Length-Hiding Redactable Signatures from One-Way Accumulators in $\mathcal{O}(n)$

Henrich C. Pöhls, Kai Samelin, Joachim Posegga and Hermann de Meer
{hp,ks,jp}@sec.uni-passau.de, demeer@uni-passau.de
Institute of IT-Security and Security-Law (ISL), University of Passau, Germany

Technical Report, Number MIP-1201
Department of Informatics and Mathematics
University of Passau, Germany
April 2012

Length-Hiding Redactable Signatures from One-Way Accumulators in $\mathcal{O}(n)$

Henrich C. Pöhls[*], Kai Samelin[**], Joachim Posegga, Hermann de Meer

Institute of IT-Security and Security-Law (ISL), University of Passau, Germany
{hp,ks,jp}@sec.uni-passau.de, demeer@uni-passau.de

Abstract. In this paper, we propose two provably secure and length-hiding redactable signature schemes. Both have a runtime complexity of $\mathcal{O}(n)$. This is lower than existing schemes, which have at best $\mathcal{O}(n \cdot log(n))$. Our first scheme protects the integrity of the elements in unordered (multi-)sets, with a storage complexity of $\mathcal{O}(1)$. Our second construction protects the structural relation of ordering of elements in lists, with a storage complexity of $\mathcal{O}(n)$. We build on a family of quasi-commutative accumulators and a family of hash-functions based on non-abelian but associative operations.

Keywords: Redactable Signatures, Transparency, Hash-Functions

1 Introduction

1.1 Contribution and Outline

We present two runtime efficient ($\mathcal{O}(n)$) secure schemes. When information is stored within a data structure one must take the information stored inside the structure itself into account. In our case the structure of the message stores the order of message blocks. We introduce a precise formal model for both types of RSS in Sect. 2. Our security model is as strong as *Brzuska* et al.'s [5]. We discuss the difference to *Ahn* et al.'s security model in Sect. 2.

To ease explanation our first scheme RSS_S ignores structure and protects an unordered multi-set of blocks. We will use the notion "set" instead of "multi-set" for the remaining part of this paper. Our second scheme RSS_D protects the order as a one possible structural relation between any two blocks. Both constructions make use of pre-image- and 2^{nd}-pre-image-resistant one-way accumulators. For sets, we use a quasi-commutative hash-function, while for ordered documents, a hash-function with associative but non-abelian operations will be utilized. The final digest is signed with an unforgeable signature scheme like RSA-OAEP [3; 18].

[*]Is funded by BMBF (FKZ:13N10966) and ANR as part of the ReSCUeIT project
[**]Is funded by "Regionale Wettbewerbsfähigkeit und Beschäftigung", Bayern, 2007-2013(EFRE) as part of the SECBIT project. (http://www.secbit.de)

Existing efficient constructions that rely on the idea of using accumulators do not meet our strong privacy requirements [8; 12]. An efficient and secure construction by *Miyazaki* et al. is not useable for multi-sets [13]. The secure construction of *Chang* et al. requires $\mathcal{O}(n^2)$ operations [9], and the secure construction of *Ahn* et al. still requires $\mathcal{O}(log(n) \cdot n)$ operations [1]. Our schemes both have a runtime complexity of only $\mathcal{O}(n)$, where n denotes the number of blocks. The storage complexity for our first scheme for multi-sets is $\mathcal{O}(1)$, while the second scheme for ordered lists needs $\mathcal{O}(n)$.

The schemes are presented in detail in Sect. 3 and in Sect. 4, including a discussion of their storage and runtime complexities. All formal proofs can be found in appendices.

2 Preliminaries and Definitions

We build upon the model introduced by *Brzuska* et al. in [5]. However, we need to adjust the concrete definitions, since we are discussing sets and linear documents and not trees. The introduced security model is rigid, i.e., it is more restrictive than the one introduced in the original papers [11; 20]. It is not as restrictive as *Ahn* et al. [1]. We think that the security mode introduced by *Brzuska* et al. in [5] is sufficient for most use cases, while the one introduced in [1] by *Ahn* et al. seems to be overly strong, if it comes to real world applications. In particular, we do not require unlinkability [1; 7], a even stronger privacy notation.

We first define the required algorithms for sets:

Definition 1 (*RSS* Algorithms for Sets). *A RSS for sets consists of four efficient algorithms $\mathcal{RSS}_S := (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Redact})$ such that:*

KeyGen. *The algorithm KeyGen outputs the required key pair, i.e.,* $(\mathrm{pk}, \mathrm{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$*, where λ is the security parameter.*

Sign. *The algorithm Sign outputs the signature σ on input of the secret key* sk *and the set S. It outputs $(S, \sigma) \leftarrow \mathsf{Sign}(\mathrm{sk}, S)$*

Verify. *The algorithm Verify outputs a bit $d \in \{0, 1\}$ indicating the correctness of the signature σ, w.r.t.* pk*, protecting the set S. In particular: $d \leftarrow \mathsf{Verify}(\mathrm{pk}, S, \sigma)$*

Redact. *The algorithm Redact takes as input the set S, the public key* pk *of the signer, the signature σ and a subset $\diamond \subseteq S$, which denotes the blocks to be redacted. The algorithm outputs $(S', \sigma') \leftarrow \mathsf{Redact}(\mathrm{pk}, \sigma, S, \diamond)$, where $S' = S \setminus \diamond$*

We assume that the blocks v_i can efficiently be derived from the set S. Next, we define the required algorithms for linear documents. Here, we also require that the splitting of m into the blocks $m[i]$, along with their ordering, can efficiently and uniquely be derived from any received m.

Definition 2 (*RSS* **Algorithms for Linear Documents).** *A RSS for linear documents consists of four efficient algorithms. In particular,* $\mathcal{RSS}_D := (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Redact})$ *such that:*

KeyGen. *The algorithm KeyGen outputs the public and private key of the signer, i.e.,* $(\mathrm{pk}, \mathrm{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$*, where* λ *is the security parameter.*

Sign. *The algorithm Sign outputs the signature* σ *on input of the secret key* sk *and the document* m*. It outputs* $(m, \sigma) \leftarrow \mathsf{Sign}(\mathrm{sk}, m)$

Verify. *The algorithm Verify outputs a bit* $d \in \{0,1\}$ *indicating the correctness of the signature* σ*, w.r.t.* pk*, protecting* m*. In particular:* $d \leftarrow \mathsf{Verify}(\mathrm{pk}, m, \sigma)$

Redact. *The algorithm Redact takes* m*, the public key* pk *of the signer, the signature* σ *and an index set* $\mathcal{I}$ *which denotes the blocks* $m[i]$ *to be redacted. The algorithm outputs* $(m', \sigma') \leftarrow \mathsf{Redact}(\mathrm{pk}, \sigma, m, \mathcal{I})$*, where* m' *is the altered list. We denote an alteration of* m *w.r.t.* $\mathcal{I}$ *as* $m' \leftarrow \textsc{mod}(m, \mathcal{I})$

In particular, the following soundness requirements have to hold: Every document generated using Sign has to verify with overwhelming probability, i.e., for any security parameter λ, any key pair $(pk, sk) \leftarrow \mathsf{KeyGen}(\lambda)$, and any set/list S/m, any $(S/m, \sigma) \leftarrow \mathsf{Sign}(sk, S/m)$ we have $\mathsf{Verify}(pk, S/m, \sigma) = 1$. For Redact we require the same, i.e., for any security parameter λ, any key pair $(pk, sk) \leftarrow \mathsf{KeyGen}(\lambda)$, and any set/list S/m, any $(S/m, \sigma) \leftarrow \mathsf{Sign}(sk, S/m)$ and for any subset $\diamond$ resp. for any blocks indexed by $\mathcal{I}$, we have $(S'/m', \sigma') \leftarrow \mathsf{Redact}(pk, \sigma, S/m, \diamond/\mathcal{I})$ and require $\mathsf{Verify}(pk, S'/m', \sigma') = 1$. Moreover, a document/signature pair created using Redact can be subject to Redact again, hence allowing to delegate the sanitization to a consecutive third party.

