Relaxed Security Notions for Signatures of Knowledge

Marc Fischlin and Cristina Onete

Darmstadt University of Technology & CASED, Germany
www.minicrypt.de

Abstract. We revisit the definition of signatures of knowledge by Chase and Lysanskaya (Crypto 2006) which correspond to regular signatures but where the signer also proves knowledge of the secret key to the public key through any signature. From a more abstract point of view, the signer holds a secret witness w to a public NP statement x and any signature to a message allows to extract w given some auxiliary trapdoor information. Besides extractability, Chase and Lysanskaya also demand a strong witness-hiding property, called simulatability, akin to the zero-knowledge property of non-interactive proofs. They also show that this property ensures anonymity for delegatable credentials or for ring signatures, for example.
In this work here we discuss relaxed notions for simulatability and when they are sufficient for applications. Namely, in one notion we forgo any explicit witness-hiding notion, beyond some weak requirement that signatures should not help to produce further signatures, analogously to unforgeability of regular signature schemes. This notion suffices for example for devising regular signature schemes with some additional proof-of-possession (POP) or knowledge-of-secret-key (KOSK) property. Our stronger notion resembles the witness-indistinguishability notion of proofs of knowledge and can be used to build anonymous ring signatures. Besides formal definitions we relate all notions and discuss constructions and the aforementioned applications.

Keywords Signature of Knowledge, Anonymity, Credential, Ring Signature

1 Introduction

Signatures of knowledge (SoK), a term coined in [8], are widely used in cryptography (e.g., [7,6,18,22]). The intuition behind SoKs is clear: besides basic signature security, signatures of knowledge should also prove that the signer "knows" the secret key. SoKs were, however, formalized only recently by Chase and Lysyanskaya [12].

In [12], SoKs are abstractly considered, as primitives allowing users $\mathcal{S}$ to sign messages such that if the signature verifies, a verifier knows that $\mathcal{S}$ has a witness w to some NP statement x; no further information is leaked about w though.

Chase and Lysyanskaya give two equivalent formalizations: a simulation-based definition in Canetti's Universal Composition (UC) framework [10], following approaches for regular signature schemes [11], and a game-based definition —called SimExt security— containing an extractability experiment akin to knowledge extractors [2] and a simulatability notion similar to non-interactive zero-knowledge (NIZK) proofs [4].

Since SimExt security closely resembles the security of NIZK proofs of knowledge (NIZKPoK), it is unsurprising that the construction in Chase and Lysyanskaya [12] is based on such proofs. The generality of this approach on the one hand yields quite expensive solutions, deploying general NIZKPoKs, but on the other hand also supports many applications. Two applications shown in [12] are ring signatures, where signers prove knowledge of a secret key corresponding to one of the public keys of the ring but without revealing its identity, and delegatable anonymous credentials, where zero-knowledge guarantees anonymity.

1.1 Relaxing the Notion of Signature of Knowledge

Reconsider SoK-based ring signatures. In this case simulatability as defined in [12] yields very strong anonymity: the SoK is simulatable without any witness. This is stronger than the security requirements of ring signatures, where only the actual signer should be hard to identify. We may thus consider a switch to the weaker notion of witness indistinguishability (WI) for SoKs, ensuring that one cannot deduce which (valid) witness w was used to sign. This relaxation thus allows for potentially more efficient solutions.

Consider furthermore simple digital signatures with key registration, where (some) information about the secret key is shown. Such registration steps are both common in practice [1,19], where one simply signs the public key to be registered, and often required in theory to prove security of protocols based on such signature schemes [5,17,20]. The corresponding model is called the "knowledge of secret keys" (KOSK) model and it implements some kind of proof of knowledge. Extractability of SoKs combines theory with practice, because self-signed public keys then mirror the KOSK model (though one can now extract with each signature and does not need an extra registration step). However, ordinary digital signatures usually do not require simulatability, but only unforgeability.

1.2 Our Contributions

We introduce two relaxed security notions for SoKs, following the NIZKPoK approach of [12] and thus inheriting extractability; however simulatability no longer holds. Instead, we transfer the definition of unforgeability from regular signatures and introduce UnfExt (*Unf*orgeability & *Ext*ractability) security as a minimal security level for SoKs. We augment this notion by adding witness indistinguishability, deriving the stronger WIUnfExt security. The SimExt definition of [12] is yet one step stronger, replacing witness indistinguishability by simulatability (as [12] shows, SimExt security implies unforgeability). We relate

all three notions formally, showing a strict hierarchy, and also provide equivalent definitions in the UC framework.[1]

We then instantiate our notions. Using a result about the security of Waters' signature scheme [23] in the KOSK model [20], we easily get an UnfExt SoK (for a special NP relation). In fact, this scheme is trivially witness-indistinguishable, too, as witnesses are unique. We next present a general construction of WIUnfExt-secure SoKs for arbitrary NP statements based on general assumptions. This construction relies on witness-indistinguishable proofs of knowledge (a.k.a. ZAPs [13]), which are a relaxation of non-interactive zero knowledge proofs; however, our construction does in fact achieve SimExt security in the definition of Chase and Lysyanskaya [12]. Our third construction achieves UnfExt security by signing message m on behalf of an extension of the original statement-witness pair.

We finally address the aforementioned applications, especially ring signatures. We discuss that adding witness indistinguishability reflects strong anonymity of ring signatures. We use anonymity and unforgeability notions from the framework for ring signatures of Bender et al. [3].

2 Signatures of Knowledge

Signatures of knowledge (SoKs) are protocols between a *signer* $\mathcal{S}$, which signs messages $m \in \mathcal{M}$, and a *verifier* $\mathcal{V}$ checking signature validity.

We identify NP languages L with arbitrary, but fixed relations $\mathcal{R}^L$, i.e., $x \in L$ iff there exists a polynomial-size witness w such that $(x, w) \in \mathcal{R}^L$. Jumping ahead, we also require that it is hard, given some x, to compute a valid witness w (we formalize this w.r.t. an instance generator, as shown in section 3). We assume efficient (i.e., polynomial in the length $|x|$ of x) verification of $(x, w) \in \mathcal{R}^L$; denote by $\mathcal{W}^L(x)$ the possibly empty set $\{w : (x, w) \in \mathcal{R}^L\}$ of witnesses to x. Note that $\mathcal{W}^L(x)$ formally depends on $\mathcal{R}^L$, not on L. Sets $S = \mathcal{M}, L, \mathcal{R}^L, \mathcal{W}^L, \ldots$ are usually indexed by the security parameter $k \in \mathbb{N}$ and S_k denotes the strings $s \in S$ of polynomial complexity in k (for some fixed polynomial).

Definition 1 (Signature of Knowledge). *A* Signature of Knowledge (SoK) *for relation $\mathcal{R}^L$ is a tuple of efficient algorithms* $\mathsf{SoK} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Vf})$ *where:*

$\mathsf{par} \leftarrow \mathsf{Setup}(1^k)$. *For a security parameter k,* Setup *outputs public parameters* par. *We assume that k is efficiently recoverable from* par.

