but which are not added to $\mathcal{QE}$. We inductively then define the set Triv of "trivial" pairs (id, m) as follows:

Definition 3.1 (Trivial Identifier-Message-Pairs)**.** *Let* $\mathsf{OSS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Verify})$ *be an operational signature scheme for predicated* $\mathcal{P}$ *over* $\mathcal{ID}_{pub} \subseteq \mathcal{ID}$ *and* $\mathcal{M}$*. For sets* $\mathcal{QID} \subset \mathcal{ID}$ *and* $\mathcal{QE} \subseteq (\mathcal{ID} \times \mathcal{M})^+ \times \mathcal{P}$ *let the set* $\mathsf{Triv}(\mathcal{QID}, \mathcal{QE}) \subseteq \mathcal{ID} \times \mathcal{M}$ *be the following set:*

Case 1 (Base Case): *for any* $(\mathbf{id}, \mathbf{m}, P) \in \mathcal{QE}$ *we have* $(id_{\mathsf{out}}, m_{\mathsf{out}}) \in \mathsf{Triv}(\mathcal{QID}, \mathcal{QE})$*;*
// results of evaluations queries are trivial (where we leave it to the adversary to avoid irregular queries and to keep the set of trivial messages small[3]*)*

Case 2 (Recursion): *If* $(id_{\mathsf{in},i}, m_{\mathsf{in},i}) \in \mathsf{Triv}(\mathcal{QID}, \mathcal{QE})$ *for each* i*, and* $id_{\mathsf{eval}} \in \mathcal{QID}$*, then also any* $(id_{\mathsf{out}}, m_{\mathsf{out}})$ *for which there is some* $P \in \mathcal{P}$ *and some* m_{eval} *such that* $P(\mathbf{id}, \mathbf{m}) = 1$*, is in* $\mathsf{Triv}(\mathcal{QID}, \mathcal{QE})$*;*
// messages which are deducible with the known key for id_{eval} *are trivial*

Note that in the standard case, the set Triv only contains the signature queries, or in case the adversary corrupts the key holder and requests the key to id_{sig}, also any pair (id_{pub}, m) in $m \in \mathcal{M}$. In the case of redactable schemes, the set also contains all redacted messages for which a signature has been created, or which have already been redacted successfully. For sanitizable schemes the set contains the signature and evaluation queries only, if the adversary does not request the sanitizing key. In [BFF+09] for sanitizable signatures security against these "outside" attackers are captured under the term unforgeability. If the adversary requests the sanitizing key, and becomes an "inside" attacker, then the trivial set also contains sanitized messages of otherwise available pairs. Security against such insiders for sanitizable signatures is called immutability in [BFF+09]. Both notions are captured in our single definition, by leaving the choice to request the sanitizer's secret key to the adversary.

In the definition below we now demand that the adversary cannot successfully create a forgery σ^* for some message m^* which verifies under chosen key id^* such that the pair (id^*, m^*) is non-trivial. It suffices for the adversary to find such tuples at any point during the attack, where the set of trivial pairs grows over time, depending on the key generation and evaluation queries. There might be some keys that are publicly known and computing a forgery for these keys is not a valid attack; recall that we denote this set of (public-key) identifiers by $\mathcal{ID}_{\text{pub}}$ and we assume that these keys are known to the adversary (and can be fetched by the adversary at will). This is captured formally by setting $\mathcal{QID} = \mathcal{ID}_{\text{pub}}$.

Definition 3.2 (EUF-CMA Security.)**.** *An operational signature scheme* $\mathsf{OSS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Verify})$ *for predicates* $\mathcal{P}$ *over* $\mathcal{ID}_{pub}, \mathcal{ID}, \mathcal{M}$ *is* $(t, q_k, q_e, q_v, \varepsilon)$*-existentially-unforgeable under adaptively chosen-message attacks if for any algorithm* $\mathcal{A}$ *with runtime* t *and making at most* q_k *(resp.* q_e *and* q_v*) queries to his key-generating (resp. evaluation and verifying) oracle, the probability that the following experiment returns* 1 *is at most* ε*.*

[3]For example, while verifying the input tags in case of signatures is possible for Eval' with the help of the public verification keys, for MACs we would otherwise put the burden of distinguishing valid input tags from invalid ones on Eval' without being able to rely on any keys.

Experiment $\mathbf{Exp}^{\textit{EUF-CMA}}_{\mathsf{OSS},\mathcal{A}}(\lambda)$

Set $\mathcal{QE} = \emptyset$ *and* $\mathcal{QID} = \mathcal{ID}_{pub}$.
$(\mathsf{MSK}, \mathsf{PP}) \leftarrow \mathsf{Setup}(1^\lambda)$
$(id^*, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{KeyGen}',\mathsf{Eval}',\mathsf{Verify}'}(\mathsf{PP})$
Return 1 *if, at some point,* $\mathcal{A}$ *queried* Verify' *about* $(\mathsf{PP}, id^*, m^*, \sigma^*)$ *such that*
(a) $\mathsf{Verify}'(\mathsf{PP}, id^*, m^*, \sigma^*) = 1$, *and*
(b) $(id^*, m^*) \notin \mathsf{Triv}(\mathcal{QID}, \mathcal{QE})$ *at this point.*

If $\mathcal{A}$ *queries* $\mathsf{KeyGen}'(id)$,
add id *to* $\mathcal{QID}$, *and*
return $\mathsf{KeyGen}(\mathsf{MSK}, id)$

If $\mathcal{A}$ *queries* $\mathsf{Verify}'(id, m, \sigma)$,
return $\mathsf{Verify}(\mathsf{k}_{id}, m, \sigma)$.

If $\mathcal{A}$ *queries* $\mathsf{Eval}'(\mathsf{PP}, (\mathbf{id}, \mathbf{m}, \sigma), P)$,
add $(\mathbf{id}, \mathbf{m}, P)$ *to* $\mathcal{QE}$, *and*
return $\mathsf{Eval}(\mathsf{PP}, (\mathbf{id}[\mathsf{k}_{eval} \to \mathbf{id}_{eval}], \mathbf{m}, \sigma), P)$

The probability is taken over all coin tosses of algorithms Setup, KeyGen, Eval, *and* $\mathcal{A}$. *We let* $\boldsymbol{Adv}^{\textit{EUF-CMA}}_{\mathsf{OSS}}(t, q_k, q_e, q_v)$ *denote (a bound on) the value* ε *for which the scheme* OSS *is* $(t, q_k, q_e, q_v, \varepsilon)$*-existentially-unforgeable. If* OSS *is* $(t, q_k, q_e, q_v, \varepsilon)$*-existentially-unforgeable for time* t *and queries* q_k, q_e, q_v *polynomial in* λ *and* ε *is negligible in* λ, *then we simply say* OSS *is unforgeable.*

Note that the above notion simplifies in case of digital signature schemes where verification is public, and deterministic MACs where the verification is carried out by re-computing MACs, to the case that $\mathcal{A}$ never queries the verification oracle and immediately stops when creating the forgery attempt. This loses a factor q_v in the success probability. However, since we consider general schemes we prefer to give the more general definition above.

3.2 Privacy

In the context of P-homomorphic signatures, such as redactable and homomorphic signatures, Boneh et al. [ABC+12] defined a strong privacy requirement, called context hiding. Basically it says that one cannot distinguish deduced signatures from fresh signatures, thus hiding for example which parts of the message have been redacted. The notion has later been refined in [ALP12, ALP13, DFF+13]. It implies several privacy notions in the context of redactable signature, for example [BBD+10]. Here, we more generally let the adversary decide upon two predicates $P_0, P_1 \in \mathcal{P}$, trying to decide which P_b has been used; since in our case of well-formed $\mathcal{P}$ we have $P_{\text{mac}} \in \mathcal{P}$ or $P_{\text{sig}} \in \mathcal{P}$, representing fresh MACs or signatures, this subsumes the context-hiding property.

The biggest difference of MACs to the case of signatures is that, in general, the challenge oracle, creating either fresh authentication data or assembling it out of the given one, cannot check the validity of the inputs $m_{\mathsf{in},i}, \sigma_{\mathsf{in},i}$, because it may lack knowledge of the corresponding verification key. Hence, our definition below allows for arbitrary, not necessarily valid inputs

$m_{\mathsf{in},i}, \sigma_{\mathsf{in},i}$. However, we also define a weaker notion for which these values must be valid according to the game.

For MACs it makes also sense to differentiate between insider and outsider privacy. Outsider privacy basically means that no one is able to distinguish tags generated through operations P_0, P_1 as long as the verification key id_{out} is oblivious to him. In other words, only an honest verifier is able to tell the function applied when generating the tag, and no one else. We call OSSs which are insider-private simply private.

Definition 3.3 (Privacy)**.** *An operational signature scheme* $\mathsf{OSS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Verify})$ *for well-formed predicates* $\mathcal{P}$ *(i.e., with* $P_{mac\text{-}or\text{-}sig} \in \mathcal{P}$*) over* $\mathcal{ID}_{pub} \subseteq \mathcal{ID}, \mathcal{M}$ *is* (t, q_k, q, ε)*-[weakly] private if for any algorithm* $\mathcal{A}$ *with runtime* t *and making at most* q_k *(resp.* q*) queries to his key-generating (resp. challenge oracle), the probability that the following experiment returns* 1 *is at most* $\frac{1}{2} + \varepsilon$*.*

Experiment $\mathbf{Exp}^{\textit{[weak] [outsider-]privacy}}_{\mathsf{OSS},\mathcal{A}}(\lambda)$

Set $\mathcal{QID} = \emptyset$.
$b \leftarrow \{0,1\}$
$(\mathsf{MSK}, \mathsf{PP}) \leftarrow \mathsf{Setup}(1^\lambda)$
$b^* \leftarrow \mathcal{A}^{\mathsf{KeyGen}', \mathsf{Ch}_b}(\mathsf{PP})$
Return 1 *iff* $b = b^*$

$\mathsf{KeyGen}', \mathsf{Verify}', \mathsf{Eval}'$ *as defined Definition 3.2*

If $\mathcal{A}$ *queries* $\mathsf{KeyGen}'(id)$,
add id *to* $\mathcal{QID}$*, and*
return $\mathsf{KeyGen}(\mathsf{MSK}, id)$

If $\mathcal{A}$ *queries* $\mathsf{Ch}_b((\mathbf{id^0}, \mathbf{m^0}, \sigma^0), (\mathbf{id^1}, \mathbf{m^1}, \sigma^1), P_0, P_1)$,
if $P_0((\mathbf{id^0}, \mathbf{m^0})) = 0$ *or* $P_1((\mathbf{id^1}, \mathbf{m^1})) = 0$ *return* $\perp$*;*
if $(k^0_{out}, m^0_{out}) \neq (k^1_{out}, m^1_{out})$ *return* $\perp$*;*

[weak*: if* $\mathsf{Verify}'((\mathbf{id^0}, \mathbf{m^0}, \sigma^0)) = 0$
or $\mathsf{Verify}'((\mathbf{id^1}, \mathbf{m^1}, \sigma^1)) = 0$ *return* $\perp$*;]*

[outsider*: if* $k^0_{out} = k^1_{out} \in \mathcal{QID}$*, return* $\perp$*;]*

compute $\sigma^0_{eval} \leftarrow \mathsf{Eval}'(\mathsf{PP}, (\mathbf{id^0}, \mathbf{m^0}, \sigma^0), P_0)$ *and*
$\sigma^1_{eval} \leftarrow \mathsf{Eval}'(\mathsf{PP}, (\mathbf{id^1}, \mathbf{m^1}, \sigma^1), P_1)$.
return $\perp$ *if* $\sigma^0_{eval} = \perp$ *or* $\sigma^1_{eval} = \perp$,
else return σ^b_{eval}.

The probability is taken over all coin tosses of algorithms Setup, KeyGen, Eval, *and* $\mathcal{A}$*. We let* $\boldsymbol{Adv}^{\textit{[w][o]priv}}_{\mathsf{OSS}}(t, q_k, q)$ *denote (a bound on) the value* ε *for which the scheme* OSS *is* (t, q_k, q, ε)*-[weakly] [outsider-]private. If* OSS *is* (t, q_k, q, ε)*-[weakly] [outsider-]private for time* t *and queries* q_k, q *polynomial in* λ *and* ε *is negligible in* λ*, then we simply say* OSS *is [weakly] [outsider-]private.*

Note that the context hiding definitions in [ABC+12, ALP12] actually ask for distributional equivalence of fresh signatures and derived signatures. As pointed out in [DFF+13]

this can be captured by considering unbounded $\mathcal{A}$ in the experiment above.

Capturing Re-randomizable Signatures. As an example, consider how our privacy notion captures re-randomizable signatures where, given a signature τ for a message m, one can publicly compute a signature τ' for the same message, such that τ' is distributed like a signature for m computed from scratch. To this end we add a predicate P_{rand} to $\mathcal{P}$, with $P_{\text{sig}} \in \mathcal{P}$, such that $P_{\text{rand}}((id_{\mathsf{in}}, m_{\mathsf{in}}), (id_{\mathsf{eval}}, m_{\mathsf{eval}}), (id_{\mathsf{out}}, m_{\mathsf{out}})) = 1$ if and only if $id_{\mathsf{in}} = id_{\mathsf{out}} = id_{\text{pub}}$, $id_{\mathsf{eval}} = id_{\epsilon}$, and $m_{\mathsf{in}} = m_{\mathsf{eval}} = m_{\mathsf{out}}$. Privacy guarantees that one cannot distinguish between signatures created via P_{sig}, i.e., through regular signing, and via P_{rand}, implying that anyone can derive quasi fresh signatures from given ones. If we now, for example, omit any bound on the adversary's running time t and demand $\epsilon = 0$, then we obtain perfectly re-randomizable signatures.