All possible valid redactions, forming the transitive closure of a message m, w.r.t. Redact, are denoted as $\mathrm{span}_{\vdash}(m)$, following [9] and [19].

2.1 Security Properties and Security Model

Next, we will define the required security properties of *RSS*. These have already been identified in [5] for trees. We adapt and modify their notion to the needs of sets and linear documents. We have already given informal definitions in the introduction, they are suitable for both, sets and linear documents, while the formal definitions, given as adversarial games, slightly differ. Hence, we will give both notations for readability.

Unforgeability. No one should be able to compute a valid signature verifying under pk on a set or list outside the transitive closure $\mathrm{span}_{\vdash}(S/m)$ without having access to the corresponding secret key sk, even when allowed to request signatures on different sets resp. documents. We say that a scheme *RSS* for sets, resp. linear documents, is unforgeable, iff for any efficient (PPT) adversary

```
**Experiment** $\mathsf{Unforgeability}_{\mathcal{A}}^{\mathsf{RSS}_S}(\lambda)$
$(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
$(S^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(sk,\cdot)}(pk)$
    let $i = 1, 2, \ldots, q$ index the queries
return 1 iff
    $\mathsf{Verify}(pk, S^*, \sigma^*) = 1$ and
    $\forall i : 1 \leq i \leq q,\ S^* \nsubseteq S_i$
```

Fig. 1. Unforgeability for Sets

```
**Experiment** $\mathsf{Unforgeability}_{\mathcal{A}}^{\mathsf{RSS}_D}(\lambda)$
$(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
$(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(sk,\cdot)}(pk)$
    let $i = 1, 2, \ldots, q$ index the queries
return 1 iff
    $\mathsf{Verify}(pk, m^*, \sigma^*) = 1$ and
    $\forall i : 1 \leq i \leq q,\ m^* \notin \mathrm{span}_{\vdash}(m_i)$
```

Fig. 2. Unforgeability for Documents

```
**Experiment** $\mathsf{Privacy}_{\mathcal{A}}^{\mathsf{RSS}_S}(\lambda)$
$(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
$b \xleftarrow{\$} \{0, 1\}$
$d \leftarrow \mathcal{A}^{\mathsf{Sign}(sk,\cdot),\mathsf{LoRRedact}(\ldots,sk,b)}(pk)$
    where oracle $\mathsf{LoRRedact}$ for input $S_0, S_1, \diamond_0, \diamond_1$:
        if $S_0 \setminus \diamond_0 \neq S_1 \setminus \diamond_1$, return $\perp$
        $(S, \sigma) \leftarrow \mathsf{Sign}(sk, S_b)$
    return $(S', \sigma') \leftarrow \mathsf{Redact}(pk, \sigma, S, \diamond_b)$.
return 1 iff $b = d$
```

Fig. 3. Privacy for Sets

```
**Experiment** $\mathsf{Privacy}_{\mathcal{A}}^{\mathsf{RSS}_D}(\lambda)$
$(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
$b \xleftarrow{\$} \{0, 1\}$
$d \leftarrow \mathcal{A}^{\mathsf{Sign}(sk,\cdot),\mathsf{LoRRedact}(\ldots,sk,b)}(pk)$
    where oracle $\mathsf{LoRRedact}$ for input $m_0, m_1, \mathcal{I}_0, \mathcal{I}_1$:
        if $\textsc{mod}(m_0, \mathcal{I}_0) \neq \textsc{mod}(m_1, \mathcal{I}_1)$, return $\perp$
        $(m, \sigma) \leftarrow \mathsf{Sign}(sk, m_b)$
    return $(m', \sigma') \leftarrow \mathsf{Redact}(pk, \sigma, m_b, \mathcal{I}_b)$.
return 1 iff $b = d$
```

Fig. 4. Privacy for Documents

$\mathcal{A}$, the probability that the game depicted in Fig. 1, resp. Fig. 2, returns 1 is negligible (as a function of λ).

Privacy. An adversary should not be able derive any additional information besides what can be derived from the received message/signature pair. We say that a scheme *RSS* for sets, resp. linear documents, is private, iff for any efficient (PPT) adversary $\mathcal{A}$, the probability that the game depicted in Fig. 3, resp. Fig. 4, returns 1 is negligibly close to $\frac{1}{2}$ (as a function of λ).

Transparency. The verifier should not be able to decide whether a signature has been created by the signer, or through the redaction algorithm Redact. From a signed document one cannot tell whether it is a freshly signed version, where some blocks have been removed prior to signing, or a blinded version. We say that a scheme *RSS* for sets, resp. linear documents, is transparent, iff for any efficient (PPT) adversary $\mathcal{A}$, the probability that the game depicted in Fig. 5, resp. Fig. 6, returns 1 is negligibly close to $\frac{1}{2}$ (as a function of λ).

Proposition 1 (Transparency $\implies$ Privacy). *There exists no scheme which is transparent, but not private.*

Proposition 2 (Privacy $\not\Rightarrow$ Transparency). *There exists a scheme which is unforgeable and private, but not transparent.*

```
**Experiment** $\mathsf{Transparency}_{\mathcal{A}}^{\mathsf{RSS}_S}(\lambda)$
$(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
$b \xleftarrow{\$} \{0,1\}$
$d \leftarrow \mathcal{A}^{\mathsf{Sign}(sk,\cdot),\mathsf{Sign/Redact}(\ldots,sk,b)}(pk)$
    where oracle $\mathsf{Sign/Redact}$ for input $S, \diamond$:
        if $\diamond \notin S$, return $\perp$
        if $b = 0$:   $(S, \sigma) \leftarrow \mathsf{Sign}(sk, S)$,
                      $(S', \sigma') \leftarrow \mathsf{Redact}(pk, \sigma, S, \diamond)$
        if $b = 1$:   $S' \leftarrow S \setminus \diamond$
                      $(S', \sigma') \leftarrow \mathsf{Sign}(sk, S')$,
    finally return $(m', \sigma')$.
return 1 iff $b = d$
```

Fig. 5. Transparency for Sets

```
**Experiment** $\mathsf{Transparency}_{\mathcal{A}}^{\mathsf{RSS}_D}(\lambda)$
$(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
$b \xleftarrow{\$} \{0,1\}$
$d \leftarrow \mathcal{A}^{\mathsf{Sign}(sk,\cdot),\mathsf{Sign/Redact}(\ldots,sk,b)}(pk)$
    where oracle $\mathsf{Sign/Redact}$ for input $m, \mathcal{I}$:
        if $m[i] \not\subseteq m$, return $\perp$
        if $b = 0$:   $(m, \sigma) \leftarrow \mathsf{Sign}(sk, m)$,
                      $(m', \sigma') \leftarrow \mathsf{Redact}(pk, \sigma, m, \mathcal{I})$
        if $b = 1$:   $m' \leftarrow \textsc{mod}(m, \mathcal{I})$
                      $(m', \sigma') \leftarrow \mathsf{Sign}(sk, m')$,
    finally return $(m', \sigma')$.
return 1 iff $b = d$
```

Fig. 6. Transparency for Documents

Proposition 3 (Unforgeability is Independent). *There exists a scheme which is transparent and private, but not unforgeable and vice versa.*

Proof. The formal proofs for Prop. 1, Prop. 2 and Prop. 3 are in the appendices.