$\sigma \leftarrow \mathsf{Sign}(m, x, w, \mathsf{par})$. *For a message $m \in \mathcal{M}_k$, statement $x \in L_k$, witness $w \in \mathcal{W}_k^L(x)$, and parameters* par *(generated for k),* Sign *outputs SoK σ.*

$b \leftarrow \mathsf{Vf}(\sigma, m, x, \mathsf{par})$. *On input an SoK σ, a message m, a statement x, and parameters* par*, the algorithm* Vf *outputs bit b indicating the validity of the SoK ($b = 1$ for valid σ).*

[1] Note that a work by Zou and Sun [15], advertising to discuss stronger anonymity for signatures of knowledge, rather shows subliminal channels in some group signature schemes through malicious signers, and is therefore not discussed further here.

We require the usual correctness property: for any $x \in L_k$, any $w \in \mathcal{W}_k^L(x)$, and any $m \in \mathcal{M}_k$, it holds that,

$$\text{Prob}\left[\,\mathsf{par} \leftarrow \mathsf{Setup}(1^k); \sigma \leftarrow \mathsf{Sign}(m, x, w, \mathsf{par}) : \mathsf{Vf}(\sigma, m, x, \mathsf{par}) = 1\right] \approx 1,$$

i.e., is negligibly close to 1 *(as a function of k).*

3 Security Notions for SoKs

We first briefly describe SimExt security as in [12] and then introduce *UnfExt* security as a relaxation thereof. We also explain the relation between the notions and introduce WIUnfExt security as another flavor of SoK security. We then show equivalent definitions in Canetti's universal composition framework.

3.1 Simulatability, Unforgeability, and Witness Indistinguishability

In [12], SimExt security considers auxiliary inputs given to the adversary. We omit such inputs for simplicity and instead use efficient algorithms, covering both uniform and non-uniform (with auxiliary input) computational models, as needed. Furthermore SoKs are universal in [12], using (the machine verifying) the relation $\mathcal{R}^L$ as input. Here we define SoK for specific fixed $\mathcal{R}^L$, which is handier for instantiations, e.g., for specific SoK for discrete-log based relations.

Definition 2 (SimExt Security for SoK [12]). *The SoK scheme* $\mathsf{SoK} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Vf})$ *is* SimExt secure *for $\mathcal{R}^L$ iff it is:*

Simulatable. *There exists an efficient simulator* $\mathsf{Sim} = (\mathsf{SimSetup}, \mathsf{SimSign})$ *such that for all efficient adversaries $\mathcal{A}$ it holds that*

$$\left|\begin{array}{l}\text{Prob}\left[\,(\mathsf{par}, \tau) \leftarrow \mathsf{SimSetup}(1^k) : d \leftarrow \mathcal{A}^{\mathsf{Sim}(\mathsf{par}, \tau, \cdot, \cdot, \cdot)}(\mathsf{par}) : d = 1\right] \\ \quad - \text{Prob}\left[\,\mathsf{par} \leftarrow \mathsf{Setup}(1^k); d \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{par}, \cdot, \cdot, \cdot)}(\mathsf{par}) : d = 1\right]\end{array}\right| \approx 0,$$

where, on input $(\mathsf{par}, \tau, m, x, w)$, Sim *checks that $\mathcal{R}^L(x, w) = 1$; if so, it returns* $\mathsf{SimSign}(\mathsf{par}, \tau, m, x)$*, otherwise it ouputs $\perp$.*

Extractable. *There additionally exists an efficient extractor* Ext *such that for all efficient $\mathcal{A}$,*

$$\begin{array}{l}\text{Prob}\ \ [(\mathsf{par}, \tau) \leftarrow \mathsf{SimSetup}(1^k); (x, m, \sigma) \leftarrow \mathcal{A}^{\mathsf{Sim}(\mathsf{par}, \tau, \cdot, \cdot, \cdot)}(\mathsf{par}); \\ \qquad w \leftarrow \mathsf{Ext}(\mathsf{par}, \tau, x, m, \sigma) : \\ \qquad (x, w) \in \mathcal{R}_k^L \vee (m, x) \in Q \vee \mathsf{Vf}(\sigma, m, x, \mathsf{par}) = 0] \approx 1\end{array}$$

Here, Q is the list of (m, x) queries that $\mathcal{A}$ has made to Sim.

Note that Simulatability guarantees that using Sign or SimSign is essentially equivalent for extractability. Simulatability is a strong requirement for SoKs,

resembling zero-knowledge simulation for non-interactive proofs. Some witness-"protection" is necessary, however: we cannot restrict SoK security to just correctness and extractability, as this allows for insecure SoKs. Indeed, consider an SoK scheme outputting $w||m$ for each m, and where Vf checks the validity of w. This SoK is correct and trivially extractable. However, any adversary can create a SoK on fresh m^*, either by extracting a valid w from queried SoKs, or by modifying queried SoKs such that the new SoK verifies for m^*.

Thus, a minimal security of SoKs additionally requires the basic (existential) unforgeability under adaptive chosen message attacks of common signatures. This requires that computing a witness w from a statement x is infeasible, else unforgeability cannot hold. We capture this by introducing an *instance generator* IGen outputting $(x, w) \in \mathcal{R}^L$ accordingly. Consider the example where IGen outputs a group element x and its discrete logarithm w (w.r.t. some group generator). We say that IGen is a *hard-instance generator* if, in addition, no efficient algorithm can, on input x for $(x, w) \leftarrow \mathsf{IGen}(1^k)$, output some $w^* \in \mathcal{W}^L(x)$ with non-negligible probability.

We now define UnfExt SoKs in the notation of [12]. To connect Unforgeability and Extractability we assume that the parameters in the two experiments are indistinguishable (else the notions could be perfectly independent):

Definition 3 (UnfExt Security). *The SoK* $\mathsf{SoK} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Vf})$ *is* UnfExt secure *for* $\mathcal{R}^L$ *and* IGen *iff it is:*

Extractable. *There exists an efficient extractor* $\mathsf{Extr} = (\mathsf{ExtSetup}, \mathsf{Ext})$ *such that for any efficient* $\mathcal{A}$

$$\begin{aligned}\mathrm{Prob}[&(\mathsf{par}, \tau) \leftarrow \mathsf{ExtSetup}(1^k); (x, m, \sigma) \leftarrow \mathcal{A}(\mathsf{par});\\ &w^* \leftarrow \mathsf{Ext}(\mathsf{par}, \tau, x, m, \sigma) : (x, w^*) \in \mathcal{R}^L \vee \mathsf{Vf}(\sigma, m, x, \mathsf{par}) = 0] \approx 1.\end{aligned}$$

Unforgeable. *For all efficient* $\mathcal{A}$*,*

$$\begin{aligned}\mathrm{Prob}[&(x, w) \leftarrow \mathsf{IGen}(1^k); \mathsf{par} \leftarrow \mathsf{Setup}(1^k);\\ &(m, \sigma) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot, x, w, \mathsf{par})}(x, \mathsf{par}) : m \notin Q \wedge \mathsf{Vf}(\sigma, m, x, \mathsf{par}) = 1] \approx 0\end{aligned}$$

Here, the list Q *contains queries* m *to* Sign *(note that the oracle is initialized with the generated* x *and* w*, thus these parameters are not part of the queries).*

Parameter Indistinguishability. *The output* par *in* $(\mathsf{par}, \tau) \leftarrow \mathsf{ExtSetup}(1^k)$ *is computationally indistinguishable from the output* $\mathsf{par} \leftarrow \mathsf{Setup}(1^k)$.