4 Operational Signatures from Indistinguishability Obfuscation

In this section we propose our construction of operational signature schemes from indistinguishability obfuscation. The basic idea of our construction is to build an obfuscated circuit that verifies that the user is allowed to derive the signature and, if this is the case, then outputs a signature on the derived message under a universal master key. The usage of strong obfuscators (namely for random oracles) is motivated by the fact that we aim for unforgeability *and* privacy in our construction.

We remark that if one only strives for unforgeability operational signatures can be constructed from signatures with the approach of Boyle et al. [BGI14]. See Appendix D for more details.

4.1 Indistinguishable Obfuscator for Random Oracles

Our definition of indistinguishability obfuscators follows the one from the literature [BGI+12], with the difference that the circuit get access to a random oracle H. A discussion about obfuscators in the context of random oracles has been given in the Introduction Section 1.3; here we focus on the technical aspects.

The difference to the original definition is that we now merely demand indistinguishability for the "non-oracle part", in the sense that we consider the output of the obfuscator to strip off all random oracle gates before obfuscating the circuit's "core": To this end we consider circuits C^H of the form $C^H(x) = C'(x, H(x))$ and run the obfuscator only on the C' part. We say that random-oracle based circuits of this kind have *upstream hashing only*. For such circuits it is understood that the obfuscator $i\mathcal{O}$ on input C^H outputs the obfuscation of circuit C' for input pairs of the form $(x, H(x))$.

Ideally, we would like to simply hand the obfuscated part over now to the distinguisher. Unfortunately, the obfuscation of the whole circuit, including the upstream hashing, also

prevents an adversary from jumping in at the computation after the hash check; this form of integrity of intermediate values is usually required to build secure obfuscators and is fundamental to the security of our operational signature scheme. Note that this property is achieved (again), once we instantiate the random oracle and run the obfuscator on the full circuit. Here we demand, abstractly, that the obfuscated circuit is defined only on inputs $(x, H(x))$ relative to the oracle H, and that queries outside of the domain would always return $\perp$ for the circuit.

Definition 4.1. *A uniform PPT machine $i\mathcal{O}$ is called an* oracle indistinguishability obfuscator *with respect to an oracle H and for a class $\{C_\lambda\}$ of oracle circuits with upstream hashing only, if the following conditions are satisfied:*

- ***Correctness:*** *For all security parameters $\lambda \in \mathcal{N}$, for all $C^H(\cdot) = C'(\cdot, H(\cdot)) \in C_\lambda$ with upstream hashing only, and for all inputs x, we have that*

$$\text{Prob}\big[\, O(x, H(x)) = C^H(x) : O \leftarrow i\mathcal{O}(\lambda, C') \text{ for } C^H(\cdot) = C'(\cdot, H(\cdot))\big] = 1$$

- ***Indistinguishability:*** *For any (not necessarily uniform) PPT adversaries Samp, D, there exists a negligible function μ such that the following holds: if* $\text{Prob}, C_0^H(x) = C_1^H(x) : (C_0^H, C_1^H, aux) \leftarrow Samp(1^\lambda) > 1 - \varepsilon(\lambda)$*, then we have:*

$$\begin{aligned}&\big|\text{Prob}\big[\, D^H(aux, i\mathcal{O}(\lambda, C_0')) = 1 : (C_0^H, C_1^H, aux) \leftarrow Samp^H(1^\lambda)\big] \\ &- \text{Prob}\big[\, D^H(aux, i\mathcal{O}(\lambda, C_1')) = 1 : (C_0^H, C_1^H, aux) \leftarrow Samp^H(1^\lambda)\big]\big| \leq \varepsilon(\lambda)\end{aligned}$$

where $C_b^H(x) = C_b'(x, H(x))$ for $b \in \{0, 1\}$.

4.2 Constructing OSS from $i\mathcal{O}$

To describe our OSS construction based on $i\mathcal{O}$formally, we fix the following notations and parameters. Recall that the idea is to build an obfuscated circuit which contains a master signing key, checks if the identifier-message-signature inputs are valid with respect to the predicate, and if so creates a signature for $m_{\mathsf{out}}||id_{\mathsf{out}}$ under the master secret. Technically, this requires that all identifiers $id_0, id_1 \in \mathcal{ID}_\lambda$ of the same security level have equal length. As explained in the introduction, we include another random-oracle based hash verification in order to extract forgeries from the adversaries. To ensure that only authenticated key holders can run the (publicly available) obfuscated circuit we use signature-based certificates as secrets attached to key identifiers. For simplicity we will use the same signature scheme as for the message-identifier pairs and think of these certificates as signatures for empty messages.

We cast our construction with a secret key for the evaluation key under identifier id_{eval}. But if this key is public, we can assume that any party can fetch this key from the key generation algorithm. Also recall that we assume that identifiers are always recoverable from actual keys.

$$
\begin{array}{l}
\underline{C_{H,\mathsf{sk},\mathsf{pk},\mathcal{P}}(((\mathbf{id}[\mathsf{k}_{\mathsf{eval}} \to \mathbf{id}_{\mathsf{eval}}], \mathbf{m}, \sigma), P), h) :} \\
\quad \text{If } h = H((\mathbf{id}[\mathsf{k}_{\mathsf{eval}} \to \mathbf{id}_{\mathsf{eval}}], \mathbf{m}, \sigma), P) \\
\qquad \wedge \quad \forall i : \ \mathsf{Verify}_{\mathsf{S}}(\mathsf{pk}, m_{\mathsf{in},i} || id_{\mathsf{in},i}, \sigma_i) = 1 \\
\qquad \wedge \quad \mathsf{Verify}_{\mathsf{S}}(\mathsf{pk}, \mathsf{k}_{\mathsf{eval}}, \sigma_{\mathsf{eval}}) = 1 \\
\qquad \wedge \quad P(\mathbf{id}, \mathbf{m}) = 1 \\
\qquad \text{output } \sigma_{\mathsf{out}} \leftarrow \mathsf{Sig}(\mathsf{sk}, m_{\mathsf{out}} || id_{\mathsf{out}}) \\
\quad \text{else} \\
\qquad \text{output } \perp
\end{array}
$$

Figure 1: Description of the obfuscated circuit $C_{H,\mathsf{sk},\mathsf{pk},\mathcal{P}}$.

Construction 4.2. *Given a signature scheme* $\mathsf{S} = (\mathsf{KGen}_{\mathsf{S}}, \mathsf{Sig}_{\mathsf{S}}, \mathsf{Verify}_{\mathsf{S}})$*, an indistinguishability obfuscator* $i\mathcal{O}$*, and a hash function* H*, we construct an operational signature scheme* $\mathsf{OSS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Verify})$ *as follows:*

$\mathsf{Setup}(1^\lambda)$**:** *Sample a key pair* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}_{\mathsf{S}}(1^\lambda)$ *and generate an obfuscation* $c \leftarrow i\mathcal{O}(C_{H,\mathsf{sk},\mathsf{pk}})$ *of circuit* $C_{H,\mathsf{sk},\mathsf{pk},\mathcal{P}}$ *described in Fig. 1. Set the public parameters as* $\mathsf{PP} = (c, \mathsf{pk})$ *and the master key as* $\mathsf{msk} = (\mathsf{sk}, \mathsf{PP})$*.*

$\mathsf{KeyGen}(\mathsf{msk}, id)$**:** *On input a master secret* $\mathsf{msk} = (\mathsf{sk}, c, \mathsf{pk})$ *and an identifier* $id \in \mathcal{ID}$*, return the pair* (id, σ_{id}) *where* $\sigma_{id} \leftarrow \mathsf{Sig}_{\mathsf{S}}(\mathsf{sk}, id)$ *is a signature over the identifier* id*.*

$\mathsf{Eval}(\mathsf{PP}, (\mathbf{id}[\mathsf{k}_{\mathsf{eval}} \to \mathbf{id}_{\mathsf{eval}}], \mathbf{m}, \sigma), P)$**:** *On input public parameters* $\mathsf{PP} = (c, \mathsf{pk})$*, a tuple* $(\mathbf{id}[\mathsf{k}_{eval} \to \mathbf{id}_{eval}], \mathbf{m}, \sigma)$ *consisting of triples* $(id_{in,i}, m_{in,i}, \sigma_{in,i})_{i=1}^{n-2}$*, the cryptographic key* $\mathsf{k}_{eval} = (\mathsf{pk}, id_{eval}, \sigma_{eval})$ *for evaluating message* m_{eval}*, the pair of target values* (id_{out}, m_{out})*, and a predicate* P*, compute first the digest* $h \leftarrow H((\mathbf{id}[\mathsf{k}_{eval} \to \mathbf{id}_{eval}], \mathbf{m}, \sigma), P)$ *and then return the signature* $\sigma \leftarrow c(((\mathbf{id}[\mathsf{k}_{eval} \to \mathbf{id}_{eval}], \mathbf{m}, \sigma), P), h)$*.*

$\mathsf{Verify}(\mathsf{k}_{id}, m, \sigma)$**:** *On input a key* $\mathsf{k}_{id} = (\mathsf{PP}, id, \sigma_{id})$*, where* $\mathsf{PP} = (c, \mathsf{pk})$ *a message* m *and a signature* σ*, return* $\mathsf{Verify}_{\mathsf{S}}(\mathsf{pk}, m || id, \sigma)$*.*

Theorem 4.3 (Correctness)**.** *Assume* $\mathsf{S} = (\mathsf{KGen}_{\mathsf{S}}, \mathsf{Sig}_{\mathsf{S}}, \mathsf{Verify}_{\mathsf{S}})$*,* $i\mathcal{O}$ *are correct, then the above operational signature scheme* OSS *is correct.*

Proof. Let $P \in \mathcal{P}_\lambda$ be an arbitrary predicate, $\mathbf{m} = (m_{\mathsf{in},1}, \ldots, m_{\mathsf{in},n-2}, m_{\mathsf{eval}}, m_{\mathsf{out}}) \in \mathcal{M}_\lambda^n$ be any tuple of messages, $\mathbf{id} = (id_{\mathsf{in},1}, \ldots, id_{\mathsf{in},n-2}, id_{\mathsf{eval}}, id_{\mathsf{out}}) \in \mathcal{ID}_\lambda^n$ be any tuple of key identifiers, $\sigma = (\sigma_1, \ldots, \sigma_{n-2}) \in \mathcal{S}_\lambda^{n-2}$ s.t. for all i in $[n-2]$ we have that $\mathsf{Verify}(\mathsf{k}_{id_{\mathsf{in},i}}, m_{\mathsf{in},i}, \sigma_i) = 1$, and let H be any function, e.g., a hash function.

$$\Pr\left[\begin{array}{l}\mathsf{Verify}(\mathsf{k}_{\mathsf{out}}, m_{\mathsf{out}}, \sigma_{\mathsf{out}}) = P(\mathbf{id}, \mathbf{m}) : \\ \\ \\ \end{array}\begin{array}{l}(\mathsf{PP}, \mathsf{MSK}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{k}_{\mathsf{id}})_{id \in \mathbf{id}} \leftarrow \mathsf{KeyGen}(\mathsf{MSK}, id)_{id \in \mathbf{id}} \\ \sigma_{\mathsf{out}} \leftarrow \mathsf{Eval}((\mathbf{id}[\mathsf{k}_{\mathsf{eval}} \rightarrow \mathbf{id}_{\mathsf{eval}}], \mathbf{m}, \sigma), P) \\ \\ \end{array}\right] \tag{1}$$

$$=\Pr\left[\begin{array}{l}\mathsf{Verify}(\mathsf{k}_{\mathsf{out}}, m_{\mathsf{out}}, \sigma_{\mathsf{out}}) = P(\mathbf{id}, \mathbf{m}) : \\ \\ \\ \end{array}\begin{array}{l}(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}_{\mathsf{S}}(1^\lambda) \\ c \leftarrow i\mathcal{O}(C_{H,\mathsf{sk},\mathsf{pk}}) \\ (\mathsf{k}_{\mathsf{id}})_{id \in \mathbf{id}} \leftarrow \mathsf{KeyGen}((\mathsf{sk}, \mathsf{pk}, c), id)_{id \in \mathbf{id}} \\ \sigma_{\mathsf{out}} \leftarrow \mathsf{Eval}((\mathbf{id}[\mathsf{k}_{\mathsf{eval}} \rightarrow \mathbf{id}_{\mathsf{eval}}], \mathbf{m}, \sigma), P) \\ \\ \end{array}\right] \tag{2}$$

$$=\Pr\left[\begin{array}{l}\mathsf{Verify}(\mathsf{k}_{\mathsf{out}}, m_{\mathsf{out}}, \sigma_{\mathsf{out}}) = P(\mathbf{id}, \mathbf{m}) : \\ \\ \\ \end{array}\begin{array}{l}(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}_{\mathsf{S}}(1^\lambda) \\ c \leftarrow i\mathcal{O}(C_{H,\mathsf{sk},\mathsf{pk}}) \\ (\sigma_{id})_{id \in \mathbf{id}} \leftarrow \mathsf{Sig}_{\mathsf{S}}(\mathsf{sk}, id)_{id \in \mathbf{id}} \\ \sigma_{\mathsf{out}} \leftarrow \mathsf{Eval}((\mathbf{id}[\mathsf{k}_{\mathsf{eval}} \rightarrow \mathbf{id}_{\mathsf{eval}}], \mathbf{m}, \sigma), P) \\ \\ \end{array}\right] \tag{3}$$