Definition 3 (Cryptographic Hash-Function). *Let $\mathcal{H}_K : \{0,1\}^* \rightarrow \{0,1\}^\lambda$ denote a family of keyed hash-functions mapping input of arbitrary length onto an output with fixed length λ, where λ denotes the security parameter. In this paper, we call $\mathcal{H}_k \in \mathcal{H}_K$ a cryptographic hash-function, iff the following properties hold:*

Collision Resistance. *It is computationally infeasible to find m, m', such that $\mathcal{H}_k(m) = \mathcal{H}_k(m')$. More formally:*

$$\Pr[k \xleftarrow{\$} K; (x, x') \leftarrow \mathcal{A}(k) : x \neq x' \wedge \mathcal{H}_k(x) = \mathcal{H}_k(x')] < \epsilon(\lambda)$$

2nd Preimage Resistance. *It is computationally infeasible to find a x' for a given x, such that $\mathcal{H}_k(x) = \mathcal{H}_k(x')$, where $x \neq x'$. More formally:*

$$\Pr[k \xleftarrow{\$} K, x \xleftarrow{\$} \{0,1\}^*; x' \leftarrow \mathcal{A}(k, x) : x \neq x' \wedge \mathcal{H}_k(x) = \mathcal{H}_k(x')] < \epsilon(\lambda)$$

Preimage Resistance. *It is computationally infeasible to find a m, for a given value $h \xleftarrow{\$} \{0,1\}^\lambda$, such that $\mathcal{H}_k(m) = h$. More formally:*

$$\Pr[k \xleftarrow{\$} K; h \xleftarrow{\$} \{0,1\}^\lambda; x \leftarrow \mathcal{A}(k, h)) : h = \mathcal{H}_k(x)] < \epsilon(\lambda)$$

Above probabilities are taken over all coin tosses and must be negligible as a function of λ. We require efficient computation and a compression of the input. One example for such a cryptographic hash-function family is *SHA*. *Benaloh* et al. introduced quasi-commutative hash-functions in [4]:

Definition 4 (Quasi-Commutative Hash-Function ($\mathcal{QCHF}$)). *A hash-function family $\mathcal{QCHF}_K$ is called quasi-commutative, iff:*

```
**Experiment** Resistance$^{\mathcal{QCHF}}_{\mathcal{A}}(\lambda, n)$
$k \xleftarrow{\$} K$
$d \xleftarrow{\$} \mathcal{X}_k$
$o = \{o_1, o_2, \ldots, o_n \mid o_i \xleftarrow{\$} \mathcal{Y}_k\}$
$r \xleftarrow{\$} \mathcal{X}_k$
$(r^*, o^*) \leftarrow \mathcal{A}(k, n, o, d, r)$
return 1 iff
    $\mathcal{QCHF}_k(r^*, o^*) = \mathcal{QCHF}_k(r, o)$ or
    $\mathcal{QCHF}_k(r^*, o^*) = d$ and in all cases
    $o^* \notin \text{span}(o)$
```

Fig. 7. Experiment $\mathcal{QCHF}$

```
**Experiment** Resistance$^{\mathcal{AHF}}_{\mathcal{A}}(\lambda, n)$
$k \xleftarrow{\$} K$
$o = (o_1, o_2, \ldots, o_n \mid o_i \xleftarrow{\$} \mathcal{X}_k\}$
$o^* \leftarrow \mathcal{A}(k, n, o)$
return 1 iff
    $\mathcal{AHF}_k(o^*) = \mathcal{AHF}_k(o)$ and
    $o^* \notin \text{span}(o)$
```

Fig. 8. Experiment $\mathcal{AHF}$

Quasi-Commmutativity. $\forall k \in K : \mathcal{X}_k \times \mathcal{Y}_k \to \mathcal{X}_k$.

Additionally: $\forall k \in K : \forall x \in \mathcal{X}_k : \forall y_1, y_2 \in \mathcal{Y}_k : \mathcal{QCHF}_k(\mathcal{QCHF}_k(x, y_1), y_2) = \mathcal{QCHF}_k(\mathcal{QCHF}_k(x, y_2), y_1)$. *We will denote the accumulation of* $\{v_1, \ldots, v_n\}$ *(in arbitrary order), along with a seed* $r \in \mathcal{X}_k$, *as* $\mathcal{QCHF}_K(r, v_1; \ldots; v_n)$.

Security. *For a secure instantiation of a* $\mathcal{QCHF}_k \in \mathcal{QCHF}_K$, *we require the probability that the experiment depicted in Fig. 7 returns* 1 *to be negligible (as a function of* λ*) for any efficient (PPT) adversary* $\mathcal{A}$, *where span*(o) *is the transitive closure of* $\mathcal{QCHF}_k$, *i.e., the set span*(o) *contains all sets which can be generated by sequentially running* $\mathcal{QCHF}_k$ *given* o. *This prohibits winning the game using intended collisions, i.e., the output of valid redactions is not a forgery. This definition captures the required form of pre-image- and* 2^{nd}*-pre-image-resistance. Moreover, we require that an adversary* $\mathcal{A}$ *cannot decide how many additional members have been digested, i.e., the following distributions* S_1 *and* S_2 *must be computationally indistinguishable:*

$$S_1 = \{x \mid x \xleftarrow{\$} \mathcal{X}_k, k \xleftarrow{\$} K\} \quad S_2 = \{\mathcal{QCHF}_k(x, y) \mid y \xleftarrow{\$} \mathcal{Y}_k, x \xleftarrow{\$} \mathcal{X}_k, k \xleftarrow{\$} K\}$$

The probability is taken over all coin tosses. To sum up, an outsider cannot distinguish between random elements and accumulated digests.

Instantiation 1 (Quasi-Commmutativity using Modular Exponentiation) *A hash-function* $\mathcal{QCHF}_k \in \mathcal{QCHF}_K$ *can be constructed in the following way: [4]*

Choose Parameter. *Let* $m = pq$, *where* $p = 2p' + 1, q = 2q' + 1, q, p, q', p' \in \mathbb{P}$ *and* $r \xleftarrow{\$} (\mathbb{Z}/n\mathbb{Z})^\times$.

Hashing. *On input of* $r, y_1, \ldots, y_m$ *we compute* $d \leftarrow r^{\prod_{i=1}^{i=n} \mathcal{H}_p(y_i)} \pmod{m}$, *where* $\mathcal{H}_p : \{0,1\}^* \to \mathbb{P} \cap \mathbb{Z}_{m/4}$ *is a cryptographic hash-function, modeled as a random oracle, and* n *the number of blocks. Refer to [2] how to construct such a function.*

Hence, m is a RSA-modulus with safe primes. This hash function is quasi-commutative and allows to aggregate input, while producing a short digest [4].