Note that the extractability notion in [12] gives the adversary access to a simulated signing oracle, allowing the adversary to see simulated signatures for arbitrary messages (however, Ext need not extract witnesses from simulated signatures). Since we do not consider simulated signatures in our definition, we drop the oracle access and require extractability for any message.

As aforesaid, unforgeability in the UnfExt notion is equivalent to regular chosen-message unforgeability for digital signatures. As a first sanity check, note

that our trivial example where the SoK included w is not unforgeable, as $\mathcal{A}$ can insert *any* fresh message into a forgery so that it verifies.

We show later that UnfExt security is strictly weaker than SimExt security. Indeed, as aforementioned, UnfExt SoKs may leak some information about the witness w, but not to the extent that it allows forgeries. An intermediate security level between UnfExt and SimExt security combines UnfExt security with witness indistinguishability (WI). As we show after the definition, we still need unforgeability to exclude trivial examples (in particular, WI does not imply unforgeability). We formalize WIUnfExt security as follows:

Definition 4 (WIUnfExt Security). *The SoK* $\mathsf{SoK} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Vf})$ *is* WIUnfExt secure *for* $\mathcal{R}^L$ *and* IGen *iff it is:*

UnfExt. *The scheme is UnfExt scheme and, in addition,*

Witness Indistinguishable. *For all* $x \in L_k$*, all* $w_0, w_1 \in \mathcal{W}_k^L(x)$*, and all efficient* $\mathcal{A}$*,*

$$\mathrm{Prob}[\mathsf{par} \leftarrow \mathsf{Setup}(1^k); b \leftarrow \{0,1\}; \\ d \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot, x, w_b, \mathsf{par})}(x, w_0, w_1, \mathsf{par}) : d = b] \approx \tfrac{1}{2}$$

Note that we demand witness indistinguishability even if $\mathcal{A}$ knows w_0, w_1. We also show that WI does not imply unforgeability. Consider a WIUnfExt SoK and change it into SoK' such that: $\mathsf{Setup}' = \mathsf{Setup}$; on input m algorithm Sign' runs Sign on message m, then runs Sign on message $\mathbf{0}$ (the all-zero string of some fixed length), and outputs $(\mathsf{Sign}(m), \mathsf{Sign}(\mathbf{0}))$ as its signature; and finally on inputs $m, (\sigma_m, \sigma_0)$, the verifier runs Vf on inputs (m, σ_m) and then on $(\mathbf{0}, \sigma_0)$. The new SoK' is still WI and extractable, but an adversary $\mathcal{A}$ against unforgeability simply queries Sign' on input $m \neq \mathbf{0}$, receives (σ_m, σ_0), and then outputs $\mathbf{0}, (\sigma_0, \sigma_0)$ as its forgery.

An alternative SoK security definition could use witness-hiding (WH) proofs of knowledge [14] where it is infeasible to recover the *entire* witness. However, unforgeability already implies WH: if a signature of knowledge is not WH, then it is also not unforgeable (the adversary can simply re-use the recovered witness w to sign a fresh message m^*).

3.2 Relationships of Security Notions

The strict hierarchy of SimExt, UnfExt, and WIUnfExt security appears in Fig. 1. Formally we have:

Proposition 1 (Relationships of Notions). *For any* $\mathcal{R}^L$ *and any hard-instance generator* IGen*, (1) any SimExt secure SoK for* $\mathcal{R}^L$ *is also WIUnfExt secure for* $\mathcal{R}^L$ *and* IGen*, and (2) any WIUnfExt secure SoK for* $\mathcal{R}^L$ *and* IGen *is also UnfExt secure. Furthermore, (3) if there exists an UnfExt secure SoK for some* $\mathcal{R}^L$ *and* IGen*, then there exists an SoK for some* $\mathcal{R}^{L'}$ *and* IGen' *that is UnfExt but not WIUnfExt; and (4) if there exists an WIUnfExt SoK for some* $\mathcal{R}^L$ *and* IGen*, and if one-way permutations (OWP) exist, then there exists an SoK for some* $\mathcal{R}^{L'}$ *and* IGen' *which is WIUnfExt but not SimExt.*

Proof. We prove claim (1). Consider a SimExt secure $\mathsf{SoK}=(\mathsf{Setup}, \mathsf{Sign}, \mathsf{Vf})$. SimExt and UnfExt correctness are identical. Extractability follows since the definitions are almost the same; however, as aforesaid $\mathcal{A}$ uses $\mathsf{SimSign}$ in SimExt security, whereas in WIUnfExt security, $\mathcal{A}$ has no oracle access – though it may still simulate Sign for valid pairs (x, w). By Simulatability, $\mathcal{A}$ cannot distinguish between $\mathsf{SimSign}$ and Sign; thus we can interchange them with a negligible change in the success probability. Unforgeability follows as described in [12] from the fact that IGen is a hard-instance generator (despite minor technical differences).

Indistinguishability of the parameters is a weaker requirement than simulatability. Finally, we prove witness indistinguishability. For this, we replace Setup by $\mathsf{SimSetup}$ and Sign by $\mathsf{SimSign}$ in the original WI game (using the trapdoor τ output by $\mathsf{SimSetup}$). In the modified game, $\mathcal{A}$ cannot distinguish between SoKs for the two witnesses, as they are generated independently of the witness. By Simulability, using Sign and $\mathsf{SimSign}$ are indistinguishable; thus the success probability in the modified game and that of the WI game are only negligibly different. Thus the SimExt secure SoK is also WIUnfExt secure.

Statement (2) follows by definition of UnfExt and WIUnfExt security.

For claim (3) consider UnfExt secure $\mathsf{SoK}=(\mathsf{Setup}, \mathsf{Sign}, \mathsf{Vf})$ for $\mathcal{R}^L$ and IGen. We construct $\mathsf{SoK}^* = (\mathsf{Setup}^*, \mathsf{Sign}^*, \mathsf{Vf}^*)$ for $\mathcal{R}^{L'}$ and IGen', defined for witnesses $\mathsf{W}=(w||b)$ (for bit b) and statements $\mathsf{X}=x$. Then $(\mathsf{X}, \mathsf{W}) \in \mathcal{R}^{L'}$ iff $(x, w) \in \mathcal{R}^L$, and IGen' samples (X, W) by running IGen and appending a random bit to w. Algorithms Setup^* and Vf^* run Setup, resp. Vf as black boxes, forwarding the output. Algorithm Sign^* on input W runs Sign as a black box, appending b from W to the output SoK. Clearly SoK^* inherits correctness, extractability, parameter indistinguishability, and unforgeability from SoK. Yet SoK^* is *not* WI, as signatures leak the added bit for witnesses $w||0$ and $w||1$.

In claim (4) we assume the existence of a OWP f. Consider WIUnfExt secure SoK. We construct SoK' that is still WIUnfExt secure, but not SimExt secure. For each statement x, choose random r and set $\mathsf{X} = x||f(r)$. All $\mathsf{W} \in \mathcal{W}^L(\mathsf{X})$ also include r, i.e., $\mathsf{W} = w||r$ (this later ensures WI). It is also easy to derive IGen' from IGen as in Claim (3). Algorithm Setup' of SoK' runs Setup from SoK, forwarding the output par. Algorithm Sign' of SoK' first runs Sign from SoK to get SoK σ. The output of Sign' is $\sigma||r$. Verification by Vf' runs Vf from SoK, then checks that $f(r)$ for the r in the SoK is the one featured in X.