$$=\Pr\left[\begin{array}{l}\mathsf{Verify}(\mathsf{k}_{\mathsf{out}}, m_{\mathsf{out}}, \sigma_{\mathsf{out}}) = P(\mathbf{id}, \mathbf{m}) : \\ \\ \\ \\ \end{array}\begin{array}{l}(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}_{\mathsf{S}}(1^\lambda) \\ c \leftarrow i\mathcal{O}(C_{H,\mathsf{sk},\mathsf{pk}}) \\ (\sigma_{id})_{id \in \mathbf{id}} \leftarrow \mathsf{Sig}_{\mathsf{S}}(\mathsf{sk}, id)_{id \in \mathbf{id}} \\ h \leftarrow H((\mathbf{id}[\mathsf{k}_{\mathsf{eval}} \rightarrow \mathbf{id}_{\mathsf{eval}}], \mathbf{m}, \sigma), P) \\ \sigma_{\mathsf{out}} \leftarrow c((\mathbf{id}[\mathsf{k}_{\mathsf{eval}} \rightarrow \mathbf{id}_{\mathsf{eval}}], \mathbf{m}, \sigma), h) \\ \end{array}\right] \tag{4}$$

$$=\Pr\left[\begin{array}{l}\mathsf{Verify}(\mathsf{k}_{\mathsf{out}}, m_{\mathsf{out}}, \sigma_{\mathsf{out}}) = P(\mathbf{k}, \mathbf{m}) : \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array}\begin{array}{l}(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}_{\mathsf{S}}(1^\lambda) \\ c \leftarrow i\mathcal{O}(C_{H,\mathsf{sk},\mathsf{pk}}) \\ (\sigma_{id})_{id \in \mathbf{id}} \leftarrow \mathsf{Sig}_{\mathsf{S}}(\mathsf{sk}, id)_{id \in \mathbf{id}} \\ h \leftarrow H((\mathbf{id}[\mathsf{k}_{\mathsf{eval}} \rightarrow \mathbf{id}_{\mathsf{eval}}], \mathbf{m}, \sigma), P) \\ \text{If } h = H((\mathbf{id}[\mathsf{k}_{\mathsf{eval}} \rightarrow \mathbf{id}_{\mathsf{eval}}], \mathbf{m}, \sigma), P) \\ \wedge \forall i.\ \mathsf{Verify}_{\mathsf{S}}(\mathsf{pk}, m_{\mathsf{in},i} || id_{\mathsf{in},i}, \sigma_{\mathsf{in},i}) = 1 \\ \wedge \mathsf{Verify}_{\mathsf{S}}(\mathsf{pk}, id_{\mathsf{eval}}, \sigma_{\mathsf{eval}}) = 1 \\ \wedge P(\mathbf{id}, \mathbf{m}) = 1 \\ \text{then } \sigma_{\mathsf{out}} \leftarrow \mathsf{Sig}_{\mathsf{S}}(\mathsf{sk}, m_{\mathsf{out}} || id_{\mathsf{out}}) \\ \text{else } \sigma_{\mathsf{out}} \leftarrow \perp \end{array}\right] \tag{5}$$

$$=\Pr\left[\begin{array}{l}\mathsf{Verify}(\mathsf{k}_{\mathsf{out}}, m_{\mathsf{out}}, \sigma_{\mathsf{out}}) = P(\mathbf{id}, \mathbf{m}) : \\ \\ \\ \\ \end{array}\begin{array}{l}(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}_{\mathsf{S}}(1^\lambda) \\ \text{If } P(\mathbf{id}, \mathbf{m}) = 1 \\ \text{then } \sigma_{\mathsf{out}} \leftarrow \mathsf{Sig}_{\mathsf{S}}(\mathsf{sk}, m_{\mathsf{out}} || id_{\mathsf{out}}) \\ \text{else } \sigma_{\mathsf{out}} \leftarrow \perp \end{array}\right] \tag{6}$$

$$=\Pr\left[P(\mathbf{id}, \mathbf{m}) = P(\mathbf{id}, \mathbf{m})\right] = 1 \tag{7}$$

We start with the desired probability for the correctness from Definition 2.1. In the steps from (1) to (4), we replace Setup, KeyGen and Eval by their respective constructions. This does, of course, not modify the probability. In the step from (4) to (5), we replace the obfuscated circuit c by its non-obfuscated definition. By the definition of $i\mathcal{O}$, it holds that every circuit C, the obfuscated variant $c \leftarrow i\mathcal{O}(C)$ has the same functionality as C. Therefore, the replacement does not change the probability. Note that, by construction, $h = H((\mathbf{id}, \mathbf{m}, \sigma), P)$, so the check always succeeds as the random oracle gives the same answer if queried on the same inputs. Furthermore, by assumption it holds for every input tuple $(id_{\mathsf{in},i}, m_{\mathsf{in},i}, \sigma_i)$ that the

verification succeeds: $\mathsf{Verify}(\mathsf{k}_{id_{\mathsf{in},i}}, m_{\mathsf{in},i}, \sigma_i)$. Moreover, as KeyGen creates valid signatures σ_{id} for every key identifier id, the signature $\sigma_{\mathsf{k}_{\mathsf{eval}}}$ verifies because of the correctness property of the signature scheme S. We remove all these respective checks from (5) to (6) as they are always satisfied and we also remove all unused variables. By definition of Verify and the correctness of the underlying signature scheme, we know that all signatures $\sigma_{\mathsf{out}} \leftarrow \mathsf{Sig}(\mathsf{sk}, m_{\mathsf{out}}||id_{\mathsf{out}})$ verify. Consequently, the verification succeeds if and only if the predicate is satisfied and thus the probability collapses to 1.

This completes the proof for the correctness of OSS. □

4.3 Security Analysis

Regarding security, we prove the following theorem.

Theorem 4.4. *Let $\mathsf{S} = (\mathsf{KGen}_\mathsf{S}, \mathsf{Sig}_\mathsf{S}, \mathsf{Verify}_\mathsf{S})$ be an unforgeable and deterministic signature scheme, $i\mathcal{O}$ be a random-oracle based indistinguishability obfuscator for a class of upstream-hashing-only circuits $\mathcal{C}$ containing $C_{H,\mathsf{sk},\mathsf{pk},\mathcal{P}}$ and C_{fake} according to Definition 4.1, PRF is a pseudorandom function, and H be a hash function modeled as a random oracle. Then Construction 4.2 is an unforgeable and private OSS for all predicates $\mathcal{P}$ of fixed polynomial size, such that each $P \in \mathcal{P}$ is efficiently computable.*

We split the proof of this theorem in two parts by first showing unforgeability against chosen message attacks (Lemma 4.5) and then privacy (Lemma 4.6).

Lemma 4.5. *If $\mathsf{S} = (\mathsf{KGen}_\mathsf{S}, \mathsf{Sig}_\mathsf{S}, \mathsf{Verify}_\mathsf{S})$ is an unforgeable and deterministic signature scheme, $i\mathcal{O}$ a random oracle indistinguishability obfuscator for a class of circuits $\mathcal{C}$ containing $C_{H,\mathsf{sk},\mathsf{pk},\mathcal{P}}$ and C_{fake} (defined in Figure 2) according to Definition 4.1, PRF is a pseudorandom function, and H is a hash function modeled as a random oracle, then Construction 4.2 is an unforgeable OSS for all predicates $\mathcal{P}$ of polynomial size, such that each $P \in \mathcal{P}$ is efficiently computable.*

Intuition. We prove the lemma by reducing the unforgeability of the OSS construction to the unforgeability of the underlying signature scheme S. To do so, we show, via a transition of games, that an adversary $\mathcal{A}$ cannot distinguish whether it interacts with the challenger EUF-CMA$_{\mathsf{OSS}}$ or with another adversary $\mathcal{B}$ that simulates said challenger to break the unforgeability of S. First, we have $\mathcal{A}$ against the challenger for OSS. Next, we replace the random oracle by an oracle H that simulates a PRF and encodes signatures within its output, whenever the input x is a valid input for Eval that would lead to a signature y (see Figure 2). Since $\mathcal{A}$ does not know the key of the PRF, it cannot distinguish H from the random oracle. Next, we replace the game with a game in which our adversary $\mathcal{B}$ simulates the challenger for OSS. Since $\mathcal{B}$ does not have access to the secret key sk of S, it cannot generate the circuit $C_{H,\mathsf{sk},\mathsf{pk},\mathcal{P}}$. Instead, it creates a circuit C_{fake} (see Figure 2) that extracts signatures from the output of the oracle H. By definition of H, the two stripped-off versions of the circuits $C_{H,\mathsf{sk},\mathsf{pk},\mathcal{P}}$ and C_{fake} are functionally equivalent for inputs of the form $(x, H(x))$. Thus, we can apply the indistinguishability property of the indistinguishability obfuscator and see that $\mathcal{A}$ cannot distinguish the circuits.

$\underline{H(x):}$
if $x = ((\mathbf{id}[\mathsf{k}_{\mathsf{eval}} \to \mathbf{id}_{\mathsf{eval}}], \mathbf{m}, \sigma), P)$
and for all $i : \mathsf{Verify}_{\mathsf{S}}(\mathsf{pk}, m_{\mathsf{in},i}||id_{\mathsf{in},i}, \sigma_i) = 1$
and $\mathsf{Verify}_{\mathsf{S}}(\mathsf{pk}, \mathsf{k}_{\mathsf{eval}}, \sigma_{\mathsf{eval}}) = 1$
and $P(\mathbf{id}, \mathbf{m}) = 1$.
then // *extract forgery*
if there exists a tuple (id, m, σ)
in $(\mathbf{id}[\mathsf{k}_{\mathsf{eval}} \to \mathbf{id}_{\mathsf{eval}}], \mathbf{m}, \sigma)$ or $\mathsf{k}_{\mathsf{eval}} = (\mathsf{pk}, id_{\mathsf{eval}}, \sigma_{\mathsf{eval}}, c)$
such that $\mathsf{Verify}(id, m, \sigma) = 1$ and $(m||id) \notin Q$
then either output $(m||id, \sigma)$ or $(id_{\mathsf{eval}}, \sigma_{\mathsf{eval}})$ to $\mathcal{B}$
else
query $y \leftarrow \mathsf{Sig}(m_{\mathsf{out}}||id_{\mathsf{out}})$
return $H(x) := \mathsf{PRF}(\mathsf{k}_{\mathsf{PRF}}, x) \oplus y$
else return $\mathsf{PRF}(\mathsf{k}_{\mathsf{PRF}}, x)$

$\underline{C_{\text{fake}}(((\mathbf{id}[\mathsf{k}_{\mathsf{eval}} \to \mathbf{id}_{\mathsf{eval}}], \mathbf{m}, \sigma), P), h):}$
if $h = H((\mathbf{id}[\mathsf{k}_{\mathsf{eval}} \to \mathbf{id}_{\mathsf{eval}}], \mathbf{m}, \sigma), P)$
$\wedge$ forall i. $\mathsf{Verify}_{\mathsf{S}}(\mathsf{pk}, m_{\mathsf{in},i}||id_{\mathsf{in},i}, \sigma_i) = 1$
$\wedge$ $\mathsf{Verify}_{\mathsf{S}}(\mathsf{pk}, \mathsf{k}_{\mathsf{eval}}, \sigma_{\mathsf{eval}}) = 1$
$\wedge$ $P(\mathbf{id}, \mathbf{m}) = 1$
output $h \oplus \mathsf{PRF}(\mathsf{k}_{\mathsf{PRF}}, ((\mathbf{id}[\mathsf{k}_{\mathsf{eval}} \to \mathbf{id}_{\mathsf{eval}}], \mathbf{m}, \sigma), P))$
else
output $\perp$

Figure 2: Simulation of H and C_{fake}. The set Q denotes the set of messages which adversary $\mathcal{B}$ has queried its signing oracle about.

Moreover, since $C_{H,\mathsf{sk},\mathsf{pk},\mathcal{P}}$ does not contain the secret key $\mathsf{k}_{\mathsf{PRF}}$ of the PRF and C_{fake} does not contain the secret key sk of S, both values must be hidden within the obfuscated circuit. Consequently, $\mathcal{A}$ cannot use information from the obfuscated circuit $c \leftarrow i\mathcal{O}(C_{\text{fake}})$ to generate a forgery. Moreover, since $\mathcal{B}$ simulates the oracle H, it can see whenever the input to H contains a forgery. We then show that whenever $\mathcal{A}$ wins the game, $\mathcal{B}$ can also extract a forgery.

The full proof appears in Appendix B.

Lemma 4.6. *Construction 4.2 is private.*

Proof. By construction, all signatures σ for messages m under identifier id that are generated via Eval are deterministic signatures $\mathsf{Sig}(\mathsf{sk}, m||id)$ for a master signing key sk of the underlying signature scheme S. Consequently, for every computation that leads to signing any message m for an identifier id, the way in which this signature was created is completely hidden. More formally, let $\mathcal{A}$ be any machine. Whenever $\mathcal{A}$ queries $\mathsf{Ch}_b((\mathbf{id^0}, \mathbf{m^0}, \sigma^{\mathbf{0}}), (\mathbf{id^1}, \mathbf{m^1}, \sigma^{\mathbf{1}}), P_0, P_1)$, such that $(k^0_{\mathsf{out}}, m^0_{\mathsf{out}}) = (k^1_{\mathsf{out}}, m^1_{\mathsf{out}})$, then

$$\sigma^0_{\mathsf{eval}} = \mathsf{Eval}'((\mathbf{id^0}, \mathbf{m^0}, \sigma^{\mathbf{0}}), P_0) = \mathsf{Sig}(\mathsf{sk}, m_{\mathsf{out}}||id_{\mathsf{out}}) = \mathsf{Eval}'((\mathbf{id^1}, \mathbf{m^1}, \sigma^{\mathbf{1}}), P_1) = \sigma^1_{\mathsf{eval}}$$

This mathematical fact does not rely on any assumption on the strength of $\mathcal{A}$ or on any information about the other queries that $\mathcal{A}$ performs. As this challenge query is the only place in which Ch_b uses b, no information about b ever leaks to $\mathcal{A}$ and the construction above is private.