In [4] witnesses can be calculated, which are y^{th}-roots, to allow proving membership for digested values. Our construction does not require witnesses. The security proofs for collision-resistance and one-wayness of *Benaloh* et al.'s instantiation of a $\mathcal{QCHF}_k \in \mathcal{QCHF}_K$ can also be found in [4]. The homomorphic behavior of the RSA-function allows to forge digests in a simple way, i.e., by multiplying elements. The solution against this attack is to hash the input prior to aggregating it, i.e., $\mathcal{H}_p$ prohibits this simple attack. [2] contains the proofs why this function meets our security requirements.

Definition 5 (Associative Non-Abelian Hash-Functions ($\mathcal{AHF}_K$)). *The family of Associative Non-Abelian Hash-Functions, denoted as $\mathcal{AHF}_K$, is also accumulating. However, the ordering of its input becomes relevant:*

Associative but Non-Commutative. $\forall k \in K : \mathcal{X}_k \times \mathcal{Y}_k \to \mathcal{X}_k$ *and* $\forall k \in K : \forall x \in \mathcal{X}_k : \forall y_1, y_2, y_1 \neq y_2 \in \mathcal{Y}_k : \mathcal{AHF}_k(\mathcal{AHF}_k(x, y_1), y_2) \neq \mathcal{AHF}_k(\mathcal{AHF}_k(x, y_2), y_1)$[1] *but* $\forall k \in K : \forall y_1, y_2, y_3 \in \mathcal{Y}_k : \mathcal{AHF}_k(\mathcal{AHF}_k(y_1, y_2), y_3) = \mathcal{AHF}_k(y_1, \mathcal{AHF}_k(y_2, y_3))$. *This implies $\mathcal{X}_k = \mathcal{Y}_k$. We will denote the accumulation of $v_1, \ldots, v_n$ (in that particular order) as $\mathcal{AHF}_K(v_1; \ldots; v_n)$.*

Security. *For a secure instantiation of a $\mathcal{AHF}_k \in \mathcal{AHF}_K$, we require the probability that the experiment depicted in Fig. 8 returns* 1 *to be negligible (as a function of λ) for any efficient (PPT) adversary $\mathcal{A}$. This definition captures the required form of pre-image- and 2nd-pre-image-resistance. As before, span(o) is the transitive closure of $\mathcal{AHF}_k$, i.e., the set span(o) contains all lists of elements which can be generated by sequentially running $\mathcal{AHF}_k$ given o. Again, this prohibits the adversary from winning the game using intended collisions. Moreover, we require that an adversary cannot decide how many additional members have been digested, i.e., the following distributions S_1 and S_2 must be computationally indistinguishable:*

$$S_1 = \{x \mid x \overset{\$}{\leftarrow} \mathcal{X}_k, k \overset{\$}{\leftarrow} K\} \quad S_2 = \{\mathcal{AHF}_k(x; y) \mid x, y \overset{\$}{\leftarrow} \mathcal{X}_k, k \overset{\$}{\leftarrow} K\}$$

The probability is taken over all coin tosses. This prohibits an adversary from distinguishing between random elements and accumulated digests.

An instantiation of an $\mathcal{AHF}_k$ is the *Tillich-Zémor* ($\mathcal{TZ}$) construction based on $SL(2, \mathbb{F}_2[X]/P_r(X))$ [21].[2] The generators used in [21] have been broken in [10] and [14]. However, the authors of [14] give new generators in [15] that rely on a different underlying mathematical problem and are secure. We will use the secure parameters in our instantiation of $\mathcal{AHF}_k$. Our second construction RSS_D (Sect. 4) relies only on the existence of $\mathcal{AHF}_K$. We use $\mathcal{TZ}$ as an instantiation in our examples to show our ideas in a simple and understandable way.

Instantiation 2 (Hashing using SL2) *A hash-function $\mathcal{AHF}_k \in \mathcal{AHF}_K$ can be constructed in the following way: [21; 15]*

[1] To be more precise: With overwhelming probability.
[2] $SL(2, \mathbf{R})$ denotes the group of all 2×2 matrices with determinant one.

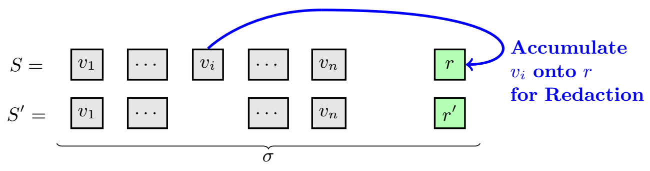

Fig. 9. Illustration of Construction 1

Choose Parameter. *Let* $r \in \{127, 157, 223, 251, 383, 509\}$.[3] *Choose an irreducible polynomial* $P_r(X)$ *with degree* r.

Hashing. *Let* A *and* B *be two matrices, i.e.,* $A = \left(\begin{smallmatrix} X^3 & 1 \\ 1 & 0 \end{smallmatrix}\right)$ *and* $B = \left(\begin{smallmatrix} X+1 & 1 \\ 1 & 0 \end{smallmatrix}\right)$. *Afterwards, define the mapping* $\pi := \{0,1\} \to A, B$, *where* $0 \mapsto A, 1 \mapsto B$. *Let the message* m *to be digested written in binary, in particular* $m = b_1 || \ldots || b_n$. *Define the digest* d *as:* $d \leftarrow \pi(b_1)\pi(b_2)\ldots\pi(b_n)$, *where any calculations are done in* $SL(2, \mathbb{F}_2[X]/P_r(X))$.

We want to explicitly remind the reader that matrix multiplication is associative, but non-abelian and that the operations are performed in a finite group, i.e., $SL(2, \mathbb{F}_2[X]/P_r(X))$. This implies compression. Hence, the digest is always a group element. This has already been used in [16] for multimedia data. As for the *Benaloh* hash, a trivial attack vector exists for $\mathcal{TZ}$, if the input is not hashed using a standard cryptographic hash like SHA-512 prior to inputting it into $\mathcal{TZ}$. In particular, let $d \leftarrow \mathcal{TZ}(m[0]; m[1])$, where $m[0]$ is arbitrary and $m[1] = 01$. This will have the same digest as $\mathcal{TZ}(m[0]||0; 1)$ due to the mapping onto bitstrings paired with the associative behaviour. Hashing the elements using a cryptographic hash for each block $m[i]$ obviously solves this problem. In particular, this enforces an adversary to find 2^{nd}-pre-images resp. collisions. Hence, if $\mathcal{TZ}$ is used as an instantiation of a $\mathcal{AHF}_k$, the blocks need to be digested using a standard cryptographic hash. In our case, we model the hash-function as a random oracle to enforce an uniform distribution: We require that the output of the hash-functions $\mathcal{AHF}_k(s; x)$ and $\mathcal{QCHF}_k(s, x)$ is computationally indistinguishable from uniform, if s is. This will be crucial to maintain transparency. This is required, since we add fake-digests to hide redactions. The instantiations given are based on group-theoretic operations and therefore already have this property due to their algebraic structure [4; 15; 21].

3 $\boldsymbol{RSS_S}$: Construction 1 for Sets

Construction 1 (*RSS* **for Sets.**) *Our first construction makes use of an unforgeable signature scheme* ($\mathsf{SS} := \{\mathsf{SKeyGen}, \mathsf{SSign}, \mathsf{SVerify}\}$). *It uses the quasi-*

[3] Here, r is the security parameter for the hash-function.

commutative behavior of $\mathcal{QCHF}_K$ to hide redacted elements. The basic idea is depicted in Fig. 9.