For the analysis, note that SoK' is still WIUnfExt secure. Completeness, extractability, and indistinguishability of the parameters are trivial. Forging SoK' also involves forging SoK. Finally, WI is preserved as r is the same for all $w \in \mathcal{W}^L(x)$ for each x. However, SoK' is *not* SimExt secure under the one-wayness of f. In particular, it is not simulatable. Assume that there exists a simulator Sim that simulates Sign' to $\mathcal{A}$. The success probability is taken over all x; if Sim is successful, then we can build an inverter $\mathcal{B}$ against f. Indeed, Sim receives for every statement x the corresponding $f(r)$ for random r. If Sim simulates Sign successfully, it outputs the correct value r (else the SoK does not verify). Algorithm $\mathcal{B}$ runs Sim, outputting r as its pre-image, and is as successful as Sim. □

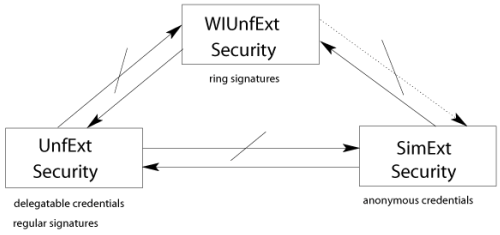

Fig. 1. Security of SoKs: arrows refer to implications, hatched arrows to separations; the dotted arrow indicates that the separation relies on an additional assumption. The figure also shows potential applications of the different notions.

3.3 Universally Composable Versions

In the Universal Composability (UC) framework due to Canetti [9], protocols are associated with ideal functionalities, describing permissible leakage of data in the protocol run. Several parties run the protocol, receiving input from a so-called environment $\mathcal{Z}$. A protocol π UC-realizes functionality F if $\mathcal{Z}$ cannot distinguish between a "real world" where parties run π around an adversary who gets inputs from, and outputs to $\mathcal{Z}$, and an "ideal world", where parties run F around a simulator Sim, also outputting to $\mathcal{Z}$. The adversary may corrupt parties, thus controlling them; these parties are marked down as corrupt. In the UC framework, π is secure if there exists a Sim such that for all $\mathcal{Z}$ and for all adversaries, $\mathcal{Z}$ cannot distinguish between the two worlds.

Chase and Lysyanskaya [12] give two equivalent definitions of SoKs. The first is UC-based, for a modification of the tweaked ideal signature functionality—[11]. For more details regarding ideal functionalities for signatures, refer to [12]. The UC definition for SoKs is equivalent to SimExt security, thus strictly stronger than UnfExt security. We show how to modify this definition to capture UnfExt security. The main difference is that we do not require simulatability for our signing algorithm. In particular, we use Sign and not $\mathsf{SimSign}$ for SoK generation.

Our ideal functionality (Figure 2) resembles the one in [12], but is simpler, as it is parameterized by $\mathcal{R}^L$, whereas [12] use a universal functionality and put the (code of the machine verifying the) $\mathcal{R}^L$ into session identities sid.

Note that as opposed to [12] the simulator is not among the algorithm descriptions. This follows our idea that full simulatability is not required for SoK. Note also that SoKs require some common parameter setup preceding it. As in [12], we use the CRS model and corresponding $\mathcal{F}_{\mathrm{CRS}}^{D}$ functionality, where D is a party-chosen distribution of the parameters. If party P forwards $(\mathrm{CRS}, \mathsf{sid})$ to $\mathcal{F}_{\mathrm{CRS}}^{D}$, the functionality checks that no value v is associated with this sid; else, it chooses v randomly according to D and stores it, returning $(\mathrm{CRS}, \mathsf{sid}, v)$ to both P and to the adversary.

For SoKs, D is the distribution of the parameters output by $\mathsf{Setup}(1^k)$. We run $\mathsf{SoK}= (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Vf})$ in a hybrid CRS environment and denote the re-

$\mathcal{F}_{SOK}(\mathcal{R}^L)$**: signature of knowledge for a witness** w **with** $(x, w) \in \mathcal{R}^L$**.**

SETUP. Upon receiving $(\mathsf{Setup}, \mathsf{sid})$ from party P, check that this is the first time a Setup request is made with parameter sid; if not, ignore, else $(\mathsf{Setup}, \mathsf{sid})$ is forwarded to the adversary, which eventually returns $(\mathsf{Algorithms}, \mathsf{sid}, x, \mathsf{Vf}, \mathsf{Sign}, \mathsf{Ext})$ to the functionality. Here Sign and Ext describe probabilistic polynomial time (PPT) algorithms (represented by PPT Turing Machines), and Vf describes a deterministic polynomial time algorithm. The algorithm descriptions and x are stored, and P receives $(\mathsf{Algorithms}, \mathsf{sid}, \mathsf{Sign}, \mathsf{Vf})$.

SIGNATURE GENERATION. Upon receiving $(\mathsf{Sign}, \mathsf{sid}, m)$ from P, run $\sigma \leftarrow \mathsf{Sign}(m)$ and check that $\mathsf{Vf}(\sigma, m, x) = 1$; if so, output $(\mathsf{Signature}, \mathsf{sid}, m, \sigma)$ to P and record (m, x, σ). Else, output $(\mathsf{Completeness\ Error})$ to P and halt.

SIGNATURE VERIFICATION. Upon receiving $(\mathsf{Verify}, \mathsf{sid}, \sigma, m, x')$ from verifier V, if (m, x', σ') is stored for some σ', then output $(\mathsf{Verified}, \mathsf{sid}, \sigma, m, x', \mathsf{Vf}(\sigma, m, x'))$ to V. Else, if $x' = x$ and $\mathsf{Vf}(\sigma, m, x) = 1$ but (m, x, σ) has not been stored yet, output $(\mathsf{Unforgeability\ Error})$ and halt. Else let $w' \leftarrow \mathsf{Ext}(m, x, \sigma)$; if $(w', x) \in \mathcal{R}^L$, output $(\mathsf{Verified}, \mathsf{sid}, \sigma, m, x, \mathsf{Vf}(\sigma, m, x))$ to V. Else, if $\mathsf{Vf}(\sigma, m, x) = 0$, output $(\mathsf{Verified}, \mathsf{sid}, \sigma, m, x, 0)$ to V. Else, output $(\mathsf{Extraction\ Error})$ and halt.

Fig. 2. Signature of Knowledge Functionality

sulting protocol $\pi_{\mathsf{SoK}}(\mathcal{R}^L, \mathsf{IGen})$. During each session of π_{SoK}, every time party P receives a $(\mathsf{Setup}, \mathsf{sid})$ message from the environment $\mathcal{Z}$, sid is checked and P queries $\mathcal{F}_{\mathrm{CRS}}^D$ so as to get $(\mathrm{CRS}, \mathsf{par})$. The public par are stored by P and P also generates $(x, w) \leftarrow \mathsf{IGen}(1^k)$. Both values are then included in the descriptions returned to $\mathcal{Z}$ as $(\mathsf{Algorithms}, \mathsf{sid}, \mathsf{Sign}(\mathsf{par}, \cdot, x, w), \mathsf{Vf}(\mathsf{par}, \cdot, \cdot))$.