□

5 Blind Signatures from Operational Signatures

We show that general unforgeable and private operational signatures imply two-move blind signature schemes. Since blind signatures cannot be derived from black-box one-wayness in

general [KSY11], and since our construction and reduction are black-box, this entails that a generic black-box construction for operational signatures solely based on one-wayness is impossible as well. Furthermore, the result by Fischlin and Schröder [FS10] about the impossibility results for basing three-move blind signatures on non-interactive problems like RSA applies here in principle as well. They have additional stipulations on the blind signature scheme and the reductions, which must thus hold for the underlying operational signatures, too. Determining these properties is beyond our scope here.

We note that Boyle et al. [BGI14] actually do show that their notion of functional signatures can be met using one-way functions, if one forgoes privacy. They also show that unforgeable functional signatures, which are succinct but not necessarily private, imply SNARGs. Such SNARGs, on the other hand, are most likely not derivable from falsifiable assumptions like one-wayness [GW11]. This result hinges of the succinctness of the signatures, though. In this sense, our result supplements their result by showing that demanding privacy (instead of succinctness) still makes constructions based on one-wayness hard to find.

We also stress again that we do not claim to be able to subsume blind signatures under our definitional framework; this is impossible, because blind signatures have an interactive signature generation protocol underneath. Instead, we build a blind signature scheme from a private and unforgeable operational signature scheme.

5.1 Recap: Blind Signatures

We briefly recall the definition of blind signatures. For more information we refer to [Cha83]. A blind signature scheme $\mathcal{BS} = (\mathsf{BSGen}, \mathsf{S}, \mathsf{U}, \mathsf{BSVf})$ consists of the key generation algorithm BSGen which on input 1^λ returns a key pair $(\mathsf{sk}, \mathsf{pk})$, and of two interactive algorithms S and U which, when run interactively on inputs sk resp. (pk, m) allow the user U to output a signature σ such that this signature can be verified with $\mathsf{BSVf}(\mathsf{pk}, m, \sigma)$. As usual, for correctly generated data, verification should succeed. We assume that all admissible messages are of equal length (otherwise, an additional step is added in which the messages are hashed). Both of the aforementioned impossibility results hold for equal-length messages. Without loss of generality, we assume for convenience below that each message starts with a '1' bit. This can always be achieved by prepending this redundant bit to each message.

Unforgeability of blind signatures imposes two requirements: First, any efficient malicious user U^* must not be able to generate $\ell + 1$ valid signatures for distinct messages $m_1, \ldots, m_{\ell+1}$ after at most ℓ interactions with the honest signer S. Since we will derive a two-move signature scheme we simply count *initiated* interactions here instead of completed ones [FS12]. Second, to achieve the notion of blindness, an efficient malicious signer S^* must not be able to tell apart the following two scenarios: S^* selects the key pair and a pair of messages m_0, m_1; the user randomly selects the order of these two messages, corresponding to the two scenarios, and uses them in that order in two interactions with the malicious signer. Since our scheme will consist of two moves only, the two executions can be carried out concurrently. We note that, in case one of the two user instances fails to compute a valid signature σ_0 or σ_1 in one of the two executions, then the signer only receives $(\perp, \perp)$ for both executions.

Our blind signature scheme will achieve only "honest-key blindness" in the sense defined in [JLO97], i.e., blindness only needs to hold with respect to honestly generated key pairs for which the adversary against the blindness property learns the genuine secret key. This is a reminiscence of our model in which KeyGen honestly generates keys for users. If we adopt the stronger model where the adversary can choose keys for corrupted users, then our scheme has this property as well and is blind with respect to maliciously chosen keys. The aforementioned impossibility results for blind signatures hold in the honest-key case as well, such that we can still conclude that deriving an unforgeable and private OSS from one-wayness is hard.

5.2 Constructing Blind Signatures from OSS

For the construction we need a two-move commitment scheme $\mathcal{C} = (\mathsf{CGen}, \mathsf{Com})$ where secrecy guarantees that, even for a maliciously generated key $\mathrm{pk}_{\mathcal{C}} \leftarrow \mathsf{CGen}(1^\lambda)$, the committed message m is hidden by the commitment $C = \mathsf{Com}(\mathrm{pk}_{\mathcal{C}}, m; r)$ for randomness r (defined through the usual indistinguishability game). Unambiguity says that a malicious committer, even an unbounded one, cannot find $(m, r), (m', r')$ with $m \neq m'$ but $\mathsf{Com}(\mathrm{pk}_{\mathcal{C}}, m; r) = \mathsf{Com}(\mathrm{pk}_{\mathcal{C}}, m'; r')$ for honestly chosen $\mathrm{pk}_{\mathcal{C}}$. This should hold with overwhelming probability over the choice of $\mathrm{pk}_{\mathcal{C}}$. We say that the scheme is secure if it has the two properties. We assume that any key $\mathrm{pk}_{\mathcal{C}}$ has a fixed length, depending only on λ; this can always be achieved by padding the keys. Such commitments can be based on any one-way functions, and if we assume (as we will below) that the operational message authenticator includes a regular signature or MAC scheme, then we can thus build such commitments without any further assumption.

The construction idea is to have the user in the blind signature ask the signer to sign the required messages m_{in} under its key such that the user can then locally derive the signature for m_{out} with $\mathsf{pk}_{id_{\mathsf{eval}}}$ (which we assume to be a public value). We will then craft an appropriate predicate P which ensures that unforgeability of the blind signature scheme basically follows from the unforgeability of the operational signatures. Blindness will follow from the privacy of the underlying operational scheme and because we let the user have the signer sign a commitment of m_{in} instead, and ensure that P is still compatible with this commitment.

Construction 5.1. *Let $\mathcal{C} = (\mathsf{CGen}, \mathsf{Com})$ be a two-move commitment scheme. Let $\mathsf{OSS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Verify})$ be an operational signature scheme for predicate $\mathcal{P}$ over $\mathcal{ID}_{pub} \subseteq \mathcal{ID}, \mathcal{M}$, where $\mathcal{P}$ only contains P_{sig} and the following predicate P:*

$$P((id_{in}, m_{in}), (id_{eval}, m_{eval}), (id_{out}, m_{out})) = 1 \iff \begin{cases} m_{eval} = (r, m_{out}), \\ id_{out} = id_{in} = id_{pub}, id_{eval} = id_\epsilon \\ m_{in} = 0 || pk_{\mathcal{C}} || \mathsf{Com}(pk_{\mathcal{C}}, m_{out}; r) \end{cases}$$

Note that $id_{eval} = id_\epsilon$ corresponds to public information, and that we have $id_{out} = id_{in} = id_{pub}$, i.e., transform signatures into signatures.

Define the following blind signature scheme $\mathcal{BS} = (\mathsf{BSGen}, \mathsf{S}, \mathsf{U}, \mathsf{BSVf})$:

Key generation: *Run* $\mathsf{Setup}(1^\lambda)$ *to generate* $(\mathsf{MSK}, \mathsf{PP})$. *Also, sample a signing key pair* $(\mathsf{sk}_{id_{sig}}, \mathsf{pk}_{id_{pub}})$ *via* MSK, *as well as* $pk_{\mathcal{C}} \leftarrow \mathsf{CGen}(1^\lambda)$. *Output* $\mathsf{sk}_{id_{sig}}$ *as the secret key, and* $(\mathsf{PP}, \mathsf{pk}_{id_{pub}}, pk_{\mathcal{C}})$ *as the public key.*

Interactive Signing: *In the interactive signing protocol let the user for input* $(\mathsf{PP}, \mathsf{pk}_{id_{pub}}, pk_{\mathcal{C}})$ *and* m *first pick randomness* r *for the commitment. It then computes* $C = \mathsf{Com}(pk_{\mathcal{C}}, m; r)$ *and sends* C *to the signer. The signer computes (via* P_{sig}*) a signature for* $0||pk_{\mathcal{C}}||C$ *under key* $\mathsf{pk}_{id_{pub}}$ *and returns this signature* σ_C *to the user. The user checks the validity of the signature (and aborts if it cannot be verified); else, it runs* Eval *for input* $(id_{pub}, 0||pk_{\mathcal{C}}||C, (id_\epsilon, (r, m)), (id_{pub}, m))$ *and* P *to derive a signature* σ_m *for* m. *It outputs this signature.*

Verification: *Simply run* Verify *for* $\mathsf{pk}_{id_{pub}}$ *on* m *and* σ, *and check that* m *starts with a bit* $'1'$.

Theorem 5.2. *If the commitment scheme* $\mathcal{C}$ *is secure and the operational signature scheme* OSS *is unforgeable, then the blind signature scheme in Construction 5.1 is secure.*

Proof. We need to show that the above scheme satisfies blindness and unforgeability.

Blindness. Here we only sketch the blindness proof; the full proof can be found in Appendix C. Basically, the idea is that we can make the view of the signer completely independent of the bit b it tries to predict. This happens in two steps: First, instead of giving the signer the correct blind signatures σ_0, σ_1 for the messages m_0, m_1, which have been signed in random order by forwarding commitments C_0, C_1 to the signer to receive intermediate signatures, we instead forward signatures σ'_0, σ'_1 which have been derived in another execution branch, using the same keys, randomness, and messages m_0, m_1 on the signer's side but independent commitments C'_0, C'_1 and an independent bit b'. Privacy of the OSS, saying that one cannot distinguish signatures derived from two possible sources, guarantees here that the malicious signer would still be able to predict b successfully from C_0, C_1 and the "wrong" signatures σ'_0, σ'_1.[4]

In the next step we replace the commitments C_0, C_1 in the main branch by commitments to all-zero strings, and still provide the signatures σ'_0, σ'_1 derived in the other branch. The latter means that we do not need to be able to open the commitments in the main branch. By the hiding property of the commitment scheme, it follows that this is again indistinguishable from the signer's viewpoint. But now the signer only receives commitments to zero strings, and signatures derived in an execution for an independent bit b'. It thus cannot predict the unknown bit b better than by guessing.

Unforgeability. To prove unforgeability we turn the adversary $\mathcal{B}$ against the unforgeability of the blind signature scheme into an adversary $\mathcal{A}$ against the unforgeability of OSS.

[4]The main part of the argument is to show that this is indeed true, because we have to take care of possible aborts in the second branch.

Initially, $\mathcal{A}$ receives PP and $\mathsf{pk}_{id_{\text{pub}}}$. It will generate $\text{pk}_{\mathcal{C}} \leftarrow \mathsf{CGen}(1^\lambda)$ and start an execution of $\mathcal{B}$ on $(\mathsf{PP}, \mathsf{pk}_{id_{\text{pub}}}, \text{pk}_{\mathcal{C}})$. Whenever $\mathcal{B}$ ask for a signature for C in an interaction with the signer, adversary $\mathcal{A}$ calls its Eval' oracle on $(id_{\text{pub}}, 0||\text{pk}_{\mathcal{C}}||C), (id_{\text{pub}}, 0||\text{pk}_{\mathcal{C}}||C)$ and P_{sig} to create a signature for $0||\text{pk}_{\mathcal{C}}||C$. Forward this signature to $\mathcal{B}$. Note that $\mathcal{A}$ does not need to make any queries to KeyGen, nor to Verify' at this point. When $\mathcal{B}$ eventually outputs $\ell + 1$ pairs $(m_1, \sigma_1), \ldots, (m_{\ell+1}, \sigma_{\ell+1})$ adversary $\mathcal{A}$ calls the verification oracle Verify' about all these pairs and id_{pub}.

To assess $\mathcal{A}$'s success probability we need to determine the set of trivial pairs. Clearly, any signed commitment C adds an entry $(id_{\text{pub}}, 0||\text{pk}_{\mathcal{C}}||C)$ to this set. As for the recursive elements, note that $P_{\text{sig}} \in \mathcal{P}$ does not add further elements from those derivable from $(id_{\text{pub}}, 0||\text{pk}_{\mathcal{C}}||C)$. Varying over all $m_{\mathsf{eval}} = (r, m_{\mathsf{out}})$, the function f can only add one pair $(id_{\text{pub}}, m_{\mathsf{out}})$ to the trivial set for a tuple $(id_{\text{pub}}, 0||\text{pk}_{\mathcal{C}}||C)$, because for the honestly generated key $\text{pk}_{\mathcal{C}}$, included in $0||\text{pk}_{\mathcal{C}}||C$, there is at most one valid message opening m_{out} to C. This holds with overwhelming probability over the choice of $\text{pk}_{\mathcal{C}}$ such that we can simply assume perfect unambiguity and lose only a negligible term in the success probability. (Note that there may be multiple matching random strings r, but this is irrelevant.) Furthermore, m_{out} must start with bit '1' and cannot recursively add further entries via P.