Key Generation. *The key pair generation algorithm* KeyGen *outputs* $(\mathrm{sk}, \mathrm{pk}) \leftarrow \mathsf{SKeyGen}(1^\lambda)$*. Additionally, it also choses a cryptographic and quasi-commutative hash-function* $\mathcal{QCHF}_k \in \mathcal{QCHF}_K$*.*

Signing. *To sign a set* $S = \{v_1, \ldots, v_n\}$*, perform the following steps:*

1. *Set* $r \overset{\$}{\leftarrow} \mathcal{X}_k$*, i.e.,* $r \overset{\$}{\leftarrow} (\mathbb{Z}/n\mathbb{Z})^\times$ *for the hash-function by* Benaloh *et al.*
2. *Choose a random nonce* $id \overset{\$}{\leftarrow} \{0,1\}^\lambda$
3. *Accumulate each tagged* $v_i \in S$ *and* r *using* $\mathcal{QCHF}_k$*. In particular, compute* $d \leftarrow \mathcal{QCHF}_k(r, (v_1||id); \ldots; (v_n||id))$*, where* $||$ *denotes a concatenation, which is uniquely reversable.*
4. *Sign the final digest* d*,* k*, and* id*, i.e.,* $\sigma_S \leftarrow \mathsf{SSign}(\mathrm{sk}, (d||id||k))$*.*
5. *Output* (S, σ)*, where* $\sigma = (\sigma_S, r, \mathrm{pk})$*.*

Redact. *To redact a subset* $\diamond$*, the sanitizer performs the following steps:*

1. *Check* σ*'s validity using* Verify*. If the signature is not valid, return* $\perp$
2. *If* $\diamond \nsubseteq S$*, return* $\perp$
3. *Accumulate the elements* $\diamond_i \in \diamond$ *onto* r*, i.e.,* $r' \leftarrow \mathcal{QCHF}_k(r, (\diamond_1||id); \ldots; (\diamond_n||id))$
4. *Output* (S', σ')*, where* $\sigma' = (\sigma_S, r', \mathrm{pk})$ *and* $S' = S \setminus \diamond$

Verify. *The algorithm* Verify *works equivalent to the* Sign *algorithm; it performs the following steps:*

1. *Calculate* $d' \leftarrow \mathcal{QCHF}_k(r, (v_1||id); \ldots; (v_n||id))$
2. *Output* 1*, iff* d' *is the value protected by* σ_S*, w.r.t.* pk*,* 0 *otherwise, resp.* $\perp$ *on error.*

Time and Storage Complexity. RSS_S requires one fake-digest r, which size depends on the security parameter used. Hence, the storage requirement is in $\mathcal{O}(1)$. The signature generation and verification require $\mathcal{O}(n)$ steps. Hence, we have a computational complexity of $\mathcal{O}(n)$. Redaction is in $\mathcal{O}(n)$ as well.

Theorem 1 (Construction 1 is Secure). *If* $\mathcal{QCHF}_K$ *is* 2^{nd}*-pre-image resistant and pre-image resistant, while SS is unforgeable, our construction 1 is unforgeable.*

Proof. Relegated to the appendices due to size requirements.

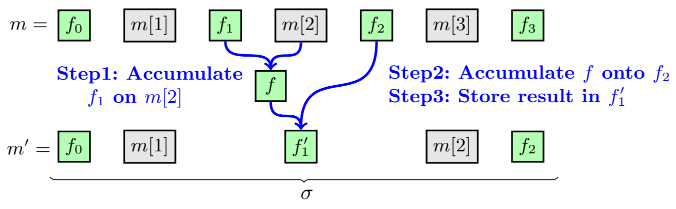
$m =$
f_0
$m[1]$
f_1
$m[2]$
f_2
$m[3]$
f_3
Step1: Accumulate f_1 on $m[2]$
f
Step2: Accumulate f onto f_2
Step3: Store result in f_1'
$m' =$
f_0
$m[1]$
f_1'
$m[2]$
f_2
σ

Fig. 10. Illustration of Construction 2

4 RSS_D: Construction 2 for linear Documents

Construction 2 (*RSS* **for Linear Documents.)** *Our second construction utilizes an unforgeable signature scheme (SS);* $(\mathsf{SS} := \{\mathsf{SKeyGen}, \mathsf{SSign}, \mathsf{SVerify}\})$*. It uses the associative but non-abelian behavior of* $\mathcal{AHF}_K$ *to hide redacted elements. The basic idea is depicted in Fig. 10.*

Key Generation. *The key pair generation algorithm is the same as in Construction 1;* KeyGen *outputs* $(\mathrm{sk}, \mathrm{pk}) \leftarrow \mathsf{SKeyGen}(1^\lambda)$*. Here,* λ *is the security parameter. Additionally, choose an associative but non-abelian hash-function* $\mathcal{AHF}_k \in \mathcal{AHF}_K$*.*

Signing. *To sign* m*, perform the following steps:*

1. *Choose* $n+1$ *random values:* $f_i \overset{\$}{\leftarrow} \mathcal{X}_k$*, where* $0 \leq i \leq n$*. For the* $\mathcal{TZ}$ *instantiation:* $f_i \overset{\$}{\leftarrow} SL(2, \mathbb{F}_2[X]/P_r(X))$*.*
2. *Choose a random nonce* $id \overset{\$}{\leftarrow} \{0,1\}^\lambda$
3. *Accumulate each tagged message and each fake-digest in the right order, i.e.,* $d \leftarrow \mathcal{AHF}(f_0; (m[1]||id); f_1; \dots; (m[n]||id); f_n)$*.*
4. *Sign* d*,* l*,* k *and* id*, i.e.,* $\sigma_m \leftarrow \mathsf{SSign}(\mathrm{sk}, (d||id||k))$*.*
5. *Output* (m, σ)*, where* $\sigma = (\sigma_m, f_0, \dots, f_n, \mathrm{pk})$*, resp.* $\perp$ *on error.*

Redact. *1. Check the validity of* σ *using* Verify*. If* σ *is not valid, return* $\perp$*. For each* $i \in \mathcal{I}$*, perform the following steps in descending order:*

2. *Accumulate* $(m[i]||id)$ *onto the associated fake-digest* f_{i-1}*: Calculate* $f \leftarrow \mathcal{AHF}_k(f_{i-1}; (m[i]||id))$*.*
3. *Merge* f *and* f_i *into* f_i'*; In particular, calculate* $f_{i-1}' \leftarrow \mathcal{AHF}_k(f; f_i)$*.*
4. *Output* (m', σ')*, where* $\sigma' = (\sigma_m, f_0, \dots, f_{i-1}', \dots, f_{n-1}, \mathrm{pk})$ *and* $m' \leftarrow \textsc{mod}(m, \mathcal{I})$*. The indices have been adjusted to account for the redaction*

Verify. *The algorithm* Verify *is similar to the* Sign *algorithm; it performs the following steps:*

1. *Accumulate each given* $m[i]$ *and* f_i *in their provided order: Calculate*

$$d \leftarrow \mathcal{AHF}_k(f_0; (m[1]||id); f_1; \ldots; (m[n]||id); f_n)$$

2. *Output* 1*, if* d *is the value protected by* σ_m*, w.r.t.* pk*,* 0 *otherwise, resp.* $\perp$ *on error.*

The verification algorithm is equal for non-redacted and redacted documents.