If $\mathcal{Z}$ sends a request $(\mathsf{Sign}, \mathsf{sid}, m)$ to party P, this party retrieves the stored par and returns $(\mathsf{Signature}, \mathsf{sid}, m, \mathsf{Sign}(\mathsf{par}, m, x, w))$. If a verifier V receives request $(\mathsf{Verify}, \mathsf{sid}, \sigma, m, x')$ from $\mathcal{Z}$, it returns $(\mathsf{Verified}, \mathsf{sid}, \sigma, m, x', \mathsf{Vf}(\mathsf{par}, \sigma, m, x'))$ to $\mathcal{Z}$.

Like in [12], we can prove the equivalence of the two definitions. In particular, we formalize the following theorem.

Proposition 2 (Equivalence of Notions). *Protocol* $\pi_{\mathsf{SoK}}(\mathcal{R}^L, \mathsf{IGen})$ *UC-realizes the functionality* $\mathcal{F}_{SOK}(\mathcal{R}^L)$ *in the* $\mathcal{F}_{CRS}^D$ *hybrid model iff* SoK *is UnfExt secure for* $\mathcal{R}^L$ *and* IGen*.*

The proof closely follows that of [12] and is omitted for space reasons. A UC equivalence can also be extended to the notion of WIUnfExt security.

4 SoK Instantiation

In this section we recall Waters' signature scheme [23] and show that it is UnfExt secure; it is in fact also trivially WIUnfExt, as witnesses are unique. The latter point is also discussed in [20] in the related KOSK model. We also describe a

universal construction based on general assumptions and for arbitrary relations, which actually achieves the stronger SimExt security. We finally describe an UnfExt secure construction where we sign messages m for extended statement-witness pairs (x, w); an advantage here is that a single proof (ZAP) suffices for every statement-witness pair, rather than a proof for each message.

4.1 Waters' Signature Scheme

We construct UnfExt secure SoKs from Waters' unforgeable pairing-based signatures [23]. In particular, such signatures are already complete and unforgeable, becoming extractable if we add some master information about the randomness used for signature generation. We outline our construction and then consider its security. Note that we slightly abuse notation here as the key pairs now depend on the parameters, i.e., the relation depends on par; all results presented before remain valid in this setting.

Let $\mathsf{Sig}_W = (\mathsf{SKGen}_W, \mathsf{SSign}_W, \mathsf{SVf}_W)$ be the unforgeable signature scheme due to Waters. We review this construction briefly before outlining our UnfExt Secure SoK. In the schema due to Waters, the parameters (generated at Key Generation) consist of: multiplicative groups $\mathbb{G}$ and $\mathbb{G}^T$; a prime q; an element $g \in \mathbb{G}$ of prime order q; and (the description of) a bilinear mapping $\hat{e}$.

Key Generation. For security parameter k, algorithm SKGen_W runs an (external) generator $\mathcal{G}$ to generate parameters $\mathsf{par} = (\mathbb{G}, \mathbb{G}^T, q, g, \hat{e})$. The algorithm then picks a random $a \leftarrow \mathbb{Z}_q$ and computes $g_1 := g^a$. Then it chooses $g_2, u_0, \ldots, u_k \leftarrow \mathbb{G}$. Finally, the algorithm outputs the public/private keypair (pk, sk) for $pk = (\mathsf{par}, g_1, g_2, u_0, \ldots, u_k)$ and $sk = g_2^a$.

Signature Generation. For message $m \in \mathcal{M}$ and private key sk, the signing algorithm SSign_W parses $m = m_1 \ldots m_k$ for $m_1, \ldots, m_k \in \{0, 1\}$ and computes $H(m) \leftarrow u_0 \prod_{i=1}^{k} u_i^{m_i}$. It then picks a random $r \in \mathbb{Z}_q$; the signature output by SSign is $\sigma = (g_2^a \cdot H(m)^r, g^r)$.

Signature verification. For signature σ, message m, and public key pk, SVf parses σ as (σ_1, σ_2) and outputs 1 iff. $\hat{e}(g, \sigma_1) = \hat{e}(\sigma_2, H(m)) \cdot \hat{e}(g_1, g_2)$.

This signature scheme is complete and existentially unforgeable under adaptive chosen message attacks under the CDH assumption (see [23]).

We turn this construction into an UnfExt secure SoK by including $u_0, \ldots, u_k$ in the public parameters, thus allowing the simulator to extract the witness if the discrete logs of $u_0, \ldots, u_k$ with respect to g are in the trapdoor information generated by $\mathsf{ExtSetup}$. This idea is outlined in the following. The public parameters, besides $\mathsf{par}_W = (\mathbb{G}, \mathbb{G}^T, q, g, \hat{e})$ now also contain $u_0, \ldots, u_k$. To generate x and w via IGen the values g_2 and a are chosen as by SKGen_W; g_1 is set as $g_1 = g^a$; and witness $w = sk = g_2^a$ is given to the signer. The statement is then $x = (g, g_1, g_2)$ and we define the relation $\mathcal{R}^L$ as follows: $(x, w) \in \mathcal{R}^L$ iff $\hat{e}(g_1, g_2) = \hat{e}(g, w)$. If the values are generated honestly, we note that $\hat{e}(g_1, g_2) = \hat{e}(g^a, g_2) = \hat{e}(g, g_2)^a = \hat{e}(g, g_2^a) = \hat{e}(g, w)$.

SoK Setup. For security parameter k, the algorithm Setup generates $\mathsf{par}_W = (\mathbb{G}, \mathbb{G}^T, q, g, \hat{e})$ as in Sig_W and also chooses $z_0, \ldots, z_k \leftarrow \mathbb{Z}_q$ and then sets $u_i = g^{z_i}$ for $i = 0, \ldots, k$. Finally, Setup outputs $\mathsf{par} = (\mathsf{par}_W, u_0, \ldots, u_k)$.

SoK Generation. For message $m \in \mathcal{M}$, witness $w = g_2^a$, statement $x = (g, g^a, g_2)$, and parameters par, the signing algorithm Sign runs SSign_W and outputs signature σ.

SoK verification. For signature σ, message m, statement x, and public parameters par, the algorithm Vf runs SVf outputting the bit b.

Theorem 1 (UnfExt Security). *The signature of knowledge scheme* SoK *defined above is UnfExt Secure under the CDH assumption.*

Proof. We have to prove correctness, extractability, unforgeability, and parameter indistinguishability. First note that correctness and unforgeability (for IGen as described above) follow from the corresponding properties of Waters' signature scheme. We now describe an extractor Extr= ($\mathsf{ExtSetup}$, Ext) which, given a valid signature σ outputs witness w. The algorithm $\mathsf{ExtSetup}$ runs Setup and sets $\tau = (z_0, \ldots, z_k)$. In particular, the public parameters have identical distributions in both cases. For signature σ, message m, statement x, parameters par, and trapdoor information τ, the algorithm Ext parses σ as (σ_1, σ_2), calculates $d = z_0 + \sum_{i=1}^{k} z_i m_i \bmod q$ and outputs $w^* = \sigma_1 \sigma_2^{-d}$. Note that, if the signature verifies, we must have

$$\hat{e}(g, \sigma_1) = \hat{e}(\sigma_2, H(m)) \cdot \hat{e}(g_1, g_2)$$

which can be rewritten as

$$\hat{e}(g_1, g_2) = \hat{e}(g, \sigma_1) \cdot \hat{e}(\sigma_2, g^d)^{-1} = \hat{e}(g, \sigma_1 \sigma_2^{-d}) = \hat{e}(g, w^*).$$

Hence, by definition of the witness relation, it holds that $(x, w^*) \in \mathcal{R}^L$. Thus, extractability is also proved for this description of the extractor Extr. □

Note that the scheme above is actually WIUnfExt since the witness is unique, given x and par.