In summary, the trivial set only contains entries $(id_{\text{pub}}, 0||\text{pk}_{\mathcal{C}}||C)$ for queries C made by $\mathcal{B}$, and one entry of the form (id_{pub}, m) for each such tuple, where m starts with a bit '1'. Hence, if $\mathcal{B}$ succeeds and produces $\ell + 1$ valid signatures for distinct messages after ℓ interactions, then $\mathcal{A}$ creates $\ell + 1$ valid signatures for distinct messages m (with '1' prepended), even though the trivial set only contains ℓ such messages. It follows that any successful $\mathcal{B}$ would yield a contradiction against the unforgeability of OSS. □

Note that the counterexample requires $m_{\mathsf{eval}} \neq m_{\mathsf{out}}$. We can easily modify the scheme to enforce $m_{\mathsf{eval}} = m_{\mathsf{out}}$ by considering predicates P_r (over all possible random strings r) in $\mathcal{P}$ instead.

It is worthwhile to note that we were unable to build other "heavy" primitives like key agreement, or even functional encryption out of operational signatures. We leave this as an open problem.

References

[ABC+12] Jae Hyun Ahn, Dan Boneh, Jan Camenisch, Susan Hohenberger, abhi shelat, and Brent Waters. Computing on authenticated data. In Ronald Cramer, editor, *Theory of Cryptography*, volume 7194 of *Lecture Notes in Computer Science*, pages 1–20. Springer Berlin Heidelberg, 2012.

[ABF+13] Joël Alwen, Manuel Barbosa, Pooya Farshim, Rosario Gennaro, S. Dov Gordon, Stefano Tessaro, and David A. Wilson. On the relationship between functional encryption, obfuscation, and fully homomorphic encryption. In *Cryptography and Coding - 14th IMA International Conference, IMACC 2013, Oxford, UK, December 17-19, 2013. Proceedings*, pages 65–84, 2013.

[ACLY00] R. Ahlswede, Ning Cai, S.-Y.R. Li, and R.W. Yeung. Network information flow. *IEEE Transactions on Information Theory*, 46(4):1204–1216, 2000.

[ACMT05] Giuseppe Ateniese, Daniel H. Chou, Breno Medeiros, and Gene Tsudik. Sanitizable signatures. In Sabrinade Capitani Vimercati, Paul Syverson, and Dieter Gollmann, editors, *Computer Security ESORICS 2005*, volume 3679 of *Lecture Notes in Computer Science*, pages 159–177. Springer Berlin Heidelberg, 2005.

[ALP12] Nuttapong Attrapadung, Benot Libert, and Thomas Peters. Computing on authenticated data: New privacy definitions and constructions. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 367–385. Springer Berlin Heidelberg, 2012.

[ALP13] Nuttapong Attrapadung, Benoit Libert, and Thomas Peters. Efficient completely context-hiding quotable and linearly homomorphic signatures. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *Public-Key Cryptography - PKC 2013*, volume 7778 of *Lecture Notes in Computer Science*, pages 386–404. Springer Berlin Heidelberg, 2013.

[BBD+10] Christina Brzuska, Heike Busch, Özgür Dagdelen, Marc Fischlin, Martin Franz, Stefan Katzenbeisser, Mark Manulis, Cristina Onete, Andreas Peter, Bertram Poettering, and Dominique Schröder. Redactable signatures for tree-structured data: Definitions and constructions. In Jianying Zhou and Moti Yung, editors, *Applied Cryptography and Network Security*, volume 6123 of *Lecture Notes in Computer Science*, pages 87–104. Springer Berlin Heidelberg, 2010.

[Bel06] Mihir Bellare. New proofs for nmac and hmac: Security without collision-resistance. In *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 602–619. Springer, 2006.

[BF11] Dan Boneh and David Mandell Freeman. Homomorphic signatures for polynomial functions. In Kenneth G. Paterson, editor, *Advances in Cryptology EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 149–168. Springer Berlin Heidelberg, 2011.

[BF14] Mihir Bellare and Georg Fuchsbauer. Policy-based signatures. In *Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings*, pages 520–537, 2014.

[BFF+09] Christina Brzuska, Marc Fischlin, Tobias Freudenreich, Anja Lehmann, Marcus Page, Jakob Schelbert, Dominique Schröder, and Florian Volk. Security of sanitizable signatures revisited. In Stanislaw Jarecki and Gene Tsudik, editors, *Public Key Cryptography PKC 2009*, volume 5443 of *Lecture Notes in Computer Science*, pages 317–336. Springer Berlin Heidelberg, 2009.

[BFKW09] Dan Boneh, David Freeman, Jonathan Katz, and Brent Waters. Signing a linear subspace: Signature schemes for network coding. In Stanislaw Jarecki and Gene Tsudik, editors, *Public Key Cryptography PKC 2009*, volume 5443 of *Lecture Notes in Computer Science*, pages 68–87. Springer Berlin Heidelberg, 2009.

[BFLS10] Christina Brzuska, Marc Fischlin, Anja Lehmann, and Dominique Schröder. Unlinkability of sanitizable signatures. In *Public Key Cryptography*, volume 6056 of *Lecture Notes in Computer Science*, pages 444–461. Springer, 2010.

[BGI+12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6, 2012.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings*, pages 501–519, 2014.

[BGLS03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, pages 416–432, 2003.

[BKM09] Adam Bender, Jonathan Katz, and Ruggero Morselli. Ring signatures: Stronger definitions, and constructions without random oracles. *J. Cryptology*, 22(1):114–138, 2009.

[BMS13] Michael Backes, Sebastian Meiser, and Dominique Schröder. Delegatable functional signatures. Cryptology ePrint Archive, Report 2013/408, 2013. `http://eprint.iacr.org/`.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993.*, pages 62–73. ACM, 1993.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Yuval Ishai, editor, *Theory of Cryptography*, volume 6597 of *Lecture Notes in Computer Science*, pages 253–273. Springer Berlin Heidelberg, 2011.

[CF13] Dario Catalano and Dario Fiore. Practical homomorphic macs for arithmetic circuits. In Thomas Johansson and PhongQ. Nguyen, editors, *Advances in Cryptol-*

ogy – EUROCRYPT 2013, volume 7881 of *Lecture Notes in Computer Science*, pages 336–352. Springer Berlin Heidelberg, 2013.

[CFGN14] Dario Catalano, Dario Fiore, Rosario Gennaro, and Luca Nizzardo. Generalizing homomorphic macs for arithmetic circuits. In *Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings*, volume 8383 of *Lecture Notes in Computer Science*, pages 538–555. Springer, 2014.

[CFW14] Dario Catalano, Dario Fiore, and Bogdan Warinschi. Homomorphic signatures with efficient verification for polynomial functions. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 371–389. Springer, 2014.

[CH91] David Chaum and Eugène Heyst. Group signatures. In DonaldW. Davies, editor, *Advances in Cryptology — EUROCRYPT '91*, volume 547 of *Lecture Notes in Computer Science*, pages 257–265. Springer Berlin Heidelberg, 1991.

[Cha83] David Chaum. Blind signatures for untraceable payments. In David Chaum, RonaldL. Rivest, and AlanT. Sherman, editors, *Advances in Cryptology*, pages 199–203. Springer US, 1983.

[CKLM13] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Malleable signatures: Complex unary transformations and delegatable anonymous credentials. In Anupam Datta and Cedric Fournet, editors, *IEEE Computer Security Foundations Symposium*, Lecture Notes in Computer Science. IEEE Computer Society, 2013.

[CLX09] Ee-Chien Chang, CheeLiang Lim, and Jia Xu. Short redactable signatures using random trees. In Marc Fischlin, editor, *Topics in Cryptology CT-RSA 2009*, volume 5473 of *Lecture Notes in Computer Science*, pages 133–147. Springer Berlin Heidelberg, 2009.

[DFF+13] Björn Deiseroth, Victoria Fehr, Marc Fischlin, Manuel Maasz, Nils Fabian Reimers, and Richard Stein. Computing on authenticated data for adjustable predicates. In *ACNS*, pages 53–68, 2013.

[FS10] Marc Fischlin and Dominique Schröder. On the impossibility of three-move blind signature schemes. In Henri Gilbert, editor, *Advances in Cryptology EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 197–215. Springer Berlin Heidelberg, 2010.

[FS12] Marc Fischlin and Dominique Schröder. Security of blind signatures under aborts and applications to adaptive oblivious transfer. *J. Mathematical Cryptology*, 5(2):169–204, 2012.

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 40–49. IEEE Computer Society, 2013.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *Proceedings of the 43rd annual ACM symposium on Theory of computing*, STOC '11, pages 99–108, New York, NY, USA, 2011. ACM.

[GW12] Rosario Gennaro and Daniel Wichs. Fully homomorphic message authenticators. Cryptology ePrint Archive, Report 2012/290, 2012.

[GW13] Rosario Gennaro and Daniel Wichs. Fully homomorphic message authenticators. In *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part II*, volume 8270 of *Lecture Notes in Computer Science*, pages 301–320. Springer, 2013.

[HKW14] Susan Hohenberger, Venkata Koppula, and Brent Waters. Universal signature aggregators. Cryptology ePrint Archive, Report 2014/745, 2014. `http://eprint.iacr.org/`.

[JLO97] Ari Juels, Michael Luby, and Rafail Ostrovsky. Security of blind digital signatures. In S. Burton and Jr. Kaliski, editors, *Advances in Cryptology CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 150–164. Springer Berlin Heidelberg, 1997.

[KN08] Eike Kiltz and Gregory Neven. *Identity-Based Signatures*. Cryptology and Information Security Series on Identity-Based Cryptography. IOS Press, 2008.

[KSY11] Jonathan Katz, Dominique Schröder, and Arkady Yerukhimovich. Impossibility of blind signatures from one-way permutations. In Yuval Ishai, editor, *Theory of Cryptography*, volume 6597 of *Lecture Notes in Computer Science*, pages 615–629. Springer Berlin Heidelberg, 2011.

[LYC03] S.-Y.R. Li, R.W. Yeung, and Ning Cai. Linear network coding. *IEEE Transactions on Information Theory*, 49(2):371–381, 2003.

[MPR11] Hemanta K. Maji, Manoj Prabhakaran, and Mike Rosulek. Attribute-based signatures. In Aggelos Kiayias, editor, *Topics in Cryptology CT-RSA 2011*, volume 6558 of *Lecture Notes in Computer Science*, pages 376–392. Springer Berlin Heidelberg, 2011.

[PST14] Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 500–517. Springer, 2014.

[RST01a] Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In *ASIACRYPT*, volume 2248 of *Lecture Notes in Computer Science*, pages 552–565. Springer, 2001.

[RST01b] Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In Colin Boyd, editor, *Advances in Cryptology — ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 552–565. Springer Berlin Heidelberg, 2001.

[SPB+12] Kai Samelin, Henrich C. Pöhls, Arne Bilzhause, Joachim Posegga, and Hermann Meer. Redactable signatures for independent removal of structure and content. In MarkD. Ryan, Ben Smyth, and Guilin Wang, editors, *Information Security Practice and Experience*, volume 7232 of *Lecture Notes in Computer Science*, pages 17–33. Springer Berlin Heidelberg, 2012.

[SR10] Daniel Slamanig and Stefan Rass. Generalizations and extensions of redactable signatures with applications to electronic healthcare. In Bart Decker and Ingrid Schaumller-Bichl, editors, *Communications and Multimedia Security*, volume 6109 of *Lecture Notes in Computer Science*, pages 201–213. Springer Berlin Heidelberg, 2010.

[SW05] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In Ronald Cramer, editor, *Advances in Cryptology EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 457–473. Springer Berlin Heidelberg, 2005.

A Expressing Special Operational Schemes in the Framework

In this section we instantiate the predicate P for a various prominent authentication primitives to show how the notion captures such schemes. We discuss here signatures only; one may easily derive the functionality for the symmetric counterpart where designated verifiers validate the authenticity of a message by adhering the convention as put forth in Definition 2.2.

A.1 Malleable Signatures: The Case of Sanitizable and Redactable Signatures

Sanitizable signatures [ACMT05, BFF+09] allow the signer to grant a designated party, called the sanitizer, the privilege of message modification without invalidating the authenticity of the unmodified part. The signer holds a signing key for id_{sig} for authentication of messages m, and the sanitizer has a secret key k_{san} for controlled sanitization, which is formally expressed in terms of a polynomial-time predicate $p_{\text{san}} : \mathcal{ID} \times \mathcal{M} \times \mathcal{M} \to \{0,1\}$. Signatures are verified with the help of the public key for id_{pub}. Thus, in a sanitizable signature scheme the key space comprises the key identifiers $\mathcal{ID} = \{id_{\text{sig}}, id_{\text{pub}}, k_{\text{san}}, id_\epsilon\}$ such that $\mathcal{ID}_{\text{pub}} = \{id_{\text{pub}}, id_\epsilon\}$ and the functionality $\mathcal{P} = \{P_{\text{sig}}, P_{\text{san}}\}$ contains the sanitizing predicate $P_{\text{san}} : (\mathcal{ID} \times \mathcal{M})^3 \to \{0,1\}$ defined as:

$$P_{\text{san}}(\mathbf{id}, \mathbf{m}) = \begin{cases} 1 & \text{if } id_{\mathsf{in}} = id_{\mathsf{out}} = id_{\text{pub}},\ id_{\mathsf{eval}} = k_{\text{san}},\ p_{\text{san}}(k_{\text{san}}, m_{\mathsf{eval}}, m_{\mathsf{in}}) = 1,\ m_{\mathsf{eval}} = m_{\mathsf{out}} \\ 0 & \text{otherwise} \end{cases}.$$

In words, if one receives a message m_{in} with a signature as input which verifies under id_{pub}, and one applies the sanitization key for k_{san} to derive a signature for m_{eval}, the sanitized version of m_{in}, then this signature verifies for $m_{\mathsf{out}} = m_{\mathsf{eval}}$ under id_{pub} again. In addition, P_{sig} realizes the usual behavior of a standard digital signature, namely if one applies id_{sig} to a message m_{eval} and verifies this signature for $m_{\mathsf{out}} = m_{\mathsf{eval}}$ under $\mathsf{pk}_{id_{\text{pub}}}$. Note also that one may refine the sanitization predicate and demand that k_{san} is specific to the sanitizing operation, e.g., that only leading 0-bits can be removed.