Time and Storage Complexity. RSS_D requires $n+1$ fake-digests. The actual size depends on $\mathcal{AHF}_k$. Hence, the storage requirement is $\mathcal{O}(n)$. The signature generation and verification requires $\mathcal{O}(n)$ steps. Hence, RSS_D also has only a runtime complexity of $\mathcal{O}(n)$. Redaction is in $\mathcal{O}(n)$ as well.

Theorem 2 (Construction 2 is Secure). *If* $\mathcal{AHF}_K$ *is* 2^{nd}*-pre-image resistant and pre-image resistant, while* SS *is unforgeable, our construction 2 is unforgeable.*

Proof. Relegated to App. B due to size requirements.

5 Conclusion, Future Work and Open Questions

We presented two secure, transparent RSS, both with a computational complexity of only $\mathcal{O}(n)$. Our first scheme protects unordered sets and has a storage complexity of $\mathcal{O}(1)$, while the second one protects linearly ordered documents, but has a storage complexity of $\mathcal{O}(n)$. Both schemes use accumulators to achieve transparency: $\mathcal{QCHF}_K$ is abelian and $\mathcal{AHF}_K$ is associative but non-commutative. We have relaxed the requirements to the existence of a family of hash-functions, which implies that the existence of a secure transparent RSS can be reduced to the existence of secure signature scheme and the existence of a family of hash-functions with quasi-commutative resp. associative but non-abelian operations. We have formally proven the security of both schemes. For describing the requirements we additionally introduced a rigid security model for RSS for sets and linear documents, which has been derived from *Brzuska* et al.'s initial model [5]. An open problem is the formulation of a formal proof for the existence of $\mathcal{AHF}_K$; as shown, constructions exist, but a formal proof, as given for abelian associative one-way-functions in [17], would be very convenient. We strongly encourage finding such a proof. We also would like to see unlinkable schemes in $\mathcal{O}(n)$, extending the work done by *Ahn* et al. [1] and *Brzuska* et al. [7].

References

1. Jae Hyun Ahn, Dan Boneh, Jan Camenisch, Susan Hohenberger, Abhi Shelat, and Brent Waters. Computing on authenticated data. In Ronald Cramer, editor, *TCC*, volume 7194 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2012.
2. Niko Barić and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *EUROCRYPT*, pages 480–494, 1997.
3. Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption. In *EUROCRYPT*, pages 92–111, 1994.
4. Josh Benaloh and Michael De Mare. One-way accumulators: A decentralized alternative to digital signatures. pages 274–285. Springer-Verlag, 1993.
5. C. Brzuska, H. Busch, O. Dagdelen, M. Fischlin, M. Franz, S. Katzenbeisser, M. Manulis, C. Onete, A. Peter, B. Poettering, and D. Schröder. Redactable Signatures for Tree-Structured Data: Definitions and Constructions. In *Proceedings of the 8th International Conference on Applied Cryptography and Network Security*, ACNS'10, pages 87–104. Springer, 2010.
6. C. Brzuska, M. Fischlin, T. Freudenreich, A. Lehmann, M. Page, J. Schelbert, D. Schröder, and F. Volk. Security of Sanitizable Signatures Revisited. In *Proc. of PKC 2009*, pages 317–336. Springer, 2009.
7. Christina Brzuska, Marc Fischlin, Anja Lehmann, and Dominique Schröder. Unlinkability of Sanitizable Signatures. In *Public Key Cryptography*, pages 444–461, 2010.
8. Sebastien Canard and Amandine Jambert. On extended sanitizable signature schemes. In *CT-RSA*, pages 179–194, 2010.
9. Ee-Chien Chang, Chee Liang Lim, and Jia Xu. Short Redactable Signatures Using Random Trees. In *Proceedings of the The Cryptographers' Track at the RSA Conference 2009 on Topics in Cryptology*, CT-RSA '09, pages 133–147, Berlin, Heidelberg, 2009. Springer-Verlag.
10. Markus Grassl, Ivana Ilic, Spyros S. Magliveras, and Rainer Steinwandt. Cryptanalysis of the tillich-zémor hash function. *J. Cryptology*, 24(1):148–156, 2011.
11. R. Johnson, D. Molnar, D. Song, and D.Wagner. Homomorphic signature schemes. In *Proceedings of the RSA Security Conference - Cryptographers Track*, pages 244–262. Springer, Feb. 2002.
12. Marek Klonowski and Anna Lauks. Extended Sanitizable Signatures. In *ICISC*, pages 343–355, 2006.
13. Kunihiko Miyazaki, Goichiro Hanaoka, and Hideki Imai. Digitally signed document sanitizing scheme based on bilinear maps. In *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, ASIACCS '06, pages 343–354, New York, NY, USA, 2006. ACM.
14. Christophe Petit and Jean-Jacques Quisquater. Preimages for the tillich-zemor hash function. In *Proceedings of the 17th international conference on Selected areas in cryptography*, SAC'10, pages 282–301, Berlin, Heidelberg, 2011. Springer-Verlag.
15. Christophe Petit and Jean-Jacques Quisquater. Rubik's for Cryptographers. `http://perso.uclouvain.be/christophe.petit/files/Rubik.pdf`, January 2011.
16. Jean-Jacques Quisquater and Marc Joye. Authentication of sequences with the sl2 hash function: application to video sequences. *J. Comput. Secur.*, 5:213–223, June 1997.
17. Muhammad Rabi and Alan T. Sherman. An observation on associative one-way functions in complexity theory. *Inf. Process. Lett.*, 64(5):239–244, 1997.

18. R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 26(1):96–99, 1983.
19. Kai Samelin, Henrich C. Pöhls, Arne Bilzhause, Joachim Posegga, and Hermann de Meer. Redactable signatures for independent removal of structure and content. In *ISPEC*, volume 7232 of *LNCS*, pages 17–33. Springer-Verlag, 2012.
20. Ron Steinfeld and Laurence Bull. Content extraction signatures. In *Information Security and Cryptology - ICISC 2001: 4th International Conference*. Springer Berlin / Heidelberg, 2002.
21. Jean-Pierre Tillich and Gilles Zémor. Hashing with sl2. In *CRYPTO*, pages 40–49, 1994.

A Relations between Security Properties

We will prove Propositions 1-3 which show the relationships between different security properties. For brevity, we will use the notation S/m whenever it is appropiate to do so.

A.1 Transparency $\implies$ Privacy

This will prove Proposition 1. Note, the proof is essentially the same as given in [5] resp. in [6]. We just restate the proof for completeness and readability and modify it to account for our altered notation. The proof is given for sets and linearly ordered documents.