4.2 General Construction

By our results in Section 3.2 any SimExt signature of knowledge is WIUnfExt (if finding witnesses for instances is hard). Theoretically thus, the general construction in [12] based on simulation-sound NIZKPoK is also WIUnfExt. We show an alternative construction using the witness-indistinguishable proof systems for any NP language, also called ZAPs [13]. The other ingredients are an IND-CCA public-key encryption scheme ($\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}$) and a pseudorandom generator G. For space reasons we merely sketch the construction and discuss its security.

Construction. Our idea is to add into par a public key pk of the encryption scheme, a random string z of length $2k$ (whose purpose becomes clear later), and a string $\mathsf{par}_{\mathrm{ZAP}}$ for the ZAP. To sign message m with respect to witness w for x let the signer encrypt the witness w together with m to $C = \mathsf{Enc}(pk, w||m)$ and append a ZAP (with respect to $\mathsf{par}_{\mathrm{ZAP}}$) that C encrypts $w||m$ for $(x, w) \in \mathcal{R}^L$ or that z is in the range of G for inputs of length k. We remark that we formally require all witnesses w to be padded to have equal length; this is easy to implement via standard paddings as all witnesses of complexity k are polynomially bounded. The verifier simply checks the validity of the ZAP.

Extractability. The extractability of this scheme can be realized by the decryption algorithm, relying on the fact that a random z is *not* in the range of G with probability at least $1 - 2^{-k}$; thus, the validity of the ZAP implies that the encrypted witness is valid. Proving unforgeability and witness indistinguishability is more sophisticated.

Unforgeability. For unforgeability consider an adversary against the original signature algorithm, being able to produce a valid SoK for a fresh message m with non-negligible probability. As the ZAP is valid and z is not in the range of G with overwhelming probability, running the decryption algorithm on the ciphertext in the adversary's forgery yields a valid witness w with non-negligible probability. Note that we now consider an adversary's success w.r.t. successful extraction of a valid witness, not to a successful forgery. We can further condition on the adversary not outputting a forgery for a previously seen ciphertext; such ciphertexts cannot contain a fresh message.

Change the game slightly by using pseudorandom $z = G(r)$ in par. By the security of G the adversary's success cannot drop significantly. In the next game hop, the altered signing algorithm uses the preimage r to provide valid ZAPs; by the WI of the ZAPs, this negligibly increases the success probability. In the final game, instead of encrypting w in C the again modified signing process uses $0^{|w|}||m$. Note that the ZAP computation is not affected by this, and all witness have the same length. By the IND-CCA security of the encryption scheme, replacing the encryptions of $w||m$ by $0^{|w|}||m$, does not significantly change the adversary's probability of finding a forgery for a *fresh* message; this decrease is easily detected by recovering the witness and message encapsulated in the adversary's forgery attempt (this has to work by IND-CCA and by message —thus ciphertext— freshness). But now the signing oracle is independent of the actual witness; thus the adversary essentially finds a valid witness w to x without help, which is infeasible according to the security of the instance generator.

Parameter Indistinguishability. The public parameters have identical distribution in the actual scheme and the extractability experiment.

Witness Indistinguishability. Again here we can run any distinguisher on fake parameters including a pseudorandom value z, encrypting $\mathsf{Enc}(pk, 0^{|w_b|}||m)$ instead, and using the preimage of z to give a valid ZAP. As for unforgeability it

follows that the behavior of the distinguisher compared to the original signature generation process (for either w_0 or w_1) cannot change significantly. But since the resulting signatures are independent of b, it also follows that the original signatures must be witness indistinguishable.

Simulatability. Actually this construction also achieves the stronger notion of Simulatability. This can be seen from the proof of Witness Indistinguishability and unforgeability, as essentially the initial game is reduced to a game where the signature is independent of the witness. The proof for simulatability would involve a SimSign procedure that does not use a valid witness at all and still outputs a verifiable signature (by using a pseudorandom value z).

4.3 Embedding Witnesses

Both the construction in section 4.2 and the one in [12] instantiate SimExt secure SoKs by encrypting the witness w and a message m, and giving a zero knowledge proof (NIZK) that the encryption is correctly formed and that w and the statement x for which the signature was created are in $\mathcal{R}^L$. This both ensures extractability for w, and it "hides" w, such that the SoK is simulatable. However, in this scheme the NIZK needs to be computed *every time a signature is generated*, as a fresh m must be encrypted every time together with w.

In this section we show how embedding the witnesses into a larger set can improve efficiency such that the proof only needs to be computed once. This, however, undermines witness indistinguishability, which does not hold in general, making the solution inapplicable e.g., to the case of ring signatures. In our construction we again use ZAPs, an IND-CCA public-key encryption scheme $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$, a pseudorandom generator, but this time also an existentially unforgeable signature scheme $\mathsf{Sig} = (\mathsf{SKGen}, \mathsf{SSign}, \mathsf{SVf})$.

Construction. We add into par the public key pk_{Enc} of the encryption scheme, a random string z of length $2k$, and a string $\mathsf{par}_{\mathrm{ZAP}}$ for the ZAP. The main idea for signature generation is that instead of signing messages m for statement-witness pairs (x, w) in the relation $\mathcal{R}^L$, we sign m with respect to an extended witness $\mathsf{W} = (w, s, r)$ and an extended statement $\mathsf{X} = (x, pk_{\mathsf{Sig}}, C, \pi)$, where π is a ZAP (with respect to $\mathsf{par}_{\mathrm{ZAP}}$) that $C = \mathsf{Enc}(pk, \mathsf{W})$ is a correct encryption of W with randomness r such that $(x, w) \in \mathcal{R}^L$, and s is the randomness which made $\mathsf{SKGen}(1^k)$ output pk_{Sig}, or that z is in the range of G for inputs of length k. The signature of knowledge is generated as $\mathsf{SSign}(sk_{\mathsf{Sig}}, m)$. Given m and x the verifier computes verifies the validity of the signature by using pk_{Sig}, and then the validity of the ZAP.

Extractability of Original Witnesses. The extractability of this scheme follows as in the previous section. Note that we extract from the proof for any X only the part of the some witness W^* which comprises a witness w and the randomness s for SKGen; the randomness r for the ciphertext would only be extractable if the encryption scheme were to support randomness recovery, i.e., the decryption algorithm could also be used to derive the randomness r.

Unforgeability. A forgery (m, σ) of a SoK must be output for a fresh message m such that σ verifies under the public key pk_{Sig}. This would straightforwardly violate the unforgeability under the signature scheme, as we can simulate the encryption and the ciphertext C and proof π without knowledge of the randomness s resp. the signing key sk_{Sig} with the same technique as in the construction in the previous section.

Parameter Indistinguishability. The parameters for the extractor and the one in the actual scheme are identically distributed.