Redactable signatures [CLX09, BBD+10, SR10, SPB+12] are special sanitizable signatures with "public sanitization". Given a message and a valid signature over the message, any party may delete parts of the message and infer a valid signature. In this context, an OSS is cast over the key space $\mathcal{ID} = \{id_{\text{sig}}, id_{\text{pub}}, id_\epsilon\}$ such that $\mathcal{ID}_{\text{pub}} = \{id_{\text{pub}}, id_\epsilon\}$ and the predicates $\mathcal{P} = \{P_{\text{sig}}, P_{\text{red}}\}$ where the redaction prdicate $P_{\text{red}} : (\mathcal{ID} \times \mathcal{M})^3 \to \{0,1\}$ is defined as:

$$P_{\text{red}}(\mathbf{id}, \mathbf{m}) : \begin{cases} 1 & \text{if } id_{\mathsf{in}} = id_{\mathsf{out}} = id_{\text{pub}},\ id_{\mathsf{eval}} = id_\epsilon,\ p_{\text{red}}(id_{\text{pub}}, m_{\mathsf{eval}}, m_{\mathsf{in}}) = 1,\ m_{\mathsf{eval}} = m_{\mathsf{out}} \\ 0 & \text{otherwise} \end{cases}.$$

Predicate P_{red} is identical to P_{san} with the exception that the evaluation key is public. More precisely, predicate P_{san} checks that message m_{eval} can be derived from m_{in} by redaction, expressed in form of predicate $p_{\text{red}} : \mathcal{ID} \times \mathcal{M} \times \mathcal{M} \to \{0,1\}$ that outputs 1 if and only if m_{eval} is a redacted version of m_{in}, that $id_{\mathsf{in}} = id_{\mathsf{out}} = id_{\text{pub}}$ and $m_{\mathsf{eval}} = m_{\mathsf{out}}$ (i.e., the input message verifies under key identifier id_{pub}, the output message $m_{\mathsf{eval}} = m_{\mathsf{out}}$, too), and that $id_{\mathsf{eval}} = id_\epsilon$ for the trivial (public) information. This way, redaction becomes a public operation.

A.2 Homomorphic Signatures: The Case of Network Coding and P-Homomorphic Signatures

In line with tremendous breakthroughs in computing on encrypted data, significant steps towards computing on authenticated data have been made [ACLY00, LYC03, BFKW09, ABC+12, BF11, GW12]. In a homomorphic signature scheme a third party derives from message-signature pairs $(m_i, \sigma_i)_{1 \leq i \leq N}$ a valid signature σ' over message $m' = f(m_1, \ldots, m_N)$. Ahn et al. [ABC+12] present a framework to compute on authenticated data where a third party may compute a derived signature as long as there is some relation (expressed in terms of a predicate $P : \mathcal{M}^{N+1} \to \{0,1\}$) between messages $m_1, \cdots, m_N$ and m'. They call signatures of that type P-homomorphic.

Ahn et al. [ABC+12] show that many related signatures are expressible in their P-homomorphic notion including signatures for linear functions (aka. signatures for network coding) [ACLY00, LYC03] and polynomial functions [BF11]. One may consider P-homomorphic signatures as an OSS defined over the identifier space $\mathcal{ID} = \{id_{\text{sig}}, id_{\text{pub}}, id_\epsilon\}$ such that $id_{\text{pub}}, id_\epsilon \in \mathcal{ID}_{\text{pub}}$ and the predicate set $\mathcal{P} = \{P_{\text{sig}}, P_{\text{P-hom}}\}$ where $P_{\text{P-hom}}$ is defined as

$$P_{\text{P-hom}}(\mathbf{id}, \mathbf{m}) = \begin{cases} 1 & \text{if } id_{\mathsf{in},i} = id_{\text{pub}}\ \forall i, id_{\mathsf{out}} = id_{\text{pub}},\ id_{\mathsf{eval}} = id_\epsilon, \\ & \mathsf{P}(m_{\mathsf{in},1}, \cdots, m_{\mathsf{in},N}, m_{\mathsf{eval}}) = 1,\ m_{\mathsf{eval}} = m_{\mathsf{out}} \\ 0 & \text{otherwise} \end{cases}.$$

In other words, predicate $P_{\text{P-hom}}$ assures that messages $M = \{m_{\mathsf{in},i}\}_i$ and $m_{\mathsf{eval}} = m_{\mathsf{out}}$ satisfy the relation $\mathsf{P}(M, m_{\mathsf{eval}}) = 1$ of the P-homomorphic signature and that all keys satisfy $id_{\mathsf{in},i}, id_{\mathsf{out}} = id_{\text{pub}}$ and $id_{\mathsf{eval}} = id_\epsilon$.

A.3 Indexed Signatures: The Case of Identity-Based and Attribute-based Signatures

In some cases, it is useful to add additional structure to the message space. In many applications the plaintext $m \in \mathcal{M}$ itself is a pair $(\mathsf{ind}, m') \in \mathcal{I} \times \mathcal{M}'$ where ind is called the index (or preamble) and m' is called the payload (or simply message). For example, the index may be a string or set thereof. This additional structure already suffices to express identity-based signatures [KN08] and more general derivatives like attribute-based signatures [MPR11]. In an index-based signature scheme, parties have signing keys associated with their index and they are eligible to sign a message, if key and index are related in terms of some polynomial-time predicate $p : \mathcal{ID} \times \mathcal{I} \to \{0,1\}$. In this context, we instantiate the OSS with identifier space $\mathcal{ID} = \{k_1, \ldots, k_{|\mathcal{I}|}, id_{\text{pub}}, id_\epsilon\}$ such that $id_{\text{pub}}, id_\epsilon \in \mathcal{ID}_{\text{pub}}$ and predicates $\mathcal{P} = \{P_{\text{sig}}, P_{\text{ind}}\}$ where we define P_{ind} as:

$$P_{\text{ind}}(\mathbf{id}, \mathbf{m}) = \begin{cases} 1 & \text{if } id_{\mathsf{in}} = id_{\mathsf{out}} = id_{\text{pub}},\ id_{\mathsf{eval}} = k_{\text{ind}},\ p(k_{\text{ind}}, \mathsf{ind}) = 1,\ m_{\mathsf{eval}} = m_{\mathsf{out}} \\ 0 & \text{otherwise} \end{cases}.$$

Phrased in our terminology, the functionality ensures that the verification with public key id_{pub} of the signature generated with the evaluation key k_{ind} for message $m_{\mathsf{in}} = m_{\mathsf{out}}$ is

valid, if the user with index ind is a member of the polynomial-time relation p. This way, one may immediately obtain definitions for identity-based or attribute-based signatures. In an identity-based signature scheme, keys are assigned to identities, and the predicate $p_{\mathsf{ind}} : \mathcal{ID} \times \mathcal{I} \to \{0,1\}$ accepts, if keys are linked to the matching index $k_{\mathrm{ind}} = \mathsf{ind}$. In an attribute-based signature scheme, keys are associated with sets of strings (attributes), and the predicate $p_{\mathrm{att}} : \mathcal{ID}^* \times \mathcal{I}^* \to \{0,1\}$ accepts if there exists a polynomial time relation between the key identifiers and indices.

Maji et al. [MPR11] observe that attribute-based signatures relate to many important concepts of multi-user signature systems such as group, ring, and mesh signatures. (As mentioned in the Introduction, this addresses the core signing operation, but excludes additional functionality such as the revocation of anonymity in case of group signatures.) In a nutshell, in these multi-user signature systems, a set of users (group) is entitled to sign a message on behalf of the group and the common interpretation in all these signature systems is that they provide some form of *unforgeability* and *privacy*. More specifically, group and ring signatures reveal only the fact that the signer is a member of the group. In a group signature scheme, the list of group members is public, but chosen by the group manager in advance; in a ring signature, the list is public as well, but chosen by the signer in an ad hoc way. Mesh signatures extend ring signatures with some finer-grained access structure.

One easily may cast the "eligibility" of the user within the group by defining an appropriate polynomial time predicate P where keys are linked with user identities and the relation is a disjunction over the universe of attributes (group and ring signatures) or a finer-grained mechanism (mesh signatures). In the forthcoming section, we give a concrete instantiation of ring signatures. In fact, we discuss an alternative way to cast ring signatures in our framework, without relying on indices as here. There we also discuss our notion of unforgeability and privacy, as defined in Section 3, along a comparison to standard security definitions for ring signatures in the literature.

A.4 Ring Signatures

To exemplify further our definitions we now show how to map the security requirements of ring signatures to our notions of unforgeability and privacy. To cast ring signatures in our framework we have (adjacent) key pairs $\mathsf{sk}_{id}, \mathsf{pk}_{id}$ for each user, and also "rings of identifiers" $id_R = (id_1, \ldots, id_n)$ of public keys $\mathsf{pk}_{id_1}, \ldots, \mathsf{pk}_{id_n}$. The predicate ensemble $\mathcal{P}$ only consists of the predicate $P_{\mathtt{is\_part}}$ which, on input $(id_{\mathsf{eval}}, m_{\mathsf{eval}}), (id_{\mathsf{out}}, m_{\mathsf{out}})$ returns 1 if and only if $id_{\mathsf{eval}} = id_{\mathrm{sig}}$ for some $id_{\mathrm{sig}} \notin \mathcal{ID}_{\mathrm{pub}}$, and the matching public key $id_{\mathrm{pub}} \in \mathcal{ID}_{\mathrm{pub}}$ to id_{sig} is in the ring id_{out}, and $m_{\mathsf{eval}} = m_{\mathsf{out}}$. Note that the fact that we have a master secret key which is used to generate keys in an ad-hoc manner is somewhat corresponds to the case of ring signatures in [RST01a, BKM09] where the key pairs are generated at the outset.

Unforgeability according to our Definition 3.2 is then equivalent to unforgeability with respect to insider corruption in [BKM09]. That is, the notion in [BKM09] says that the adversary receives public keys of honest parties (as here) and can ask to see ring signatures for arbitrary messages, rings, and signing parties. This is given here through Eval' queries, where id_{eval} determines the signing party, m_{eval} the input message, and id_{out} the ring. The

adversary there also gets to corrupt parties and learn their secret keys; this can be modeled here through KeyGen' queries.

The adversary in [BKM09] eventually outputs a forgery for a message m^* and ring R^* of uncorrupt parties only, where it has never ask for a signature for (m^*, R^*) for any signer. In our terminology, the signing queries about R, m (i.e., the Eval' queries) add exactly such entries $(id_{\mathsf{out}}, m_{\mathsf{out}})$ corresponding to (R, m) to the set of trivial pairs. Furthermore, for any corrupt user (i.e., with $id_{\mathrm{sig}} \in \mathcal{QID}$), the trivial set contains any pair $(id_{\mathsf{out}}, *)$ where the public key id_{pub} to id_{sig} is in the ring id_{out}, because the adversary could trivially compute such signatures if it knows one of the secret keys of the users in the ring. However, the fact that R^* in a successful forgery corresponds to a ring of honest users only means that no pair $(id_{\mathsf{out}}, *)$ for id_{out} corresponding to R^* is trivial, unless it was signed, such that the forgery would also be valid according to our definition. In fact, there is a one-to-one correspondence between the two notions.

As for anonymity, our privacy notion (almost) corresponds to the anonymity against full key exposure notion in [BKM09], the only difference being that there the adversary does not only receive the secret key of parties, but also the randomness used to create them. There, the adversary can query a challenge oracle (multiple times) about a message m, a ring R including two user indices, and then receives a signature for m for ring R under either the left or right user index, depending on a secret random bit b. This corresponds exactly to our challenge oracle here, where the requirements for such queries stipulate that $P_0 = P_1 = P_{\mathtt{is\_part}}$, that $m^0_{\mathsf{eval}} = m^1_{\mathsf{eval}} = m_{\mathsf{out}}$, and that both ${id_{\mathsf{eval}}}^0$ and ${id_{\mathsf{eval}}}^1$ are thus part of the (same) ring id_{out}. The final prediction of the adversary for b there is then exactly the same as here.

A.5 Delegatable Functional Signatures

Delegatable Functional Signatures (DFS) [BMS13] 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. The functionality describes both the *malleability* of the signature (i.e., *how* it can be modified) and the *delegatability* of the signature (i.e., *to whom* it can be delegated).