Proof. Assume an (efficient) adversary $\mathcal{A}_{Priv}$ that wins our privacy games with probability $\frac{1}{2} + \epsilon$. We can then construct an (efficient) adversary $\mathcal{A}_{trans}$ which wins the transparency games. According to the transparency game, $\mathcal{A}_{trans}$ receives a public key pk and oracle access to $\mathcal{O}^{Sign}$ and $\mathcal{O}^{Sign/Redact}$. Let $\mathcal{A}_{trans}$ randomly pick a bit b' and forward pk to $\mathcal{A}_{priv}$. Whenever $\mathcal{A}_{priv}$ requests access to the signing oracle $\mathcal{O}^{Sign}$, $\mathcal{A}_{trans}$ just forwards the query to its oracle and returns the unmodified answer to $\mathcal{A}_{priv}$. When $\mathcal{A}_{priv}$ requests access to $\mathcal{O}^{LoRRedact}$ for sets, i.e., sends a query $((S_0, \diamond_0), (S_1, \diamond_1))$ then $\mathcal{A}_{trans}$ checks that $S_0 \setminus \diamond_0 = S_1 \setminus \diamond_1$ and forwards $(S_{b'}, \diamond_{b'})$ to $\mathcal{O}^{Sign/Redact}$. Similarly, if $\mathcal{O}^{LoRRedact}$ is queried for documents $((m_0, m[i]_0), (m_1, m[i]_1))$, then $\mathcal{A}_{trans}$ checks that $m_0 \setminus m_0[i_0] = m_1 \setminus m_1[i_1]$ and forwards $(m_{b'}, m[i_{b'}])$ to $\mathcal{O}^{Sign/Redact}$. Eventually, $\mathcal{A}_{priv}$ outputs its guess d. Our adversary $\mathcal{A}_{trans}$ outputs 0, if $d = b'$ and 1 otherwise.

What is the probability that $\mathcal{A}_{trans}$ is correct? We have to consider two cases:

1. If $b = 0$, then $\mathcal{O}^{Sign/Redact}$ signs the set (or the message) and redacts the chosen element or block afterwards. This gives exactly the same answer as $\mathcal{O}^{LoRRedact}$ would do, if using the bit b'. Hence, $\mathcal{A}_{priv}$ can correctly guess the bit b' with probability at least $\frac{1}{2} + \epsilon$, if $b = 0$.

2. If $b = 1$, then $\mathcal{O}^{Sign/Redact}$ always signs the document $S'_{b'}$ resp. $m'_{b'}$ *after* removing the chosen elements. Since the privacy game requires that the modified documents are equal, i.e., that $S'_0 = S'_1$ and $m'_0 = m'_1$, the answer is independent of b'. Hence $\Pr[\mathcal{A}_{Trans} = 1 \mid b = 1] = \frac{1}{2}$.

Hence, due to the probability of $\frac{1}{2}$ that $b = 1$, it follows that $\Pr[\mathcal{A}_{Trans} = b] \geq \frac{1}{2} + \frac{\epsilon}{2}$. Hence, $\mathcal{A}_{Trans}$ has non-negligible advantage, iff ϵ is non-negligible. □

A.2 Privacy $\not\Rightarrow$ Transparency

This will prove Proposition 2. Note, the proof is essentially the same as given in [5] resp. in [6]. We just restate the proof for completeness and readability and modify it to account for our altered notation. The proof is given for unordered sets and linearly ordered documents.

Proof. We have to create a redactable signature scheme which provides privacy without transparency. To do so, we append a bit b to the signature σ with the interpretation that $b = 0$ if the documents has been redacted. More precisely, let $RSS := \{\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Redact}\}$ denote a RSS which fulfills all security properties. Alter RSS into RSS' as follows:

– $\mathsf{KeyGen'}(1^\lambda) :=$ return $\mathsf{KeyGen}(1^\lambda)$

– $\mathsf{Sign'}(sk, S/m) :=$ return $(S/m, \sigma||1)$, where $(S/m, \sigma) \leftarrow \mathsf{Sign}(sk, S/m)$

– $\mathsf{Verify'}(pk, S/m, \sigma||b) :=$ return $\mathsf{Verify}(pk, S/m, \sigma)$

– $\mathsf{Redact'}(pk, \sigma||b, S/m, \diamond/i) :=$ return $(S'/m', \sigma'||0)$
  where $(S'/m', \sigma') \leftarrow \mathsf{Redact}(pk, \sigma, S/m, \diamond/i)$

A third party can easily decide whether a signature originates from the signer or not. Hence, Transparency is clearly violated. On the other hand, the added bit b gives no information about the document, so it does not help in the privacy experiment. □

A.3 Independence of Unforgeability

This will prove Proposition 3. To the best of our knowledge, this is the first proof that unforgeability for RSS is independent of privacy and transparency.

Proof. First, we show that Unforgeability implies neither transparency nor privacy. To do so, we append a collision-resistant one-way hash $\mathcal{H}$ of the original document $d \leftarrow \mathcal{H}(m)$ or the ordered set $d \leftarrow \mathcal{H}(\text{sort}(S))$ to the signed document. So we sign $(\text{sort}(S)/m||d)$. Let $RSS := \{\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Redact}\}$ denote a redactable signature scheme which fulfills all the mentioned security properties. We then alter RSS into RSS' as follows:

– $\mathsf{KeyGen'}(1^\lambda) := \mathsf{KeyGen}(1^\lambda)$
– $\mathsf{Sign'}(sk, \mathrm{sort}(S)/m) := ((S/m, d), \sigma)$ by first calculating $d = \mathcal{H}(m)$ resp. $d = \mathcal{H}(\mathrm{sort}(S))$ and then $\mathsf{Sign}(sk, \mathrm{sort}(S)/m||d)$.
– $\mathsf{Verify'}(pk, (S/m, d), \sigma) := \mathsf{Verify}(pk, \mathrm{sort}(S)/m, \sigma)$
– $\mathsf{Redact'}(pk, (S/m, d), \diamond/i) := ((S'/m', d), \sigma') \leftarrow \mathsf{Redact}(pk, S/m, \diamond/i)$

To win the privacy game two cases must be considered:

1. The digest $\mathcal{H}(\mathrm{sort}(S)/m)$ is signed but $\mathcal{H}(\mathrm{sort}(S)/m) \neq \mathcal{H}(\mathrm{sort}(S')/m')$
2. The digest $\mathcal{H}(\mathrm{sort}(S)/m)$ is signed and $\mathcal{H}(\mathrm{sort}(S)/m) = \mathcal{H}(\mathrm{sort}(S')/m')$

Here, the digest is appended to the signature, not to the message, to "trick" the privacy game. Obviously, all cases are an invasion of privacy. Due to Prop. 1 we also know that Transparency is lost. However, the scheme is still unforgeable, as long as $\mathcal{H}$ is collision-resistant.

To show that neither privacy nor transparency imply unforgeability, we only need to assume a forgeable underlying signature scheme. Hence, transparency and privacy are given, while an adversary can win the unforgeability game. $\square$

B Security Proofs of Construction 1 and 2

Both constructions can be proven secure in the same way. Hence, we will join the proofs. If any differences are present, we will highlight them at the specific location. Note, it is sufficient to show that transparency and unforgeability hold to show that our schemes are secure due to transparency $\implies$ privacy. We will show each property on its own. As before, we will use the notation S/m whenever we do not need to distinguish between a set S or a message m. d will denote a digest.