5 Application Scenarios

SoKs allow users to sign messages on behalf of any NP statement x; in particular, if there exist more witnesses corresponding to this statement, SoKs naturally provide ring signatures. In this context, the SimExt security of SoKs due to [12] actually guarantees that signatures are simulatable without the witness. However, with our definition of Witness Indistinguishable UnfExt SoKs we ensure that witnesses are merely indistinguishable.

Below we show applications of SoKs to regular digital signatures and ring signatures.

5.1 Digital Signatures

SoKs can be easily used as simple signature schemes as described in e.g. [16] as shown in construction 2.

Construction 2. *Let* $\mathsf{SoK} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Vf})$ *be a UnfExt SoK scheme for a relation* $\mathcal{R}^L$*. Define the signature scheme* $\mathsf{Sig} = (\mathsf{SKGen}, \mathsf{SSign}, \mathsf{SVf})$ *as follows, for some security parameter* k*.*

Key Generation. *On input* k*, algorithm* SKGen *first runs* $\mathsf{Setup}(1^k)$ *and outputs* par *then it runs the instance generator* IGen *for* SoK *on input* par *to obtain a statement/witness keypair* (x, w) *such that* $(x, w) \in \mathcal{R}^L$*.* SKGen *outputs* $(pk = (x, \mathsf{par}), sk = (w, x, \mathsf{par}))$*.*

Signature Generation. *On input message* m *and* $sk = (w, x, \mathsf{par})$*, algorithm* SSign *runs* $\mathsf{Sign}(m, w, x, \mathsf{par})$ *and outputs the resulting SoK* σ *as its signature.*

Signature Verification. *On input signature* σ*, message* m*, and* $pk = (x, \mathsf{par})$*, algorithm* SVf *runs* $\mathsf{Vf}(\sigma, m, x, \mathsf{par})$ *and outputs the resulting bit* b*.*

The security of digital signatures as in [16] is defined in terms of correctness and existential unforgeability against chosen message attacks. The following holds.

Proposition 3. *If* SoK *is UnfExt secure, then construction 2 is a secure digital signature.*

Proof. Correctness is trivially inherited. Furthermore, Existential Unforgeability holds: given an efficient adversary $\mathcal{A}$ outputting forgery $s = \mathsf{SSign}_{sk}(m)$, the adversary $\mathcal{B}$ against SoK Unforgeability uses $\mathcal{A}$ as follows: whenever $\mathcal{A}$ queries SSign on m_i, adversary $\mathcal{B}$ queries Sign on the same input, forwarding the output signature σ. Finally, when $\mathcal{A}$ outputs a forgery (m, s), $\mathcal{B}$ outputs the same, together with $pk = x$. By construction, if $\mathcal{A}$ is successful, then the SoK is valid, thus $\mathcal{B}$ succeeds too. Furthermore, m must be fresh for $\mathcal{B}$, as it is fresh for $\mathcal{A}$. □

5.2 Ring Signatures

Ring signatures were formalised by Rivest, Shamir, and Tauman [21] in 2001. In this setting, a *signer* signs message m on behalf of a so-called *ring* of participants such that it is impossible to tell which ring member actually signed m. We denote ring members by $U_1, U_2, \ldots, U_n$.

Ring signatures assume the existence of a PKI where users U_i are associated with private/public key pairs (sk_i, pk_i). Message m can be signed by U_i under public keys $\{pk_i\}_{i=1}^n$ and under private key sk_i, resulting in a signature σ. Verification requires the public keys of all the users, outputting a bit. In particular, Rivest et al. [21] define ring signatures to be setup-free, i.e. any signer can dynamically select a ring just by knowing the public keys of the other ring members.

We adopt the ring signature definitions due to Bender et al. [3]. This work defines rings of n of users to be the subset of their public keys, which may be honestly generated or chosen by the adversary. In the notation of [3], we write $R = (pk_1, \ldots, pk_n)$ for the ring of users U_j with $j \in \{1, \ldots, n\}$.

Definition 5 (Ring Signatures [3]). *A* ring signature *is a tuple of efficient algorithms* $\mathsf{RSig} = (\mathsf{RSKGen}, \mathsf{RSSign}, \mathsf{RSVf})$ *such that:*

KEY GENERATION. *Run on security parameter* k, RSKGen *outputs key-pair* (sk, pk).
SIGNATURE GENERATION. *On input index* i, *message* m, *ring* R *of size* n *with* n *distinct elements, and* sk *s.t.* (sk, pk_i) *is a legitimate key-pair.*
SIGNATURE VERIFICATION. *On input* (R, m, σ), *algorithm* RSVf *returns bit* b.

We require perfect completeness, i.e., for all k, *for all* n *key-pairs* (sk_i, pk_i) *for* $i \in \{1, \ldots, n\}$, *any* $j \in \{1, \ldots, n\}$, *and any message* m, *it holds that* $\mathsf{RSVf}(R, m, \mathsf{RSSign}(j, sk_j, m, R)) = 1$ *for* $R = (pk_1, \ldots, pk_n)$.

Ring signatures have two main properties: anonymity and unforgeability. Bender et al. [3] introduce various degrees of these notions and prove strict implications between the different flavors. The adversary may query an OSign oracle with input an index j, a message m, and a ring R, and running $\mathsf{RSSign}(j, m, R, sk_j)$ to obtain the honestly generated signature σ. We reiterate only the strongest form of anonymity – anonymity against attribution attacks – and the basic-most form of unforgeability – unforgeability against fixed-rings attacks and call them anonymity, resp. unforgeability. Intuitively, this form of anonymity allows

the adversary to know the secret keys of all-but-one users, but it still can't distinguish the signer of a message (i.e., if even a single signer is honest, a signature cannot be attributed to him, even by everyone else colludes together). In unforgeability against fixed-rings attacks, the signature is unforgeable if the adversary uses the same ring of users. This is equivalent, as we see below, to using the same statement x for SoKs. For further details on ring signatures, please refer to [3]. Ring signature security now follows:

ANONYMITY. In order to formalize anonymity, Bender et al. allow the adversary to learn the randomness used by RSKGenin generating the users' key pairs. Instead, we give the adversary access to all the honestly generated secret keys after they have been generated. Another subtle difference in our definition allows our adversary to be stronger: Bender et al. give the adversary access to the secret keys only *after* the adversary has chosen a challenge message m, indices i_0 and i_1, and a ring R such that pk_{i_0}, pk_{i_1} are in R. In our definition, the adversary may know the secret keys even before it has made its choice. We call the ring signature scheme anonymous iff for all integers n (depending on k), and all efficient adversaries $\mathcal{A}$, the following holds:

$$\begin{aligned}\text{Prob } [(sk_i, pk_i)_{i=1}^n \leftarrow \mathsf{RSKGen}(1^k); (\mathsf{st}, i_0, i_1, m, R) \leftarrow \mathcal{A}^{\mathsf{OSign}(\cdot,\cdot,\cdot)}((sk_i, pk_i)_{i=1}^n) \\ b \leftarrow \{0,1\}; \sigma \leftarrow \mathsf{RSSign}(i_b, m, R, sk_{i_b}); d \leftarrow \mathcal{A}^{\mathsf{OSign}(\cdot,\cdot,\cdot)}(\sigma, \mathsf{st}) : d = b] \approx \tfrac{1}{2}.\end{aligned}$$

UNFORGEABILITY. For all security parameters k, all integers n, and all efficient adversaries $\mathcal{A}$, the following holds:

$$\begin{aligned}\text{Prob } [(sk_i, pk_i)_{i=1}^n \leftarrow \mathsf{RSKGen}(1^k); (m, \sigma) \leftarrow \mathcal{A}^{\mathsf{OSign}(\cdot,\cdot,R)}(\{pk_i\}_{i=1}^n) : \\ (\cdot, m) \notin Q \wedge \mathsf{RSVf}(R, m, \sigma) = 1 \qquad\qquad] \approx 0.\end{aligned}$$

Here we denote by Q the list of queries made to the OSign oracle.