DFS are general enough to imply many other primitives, amongst them functional signatures [BGI14]. However, DFS are a special case of OSS, as we can map the security requirements of DFS to our notions of unforgeability and privacy. As for ring signatures we have (adjacent) key pairs $\mathsf{sk}_{id}, \mathsf{pk}_{id}$ for each user, where we assume that there is a function pk that relates these keys as $\mathsf{pk}_{id} = \mathsf{pk}(\mathsf{sk}_{id})$. As in DFS the signer of a message can choose a function for the malleability and delegatability and specify an evaluator by its public key. Consequently, we compose messages as a combination of the real message m, the function f for malleability and delegatability and the public key k of the evaluator: $\mathsf{m} = (m, f, k)$. Then we build a predicate P that computes the functionality $\mathcal{F}$ of the DFS. To this end, the predicate checks whether the messages m_{in} and m_{out} are of the form (m, f, k). Then, P computes the functionality $\mathcal{F}$ and checks whether m_{out} contains the val-

ues that $\mathcal{F}$ computes. Formally, given a functionality $\mathcal{F}$, we create a predicate $P_{\mathcal{F}}$ on input $(m_{\mathsf{in}}, id_{\mathsf{in}}, m_{\mathsf{eval}}, id_{\mathsf{eval}}, m_{\mathsf{out}}, id_{\mathsf{out}})$ as follows: If $id_{\mathsf{out}} = \mathsf{pk}(id_{\mathsf{eval}})$, then this is a new signature and the predicate outputs 1. Otherwise, the predicate checks whether the evaluation is allowed, by verifying that m_{in} is of the form (m, f, k) and that m_{out} is of the form (m', f', k') and that moreover $k = \mathsf{pk}(id_{\mathsf{eval}}) \wedge id_{\mathsf{in}} = id_{\mathsf{out}} \wedge \mathcal{F}(f, m_{\mathsf{eval}}, k', m) = (m', f')$.

Unforgeability according to our Definition 3.2 is then equivalent to unforgeability with respect to insider or outsider attackers (depending on whether the adversary asks for secret keys and not just for public keys). The notion in [BMS13] is very related to our notion in that the adversary has access to (almost) the same oracles and also their definition of trivial messages (via the transitive closure of $\mathcal{F}$) is equivalent to our set Triv. Only their security against strong insider attackers, that may generate their own keys, is stronger and cannot be achieved in general.

The privacy notion of DFS marks a special case of our privacy notion from Definition 3.3 as it states that evaluations of signatures are indistinguishable from fresh signatures. However, the notions are equivalent, as their privacy (against fresh signatures) implies Definition 3.3.

B Proof of Unforgeability (Lemma 4.5)

Proof. Let $\mathcal{A}$ be an efficient adversary with running time t that makes at most (q_k, q_e, q_v) queries to its respective oracles (as in Definition 3.2), where t, q_k, q_e, q_v are polynomial in λ and assume for contradiction that $\mathcal{A}$ breaks the unforgeability with non-negligible probability λ. Then, we construct an adversary $\mathcal{B}$ against unforgeability of S. This algorithm gets as input a key pk and it chooses a key $\mathsf{k}_{\mathsf{PRF}}$ for a pseudorandom function PRF. We denote by Q the set of message that $\mathcal{B}$ queries to its signing oracle. In the following we describe $\mathcal{B}$'s simulation of the random oracle $H(\cdot)$ and the circuit $C_{\text{fake}}(((\mathbf{id}[\mathsf{k}_{\mathsf{eval}} \to \mathbf{id}_{\mathsf{eval}}], \mathbf{m}, \sigma), P), h)$ that will be used later.

$\mathcal{B}$ computes $c \leftarrow i\mathcal{O}(C_{\text{fake}})$, sets $\mathsf{PP} := (c, \mathsf{pk})$, and runs a black-box simulation of $\mathcal{A}$ on input PP and handles $\mathcal{A}$'s oracle queries as follows (where it updates $\mathcal{QID}$ and $\mathcal{QE}$ just as the challenger would).

- Upon receiving a request for $\mathsf{KeyGen}'(id)$ for any identifier $id \in \mathcal{ID}$, the algorithm $\mathcal{B}$ sends a query $\mathsf{Sig}(id)$ to its challenger, adds id to $\mathcal{QID}$ and returns the corresponding answer to $\mathcal{A}$.

- Upon receiving a request for $\mathsf{Eval}'((\mathbf{id}, \mathbf{m}, \sigma), P)$, the algorithm $\mathcal{B}$ simulates Eval, i.e., checks whether the signatures are valid (for id_{eval} it only checks whether $id_{\mathsf{eval}} \in \mathcal{ID}$) and whether the predicate is fulfilled. If so, $\mathcal{B}$ adds $(id_{\mathsf{out}}, m_{\mathsf{out}}, P)$ to $\mathcal{QE}$, sends a query $\mathsf{Sig}(m_{\mathsf{out}} || id_{\mathsf{out}})$ to its challenger and upon receiving an answer σ, the algorithm $\mathcal{B}$ sends σ to $\mathcal{A}$.

In the following, we show that $\mathcal{B}$ does not compute the forgery for $\mathcal{A}$, i.e., all responses to $\mathcal{A}$'s query belong to the set of trivial queries.

Claim B.1. *Whenever $\mathcal{B}$ sends a query $\mathsf{Sig}(x)$ to its signature oracle, then either x is a known key identifier $x \in \mathcal{QID}$, or x is of the form $x = (m||id)$ and the pair (id, m) is trivial.*

Proof for Claim B.1. There are exactly four places in the simulation at which $\mathcal{B}$ sends queries $\mathsf{Sig}(x)$ to its challenger and via recursion over the queries of $\mathcal{B}$ we will see that they all fulfill the claim:

- Keys contained in the set $\mathcal{ID}_{\text{pub}}$: It is assumed that these keys are publicly known and by definition it holds that $\mathcal{ID}_{\text{pub}} \subseteq \mathcal{QID}$.
- Upon receiving a request for $\mathsf{KeyGen}'(id)$ for any identifier $id \in \mathcal{ID}$: Here, $\mathcal{B}$ sends a query $\mathsf{Sig}(id)$ to its challenger, but now id is added to $\mathcal{QID}$.
- Upon receiving a request for $\mathsf{Eval}'(x)$: Here whenever $\mathcal{B}$ sends a query $\mathsf{Sig}(m||id)$ to its challenger and receives a signature σ, then it also adds (id, m) to $\mathcal{QE}$, i.e., (id, m) is trivial by the base case.
- Upon receiving a request for $H(x)$: Here $\mathcal{B}$ might send a query $\mathsf{Sig}(m_{\mathsf{out}}||id_{\mathsf{out}})$ to its challenger, but only if all input signatures verify and if furthermore the predicate is fulfilled. If any of the signatures (either for keys or for the input messages) has not been requested by $\mathcal{B}$ to its challenger (but still verifies), then $\mathcal{B}$ halts and outputs a forgery. Otherwise, if all signatures have been requested by $\mathcal{B}$, then via recursion over this claim, $\mathcal{A}$ knows $\mathsf{k}_{\mathsf{eval}}$ ($id_{\mathsf{eval}} \in \mathcal{QID}$) and all input pairs $(id_{\mathsf{in},i}, m_{\mathsf{in},i})$ are trivial. It follows that $(id_{\mathsf{out}}, m_{\mathsf{out}})$ is also trivial.

□

We now argue why any forgery produced by $\mathcal{A}$ leads to a forgery for S. To do so, we first show that the adversary $\mathcal{A}$ cannot distinguish its interaction with $\mathcal{B}$ from a real game against a challenger for the unforgeability of OSS. We show this by the following transition of indistinguishable games:

Game_0: The real game in which $\mathcal{A}$ interacts with a challenger for the unforgeability of OSS.

Game_1: The game Game_1 is defined as Game_0, but we replace the random oracle H by a (keyed) pseudorandom function PRF for which we draw a fresh key $\mathsf{k}_{\mathsf{PRF}}$ in the beginning of the game.

Claim B.2. *The games Game_0 and Game_1 are computationally indistinguishable.*

Since this claim follows easily by the pseudorandomness of PRF, we omit a formal proof. Note that by assumption about obfuscation of the hash-free C' part only, the obfuscator here also works for the pseudorandom hash function now.

Game_2: The game Game_2 is defined as Game_1, but in this game we replace the oracle $H_{\mathsf{Game}_1}(x) = \mathsf{PRF}(\mathsf{k}_{\mathsf{PRF}}, x)$ with H defined in Figure 2.

Claim B.3. *The games* Game_1 *and* Game_2 *are computationally indistinguishable.*

Again, this claim follows easily from the pseudorandomness of PRF and the formal proof is omitted.

Game_3: This game is identical to Game_2, but now $\mathcal{B}$ uses C_{fake} as defined in Figure 2.

Claim B.4. *The games* Game_2 *and* Game_3 *are computationally indistinguishable.*

Proof of Claim B.4. The circuits C from Game_2 and C_{fake} from Game_3 are functionally equivalent with respect to the oracle H. Consequently, by Definition 4.1 the obfuscated circuits $c \leftarrow i\mathcal{O}(\lambda, C^H)$ and $c' \leftarrow i\mathcal{O}(\lambda, C^H_{\text{fake}})$ are indistinguishable. Note that $\mathcal{A}$ does not learn any information about sk from c', as it is not used anymore. Moreover, since c and c' are indistinguishable and $\mathsf{k}_{\mathsf{PRF}}$ is only used in c', the attacker $\mathcal{A}$ also does not learn information about $\mathsf{k}_{\mathsf{PRF}}$ from c'. □

We have shown, that $\mathcal{A}$ cannot distinguish a game against the challenger for OSS from a game against the unforgeability of S in which $\mathcal{A}$ interacts with $\mathcal{B}$. Note that the circuit of Game_3 does not contain secret information about the signing key anymore (we shifted it to H). Thus, $\mathcal{B}$ can create the circuit in its own game.

To conclude the proof, we show that $\mathcal{B}$ can use $\mathcal{A}$'s advantage in breaking the unforgeability of OSS to break the unforgeability of S. For doing so we analyze what happens when $\mathcal{A}$ would win the game by sending a forgery (id^*, m^*, σ^*) to Verify'. Since S is a unique signature scheme, we know that σ^* has not been the output of $\mathcal{B}$ to a query Eval' by $\mathcal{A}$, as otherwise (id^*, m^*) would be trivial. We do a case distinction on the origin of the forgery.

Case 1: $\mathcal{A}$ has not sent any value x to H such that H has answered with the output $\mathsf{PRF}(\mathsf{k}_{\mathsf{PRF}}, x) \oplus \sigma^*$. But then $\mathcal{B}$ did not receive σ^* from its signature oracle, as it only requests signatures for messages in such cases and upon receiving an Eval' request. Note that by definition of $\mathcal{ID}$, the combination of id^* and m^* cannot be a key. Formally: $\forall id^* \in \mathcal{ID}, m^* \in \mathcal{M}.\ m^*||id^* \notin \mathcal{ID}$. Since the signature scheme S is deterministic, $\mathcal{B}$ has never sent a query $\mathsf{Sig}(m^*||id^*)$ to its challenger and consequently, $(m^*||id^*, \sigma^*)$ constitutes a forgery for $\mathcal{B}$.

Case 2: $\mathcal{A}$ has sent a value x to H such that H has answered with the output $\mathsf{PRF}(\mathsf{k}_{\mathsf{PRF}}, x) \oplus \sigma^*$. But then x must be of the form $((\mathbf{id}[\mathsf{k}_{\mathsf{eval}} \rightarrow \mathbf{id}_{\mathsf{eval}}], \mathbf{m}, \sigma), P)$ and all checks on $((\mathbf{id}[\mathsf{k}_{\mathsf{eval}} \rightarrow \mathbf{id}_{\mathsf{eval}}], \mathbf{m}, \sigma), P)$ must have succeeded. Then either all signatures $\sigma_{\mathsf{in},i}$ and the key $\mathsf{k}_{\mathsf{eval}}$ were handed to $\mathcal{A}$ by $\mathcal{B}$, and by Claim B.1 they must be trivial and thus (id^*, m^*, σ^*) is trivial, or $\mathcal{B}$ has already received a forgery, as there is a signature σ_i for an identifier-message-pair $(id_{\mathsf{in},i}, m_{\mathsf{in},i})$ such that $\mathsf{Verify}(\mathsf{pk}, m_{\mathsf{in},i}||id_{\mathsf{in},i}, \sigma_{\mathsf{in},i}) = 1$ or for the key $\mathsf{k}_{\mathsf{eval}} = (\mathsf{pk}, id_{\mathsf{eval}}, \sigma_{\mathsf{eval}}, c)$ such that $\mathsf{Verify}(\mathsf{pk}, id_{\mathsf{eval}}, \sigma_{\mathsf{eval}}) = 1$ and $\mathcal{B}$ has not sent a query $\mathsf{Sig}(id_{\mathsf{in},i}||m_{\mathsf{in},i})$ or $\mathsf{Sig}(id_{\mathsf{eval}})$ to its challenger.

We have shown that if we assume that $\mathcal{A}$ has a non-negligible advantage in producing a forgery for OSS, then $\mathcal{B}$ also has a non-negligible advantage in producing a forgery for S. This, however, contradict the assumption that S is unforgeable and thus, the opposite is true and our construction for OSS is unforgeable.

This completes the proof of Lemma 4.5. □

C Proof of Blindness (Theorem 5.2)

Proof. Assume that we have a successful attacker S^* on the blindness, which outputs a bit b^* and succeeds in predicting the random bit b with probability $\frac{1}{2} + \epsilon(\lambda)$. We show that the advantage $\epsilon = \epsilon(\lambda)$ must be negligible, by performing some game hops which would preserve a non-negligible advantage, until we finally reach a game in which the advantage disappears completely. This shows that ϵ must be negligible, too.

Game_0: This is the real blindness game against the scheme.