Note that, although this definition of anonymity is the strongest of the three presented by Bender et al. [3], it is not as strong as the simulatability property required by signatures of knowledge as defined by Chase and Lysyanskaya. And yet, ring signatures can be constructed from SoKs in a natural way, as long as there exists a form of witness indistinguishability. Indeed, a signature of knowledge on a message m proves that a signer who knows *a* valid witness (out of possibly many valid witnesses) to a statement has signed a statement. In fact, if we equivalate a ring to a statement, we can perceive the set of witnesses belonging to this set as each representing a user in the ring. We describe this in what follows.

We first consider a relation $\mathcal{R}^L$ with statements of the form x and witnesses w and with an efficient instance generator IGen, which, on input a security parameter k and some parameters par, outputs a statement x and a witness w with $(x, w) \in \mathcal{R}^L$. Let $\mathsf{SoK} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Vf})$ be a witness indistinguishable signature of knowledge for a relation $\mathcal{R}$ with statements $R = (x_1, \ldots, x_n)$ and witnesses w such that $(w, R) \in \mathcal{R}$ iff. there exists an index $j \in \{1, \ldots, n\}$ such

that $(w, x_j) \in \mathcal{R}^L$. Consider a ring signature RSig= (RSKGen, RSSign, RSVf), such that:

Setup. Before running the ring signature scheme, the algorithm Setup is run on input k to output parameters par.

Key Generation. Upon input a security parameter $K = (k, \mathsf{par})$ for an integer k and parameters par, the key generation algorithm RSKGen runs IGen, outputting the tuple (x_i, w_i).

Ring Signature generation. Upon input an index $i \in \{1, \ldots, n\}$, a message m, a ring $R = (x_1, \ldots, x_n)$, and a private key w_i, the signature generation algorithm RSSign runs Sign on input m, R, w, par, and returns the output signature σ.

Ring Signature verification. Upon input a ring $R = (x_1, \ldots, x_n)$, a message m, and a signature σ, the signature verification algorithm RSVf runs Vf on input $\sigma, m, R, \mathsf{par}$, and outputs the resulting bit b.

This construction is a secure ring signature in the sense of the above security definition. In particular, the completeness property follows from the correctness of the underlying signature of knowledge scheme, and unforgeability follows from the unforgeability of the SoK (but the ring is fixed, as the definition of unforgeability for SoKs fixes the statement, in this case R). To see that RSig is also anonymous in the presence of attributions, note that the witness indistinguishability definition is quantified over all witnesses, which are freely given to the adversary. Therefore the adversary knows all w_i, but cannot tell them apart anyway.

Acknowledgments

We thank the anonymous reviewers for constructive and valuable comments.

References

1. Adams, C., and Farrell, S. Internet x.509 public key infrastructure certificate. RFC 2510, March 2009.
2. Bellare, M., and Goldreich, O. On defining proofs of knowledge. In *Advances in Cryptology — Crypto '92* (1992), vol. 740 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 390–420.
3. Bender, A., Katz, J., and Morselli, R. Ring signatures: Stronger definitions, and constructions without random oracles. In *Theory of Cryptography Conference (TCC) '06* (2006), vol. 3876 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 60–79.
4. Blum, M., Feldman, P., and Micali, S. Non-interactive zero-knowledge and its applications (extended abstract). In *Proceedings of the Annual Symposium on the Theory of Computing (STOC)* (1988), ACM Press, pp. 103–112.
5. Boldyreva, A. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *Public-Key Cryptography (PKC)* (2003), vol. 2567 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 31–46.

6. Bresson, E., and Stern, J. Efficient revocation in group signatures. In *Public Key Cryptography* (2001), vol. 1992 of *Lecture Notes in Computer Science*, Springer, pp. 190–206.
7. Camenisch, J. Efficient and generalized group signatures. In *Advances in Cryptology — Eurocrypt* (1997), Lecture Notes in Computer Science, Springer-Verlag, pp. 465–479.
8. Camenisch, J., and Stadler, M. Efficient group signature schemes for large groups (extended abstract). In *CRYPTO* (1997), vol. 1294 of *Lecture Notes in Computer Science*, Springer, pp. 410–424.
9. Canetti, R. Security and composition of multi-party cryptographic protocols. In *Journal of Cryptology* (2000), vol. 13, Springer-Verlag, pp. 143–202.
10. Canetti, R. Universally composable security: a new paradigm for cryptographic protocols. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)* (2001), IEEE Computer Society Press, pp. 136 – 145.
11. Canetti, R. Universally composable security: A new paradigm for cryptographic protocol. Cryptology ePrint Archive, Report 2000/067, 2005. EPRINTURL.
12. Chase, M., and Lysyanskaya, A. On signatures of knowledge. In *Advances in Cryptology — Crypto '06* (2006), vol. 4117 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 78 – 96.
13. Dwork, C., and Naor, M. Zaps and their applications. *SIAM J. Comput. 36*, 6 (2007), 1513–1543.
14. Feige, U., and Shamir, A. Witness indistinguishable and witness hiding protocols. In *Proceedings of the Annual Symposium on the Theory of Computing (STOC)* (1990).
15. guang Zou, X., and Sun, S.-H. Analysis of anonymity on the signatures of knowledge. In *IIH-MSP* (2006), IEEE Computer Society, pp. 621–624.
16. Katz, J. *Digital Signatures*. Springer-Verlag, 2010.
17. Lu, S., Ostrovsky, R., Sahai, A., Shacham, H., and Waters, B. Sequential aggregate signatures and multisignatures without random oracles. In *Advances in Cryptology — Eurocrypt* (2006), vol. 4004 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 465–485.
18. Mykletun, E., Narasimha, M., and Tsudik, G. Signature bouquets: Immutability for aggregated/condensed signatures. In *ESORICS* (2004), vol. 3193 of *Lecture Notes in Computer Science*, Springer, pp. 160–176.
19. Prafullchandra, H., and Schaad, J. Diffie-Hellman proof-of-possession algorithms. RFC 2875, July 2000.
20. Ristenpart, T., and Yilek, S. The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks. In *Advances in Cryptology — Eurocrypt 2007* (2007), vol. 4515 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 228–245.
21. Rivest, R. L., Shamir, A., and Tauman, Y. How to leak a secret. In *Advances in Cryptology — Asiacrypt 2001* (2001), vol. 2248 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 552 – 565.
22. Shahandashti, S. F., and Safavi-Naini, R. Construction of universal designated-verifier signatures and identity-based signatures from standard signatures. In *Public Key Cryptography* (2008), vol. 4939 of *Lecture Notes in Computer Science*, Springer, pp. 121–140.
23. Waters, B. Efficient identity-based encryption without random oracles. In *Advances in Cryptology — Eurocrypt '05* (2005), vol. 1853 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 114–127.