Game_1: Change the blindness game (Game_0) now as follows to Game_1. Initialize two copies of the malicious signer, both with honestly generated tuples $(\mathsf{MSK}, \mathsf{sk}_{id_{\text{sig}}}, \mathsf{PP}, \mathsf{pk}_{id_{\text{pub}}}, \text{pk}_{\mathcal{C}}, \omega)$ where ω is the randomness for the signer. Since these values are identical in both copies, we also obtain the same pair of messages m_0, m_1 in both executions. Run now one copy, called the main branch, on fresh randomness r_0, r_1 and $b \leftarrow \{0,1\}$ to compute the commitments C_0, C_1, and run the other instance, the side branch, with independent randomness r'_0, r'_1, b' to compute commitments C'_0, C'_1 of the same messages m_0, m_1 but in independent order. Forward the commitment pairs in both instances to the signer and wait to receive signatures τ_0, τ_1 as well as τ'_0, τ'_1. Abort the side branch.

If one of the signatures in the side branch, τ'_0 or τ'_1, is invalid, then abort the whole game and output a random guess for b; if one of the signatures τ_0, τ_1 in the main branch is invalid, then continue to run the signer in the main branch, but return $(\perp, \perp)$ on behalf of the user. Else, run locally the Eval algorithm on all four values as the honest user would, to obtain signatures $\sigma_{m_b}, \sigma_{m_{1-b}}, \sigma'_{m_{b'}}, \sigma'_{m_{1-b'}}$. Completeness says that these signatures must be valid. Output (m_0, σ_{m_0}) and (m_1, σ_{m_1}) to the signer in the main branch. Copy the signer's final prediction in the main execution for b as the output.

We claim that the probability of being able to predict b correctly in the modified game for our given malicious signer S^*, is still at least $\frac{1}{2} + \epsilon^2 - \mathsf{neg}(\lambda)$ for some negligible function $\mathsf{neg}(\lambda)$ which is still non-negligibly far from the pure guessing strategy. To see this, let $\textsc{Abort}_0$ be the event in Game_0 —with only the main branch— that the user fails to generate two valid signatures (and quasi aborts). Let $\textsc{Pred}_0$ be the event that S^* predicts $b^* = b$ correctly in Game_0. Note that the success probability of S^* in the original blindness game can then be written as follows:

$$\text{Prob}[\textsc{Pred}_0] \leq \text{Prob}[\textsc{Pred}_0 \wedge \textsc{Abort}_0] + \text{Prob}[\neg\textsc{Abort}_0]\,.$$

We argue that the first term is negligibly close to $\frac{1}{2}$, such that the second probability must be at least ϵ, up to a negligible term. To bound the first probability note that in case of an abort the only information S^* receives about b is via the commitments C_0, C_1 to the messages m_b, m_{1-b}. But since the commitments are computationally hiding, we can essentially replace them by commitments to 0's of the same length as the messages, without changing the signer's view significantly. Denote the corresponding events in this game by $\textsc{Pred}_0^0$, $\textsc{Abort}_0^0$. The probability of event $\textsc{Pred}_0^0 \wedge \textsc{Abort}_0^0$ cannot drop by more than a negligible term $\mathsf{neg}(\lambda)$ from the one in Game_0, or else we would easily obtain a successful distinguisher against the commitment scheme. Note that checking for the event $\textsc{Pred}_0^0 \wedge \textsc{Abort}_0^0$ to happen does not require the user to be able to generate signatures on the messages.

If we give commitments to 0's, on the other hand, S^* is completely oblivious about b in case an abort happens, such that:

$$\begin{aligned}\tfrac{1}{2} + \epsilon &\leq \mathrm{Prob}[\,\textsc{Pred}_0] \\ &\leq \mathrm{Prob}[\,\textsc{Pred}_0 \wedge \textsc{Abort}_0] + \mathrm{Prob}[\,\neg\textsc{Abort}_0] \\ &\leq \mathrm{Prob}\left[\,\textsc{Pred}_0^0 \wedge \textsc{Abort}_0^0\right] + \mathsf{neg}(\lambda) + \mathrm{Prob}[\,\neg\textsc{Abort}_0] \\ &\leq \tfrac{1}{2} + \mathsf{neg}(\lambda) + \mathrm{Prob}[\,\neg\textsc{Abort}_0]\,,\end{aligned}$$

implying that

$$\mathrm{Prob}[\,\textsc{Abort}_0] \leq 1 - \epsilon + \mathsf{neg}(\lambda).$$

With this we are now able to show that the success probability of predicting b in our Game_1 with the two branches (event $\textsc{Pred}_1$), is at least $\frac{1}{2} + \epsilon^2$ (minus some negligible term). Recall that the difference between the two games is that, if the side branch does not provide valid signatures, denoted as event $\textsc{Abort}_1$ here, we merely output a guess. Note that the probabilities of $\textsc{Abort}_0$ in Game_0 and of $\textsc{Abort}_1$ in Game_1 are identical, as they both consider a failure of one branch with random inputs. Since we have, under the condition that we do obtain valid signatures in the side branch, the same prediction probability here as in Game_0, we conclude:

$$\begin{aligned}\mathrm{Prob}[\,\textsc{Pred}_1] &= \mathrm{Prob}[\,\neg\textsc{Abort}_1] \cdot \mathrm{Prob}[\,\textsc{Pred}_1 \mid \neg\textsc{Abort}_1\,] + \tfrac{1}{2} \cdot \mathrm{Prob}[\,\textsc{Abort}_1] \\ &\geq (1 - \mathrm{Prob}[\,\textsc{Abort}_1]) \cdot (\tfrac{1}{2} + \epsilon) + \tfrac{1}{2} \cdot \mathrm{Prob}[\,\textsc{Abort}_1] \\ &= \tfrac{1}{2} + (1 - \mathrm{Prob}[\,\textsc{Abort}_1]) \cdot \epsilon \\ &\geq \tfrac{1}{2} + \epsilon^2 - \epsilon \cdot \mathsf{neg}(\lambda).\end{aligned}$$

Game_2: In the next game hop, let Game_2 be the game which is identical to Game_1, except that if outputting (m_0, σ_{m_0}) and (m_1, σ_{m_1}) to the signer in the main branch (if this point is reached), we now hand over (m_0, σ'_{m_0}) and (m_1, σ'_{m_1}) for the signatures from the side branch, and let S^* predict b from the main branch for these signatures.

We claim that, by the privacy of OSS, the success probability of S^* in this slightly modified game Game_2 cannot change significantly from Game_1. For this note that we

can view the original game, where S^* would obtain the genuine signatures (m_0, σ_{m_0}) and (m_1, σ_{m_1}), and the modified game here, correspond to the privacy game in which the challenger either derives the tags from the left input, or from the right input (twice). Our compound behavior of receiving the OSS keys, creating the other key for the commitment, and running S^* and the user instances as above, corresponds to an adversary against privacy. Hence, if the advantage of S^* of predicting b in the main execution would drop from non-negligible to negligible, we would obtain successful attacker against the privacy of OSS. Note that aborts are irrelevant for this argument.

Game_3: In the next game hop to Game_3, replace the commitments C_0, C_1 in the main branch by commitments to 0's (of equal length as the messages). Still use the original messages m_0, m_1 in the commitments C_0', C_1' for the side branch, and derive the final signatures from that execution. It follows straightforwardly from the hiding property of the commitment scheme that the advantage of S^* cannot change significantly, or else we would obtain a successful distinguisher.

Game_4: In the final game, the view of S^* in the main branch is now independent of the bit b, such that it cannot predict b better than by guessing. Since the game hops only lost a negligible success probability, it follows that S^* cannot predict b in the original game with non-negligible advantage either.

This completes the blindness analysis of Theorem 5.2. □

D Unforgeable Operational Signature Schemes

Boyle et al. [BGI14] give a simple construction for unforgeable functional signatures. Their construction relies on a standard digital signature scheme which shows that the existence of one-way functions implies the existence of unforgeable functional signature schemes. Their idea is to provide the signer a certified function and verification key for each functional signing key so that a signature consists of that certificate and a signature on the input message which verifies for the certified verification key.

Below we lift their construction and show how to construct even unforgeable operational signature schemes. The verification algorithm must additionally check whether the evaluator used the correct inputs when generating the tag, and whether the inputs together with the message and the signature comply with the respective predicate.

Construction D.1. *Let $\mathsf{S} = (\mathsf{KGen}_\mathsf{S}, \mathsf{Sig}_\mathsf{S}, \mathsf{Verify}_\mathsf{S})$ be a digital signature scheme. The operational signature scheme $\mathsf{OSS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Verify})$ for predicates $\mathcal{P}$ over $\mathcal{ID}, \mathcal{M}$ is defined as follows:*

$\mathsf{Setup}(1^\lambda)$: *Sample key pair $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KGen}_\mathsf{S}(1^\lambda)$. Set $\mathsf{MSK} = \mathsf{sk}$ and $\mathsf{PP} = \mathsf{vk}$. The set of (public) keys $\mathcal{ID}, \mathcal{ID}_{pub}$, the message space $\mathcal{M}$ is defined consistent to the key and message space of S.*

$\mathsf{KeyGen}(\mathsf{MSK}, id)$**:** *On input the master secret* MSK *and a key identifier* $id \in \mathcal{ID}$*, perform the following:*

1. *sample a key pair* $(\mathsf{ssk}, \mathsf{svk}) \leftarrow \mathsf{KGen}_{\mathsf{S}}(1^\lambda)$*, and*
2. *compute* $\sigma_{id} = \mathsf{Sig}_{\mathsf{S}}(\mathsf{MSK}, id||\mathsf{svk})$*, and*

return $\mathsf{k}_{id} = (\mathsf{ssk}, \mathsf{svk}, \sigma_{id})$*.*

$\mathsf{Eval}((\mathbf{id}[\mathsf{k}_{\mathsf{eval}} \to \mathbf{id}_{\mathsf{eval}}], \mathbf{m}, \sigma), P)$**:** *On input a sequence* $(id_{in,i}, m_{in,i}, \sigma_{in,i})_i$*, pairs* $(\mathsf{k}_{id_{eval}}, m_{eval})$ *and* (id_{out}, m_{out})*, and a predicate* $P \in \mathcal{P}$*, perform the following:*

1. *parse* $(\mathsf{ssk}, \mathsf{svk}, \sigma_{eval}) \leftarrow \mathsf{k}_{eval}$ *and*
2. *check whether for all* i*,* $\mathsf{Verify}(\mathsf{k}_{id_{in,i}}, m_{in,i}, \sigma_{in,i}) = 1$ *where* $\mathsf{k}_{id_{in,i}} \in \mathcal{ID}_{pub}$*,*
3. *sign all inputs, i.e.,* $\sigma_{in} \leftarrow \mathsf{Sig}_{\mathsf{S}}(\mathsf{ssk}, (\mathbf{id}, \mathbf{m}, \sigma)||P)$ *where*

$$(\mathbf{id}, \mathbf{m}, \sigma) = ((id_{in,i}, m_{in,i}, \sigma_{in,i})_i, (id_{eval}, m_{eval}), (id_{out}, m_{out}))\ .$$

Return $\sigma' = ((\mathbf{id}, \mathbf{m}, \sigma), P, \sigma_{eval}, \sigma_{in})$*.*

$\mathsf{Verify}(\mathsf{k}_{id}, m', \sigma')$**:** *On input key* $\mathsf{k}_{id} = (\mathsf{ssk}, \mathsf{svk}, \sigma)$*, a message* m'*, and a signature* $\sigma' = ((\mathbf{id}, \mathbf{m}, \sigma), P', \sigma'_{eval}, \sigma'_{in})$*, check if the following conditions hold:*

1. $m' = m_{out}$*,*
2. $P'(\mathbf{id}, \mathbf{m}) = 1$*,*
3. $\mathsf{Verify}_{\mathsf{S}}(\mathsf{svk}, (\mathbf{id}, \mathbf{m}, \sigma)||P', \sigma'_{in}) = 1$*, and*
4. $\mathsf{Verify}_{\mathsf{S}}(\mathsf{vk}, id_{out}||\mathsf{svk}, \sigma'_{eval}) = 1$*, and*
5. *for all* i*,* $\mathsf{Verify}(\mathsf{k}_{id_{in,i}}, m_{in,i}, \sigma_{in,i}) = 1$ *where* $\mathsf{k}_{id_{in,i}} \in \mathcal{ID}_{pub}$*.*

If either test fails, output 0*; else output* 1*.*

Theorem D.2. *If* S *is* (t, q_s, ε)*-unforgeable against chosen-message attacks, then the above construction for an operational message authenticator for predicates* $\mathcal{P}$ *over* $\mathcal{ID}, \mathcal{M}$ *is* $(t + O(\lambda), q_k, q_e, q_v, \varepsilon')$*-unforgeable where* q_k, q_e, q_v *are polynomially bounded in* λ*, and* $\varepsilon' = \varepsilon$*.*

Proof sketch. The construction is very close to the unforgeable functional signature scheme by Boyle et al. [BGI14], and the main difference here is that input signatures of previous evaluations are verified in the evaluation algorithm, and similarly in the verification algorithm. An adversary can forge essentially in the way as it does in the construction in [BGI14] or she must have been able to inject a forged signature in the evaluation algorithm. The former attack(s) are covered by Boyle et al. through the security reduction to the underlying signature scheme. Similarly, one can show that a forged input signature to the evaluation algorithm can also serve as a forgery in the unforgeability of the underlying signature scheme. In the beginning of the reduction, one has to guess the public key under which the forgery will later verify. Note that are only polynomial many public keys. We omit a formal analysis. □