Theorem 3 (Construction 1 and 2 are unforgeable). *If $\mathcal{QCHF}_K$ resp. $\mathcal{AHF}_K$ are 2^{nd}-pre-image resistant and pre-image resistant, while SS is unforgeable, our scheme is unforgeable.*

Proof. Let $\mathcal{A}_{unf}$ be an algorithm winning our unforgeability game for RSS. We can then use $\mathcal{A}_{unf}$ to forge the underlying signature scheme SS, or to break the 2^{nd}-pre-image resistance or to find pre-images of $\mathcal{QCHF}_K$ resp. $\mathcal{AHF}_K$. We will show next, that our schemes' security relies upon the security of SS, $\mathcal{QCHF}_K$ resp. $\mathcal{AHF}_K$. Given the games in Fig. 1 and in Fig. 2, we can derive that a forgery must fall in at least one of the three cases:

Case 1: The value protected by the underlying signature σ_m has never been signed

Case 2: The value protected by the underlying signature σ_m has been signed, but $S^* \not\subseteq S$ resp. $m^* \notin \mathrm{span}_{\vdash}(m)$, in particular the documents protected are not in the transitive closure of any queried signature. This case has to be divided as well:

Case 2a: $S \not\subseteq S^*$ resp. $m \notin \text{span}_{\vdash}(m^*)$

Case 2b: $S^* \supset S$ resp. $m \in \text{span}_{\vdash}(m^*)$

We omit the case for collisions that are in the transitive closure. Collisions in the transitive closure are a result of valid redactions. Hence, these collisions are intentionally allowed and make our schemes correct. Therefore, it is not considered a valid collision in terms of the forgery experiments.

Case 1: We use $\mathcal{A}_{unf}$ to forge a signature of the underlying SS. To forge a SS signature, the attacker can construct an algorithm $\mathcal{A}_{unfSS}$ that generates a valid signature for a new message m^* not queried. If $\mathcal{A}_{unf}$ is an algorithm with non-negligible advantage ϵ winning the unforgeability game, it can be used to construct $\mathcal{A}_{unfSS}$. To do so, we use $\mathcal{A}_{unf}$ as a black-box.

1. First, $\mathcal{A}_{unfSS}$ chooses a hash-function $\mathcal{QCHF}_k \in \mathcal{QCHF}_K$ resp. $\mathcal{AHF}_k \in \mathcal{AHF}_K$ and passes k to instruct $\mathcal{A}_{unf}$ to use the same hash-function.
2. This is also done with pk of the SS to forge.
3. Any queries to the RSS signing oracle from $\mathcal{A}_{unf}$ are forwarded to $\mathcal{A}_{unfSS}$'s own signing oracle and genuinely returned to $\mathcal{A}_{unf}$.
4. Eventually, $\mathcal{A}_{unf}$ will output a pair $(S^*, (\sigma_S^*, r^*, pk))$ resp. $(m^*, (\sigma_m^*, f_1^*, \ldots, f_n^*, pk))$
5. $\mathcal{A}_{unfSS}$ returns $(\mathcal{QCHF}_k(r^*, (v_1^*||id^*); \ldots; (v_n^*||id^*)), \sigma_S^*)$ for sets or $(\mathcal{AHF}_k(f_0^*; (m[1]^*||id^*); f_1^*; \ldots; (m[n]^*||id^*); f_n^*), \sigma_m^*)$ for lists resp. If the digest has been queried, abort.

The tuples are valid forgeries of the underlying signature scheme, since $\mathcal{AHF}_k(f_0^*; (m[1]^*||id^*); f_1^*; \ldots; (m[n^*]^*||id^*); f_{n^*}^*)$ resp. $\mathcal{QCHF}_k(r^*, (v_1^*||id^*); \ldots; (v_n^*||id^*))$ have never been queried, i.e., no second pre-image has been found.

Therefore, the SS itself must have been forged.

Case 2a: We can use $\mathcal{A}_{unf}$ in a new algorithm $\mathcal{A}_{2nd}$ to break the 2^{nd}-pre-image resistance of the underlying hash-function.

1. First, $\mathcal{A}_{2nd}$ generates a key pair of a SS to emulate the signing oracle and receives $\mathcal{QCHF}_k \in \mathcal{QCHF}_K$ or $\mathcal{AHF}_k \in \mathcal{AHF}_K$ resp.
2. It passes pk and k to $\mathcal{A}_{unf}$.
3. For every request to the signing oracle, $\mathcal{A}_{2nd}$ generates the signature σ using sk and returns it to $\mathcal{A}_{unf}$.
4. Eventually, $\mathcal{A}_{unf}$ will output $(S^*, (\sigma_S^*, r^*, pk))$ or $(m^*, (\sigma_m^*, f_0^*, \ldots, f_{n^*}^*, pk))$ resp.

5. Given the transcript of the simulation, $\mathcal{A}_{2nd}$ searches for a pair

$$\mathcal{AHF}_k(f_{0,i};(m[1]_i||id_i);f_{1,i}\ldots;(m[n]_i||id_i);f_{n,i}) = \mathcal{AHF}_k(f_0^*;(m[1]^*||id^*);f_1^*;\ldots;(m[n^*]^*||id^*);f_{n^*}^*)$$

   or $\mathcal{QCHF}_k(r_i,(v_{1_i}||id_i);\ldots;(v_{n,i}||id_i)) = \mathcal{QCHF}_k(r^*,(v_1^*||id^*);\ldots;(v_{n^*}^*||id^*))$ resp.

6. If such a pair is found and $S^* \nsubseteq S_i$ resp. $m^* \notin \text{span}_\vdash(m_i)$ and $S_i \nsubseteq S^*$ resp. $m_i \notin \text{span}_\vdash(m^*)$ yields, $\mathcal{A}_{2nd}$ outputs exactly this pair, else it aborts.

Hence, the adversary was able to find a 2^{nd}-pre-image of $\mathcal{AHF}_k$ resp. $\mathcal{QCHF}_k$. In particular, it is able to provide a list which maps to the same digest value, which was not in the transitive closure.

Case 2b: In this case the attacker was able to add members to the set and created a larger set S^* or add blocks while the signature is still valid as the digest is equal. We will now show how to use $\mathcal{A}_{unf}$ to construct $\mathcal{A}_{one}$ which breaks the pre-image resistance of the underlying hash-function.

1. To do so, $\mathcal{A}_{one}$ generates a key pair of a SS to emulate the signing oracle and receives $\mathcal{QCHF}_k \in \mathcal{QCHF}_K$ or $\mathcal{AHF}_k \in \mathcal{AHF}_K$ resp.
2. It passes pk and k to $\mathcal{A}_{unf}$.
3. For every request to the signing oracle, $\mathcal{A}_{one}$ generates the signature σ using sk and returns it to $\mathcal{A}_{unf}$.
4. Eventually, $\mathcal{A}_{unf}$ will output $(S^*,(\sigma_S^*,r^*,pk))$ resp. $(m^*,(\sigma_m^*,f_0^*,\ldots,f_{n^*}^*,pk))$.
5. Given the transcript of the simulation, $\mathcal{A}_{one}$ searches for a pair $\mathcal{AHF}_k(f_{0,i};(m[1]_i||id_i);f_{1,i};\ldots;(m[n]_i||id_i);f_{n,i}) = \mathcal{AHF}_k(f_0^*;(m[1]^*||id);f_1^*;\ldots;(m[n^*]^*||id);f_{n^*}^*)$
   resp. $\mathcal{QCHF}_k(r,(v_{1,i}||id);\ldots;(v_{n,i}||id)) = \mathcal{QCHF}_k(r^*,(v_1^*||id);\ldots;(v_{n^*}^*||id))$.
6. If such a pair is found and $S_i \subsetneq S^*$ resp. $m_i \in \text{span}_\vdash(m^*)$, where $m \neq m^*$ yields, $\mathcal{A}_{one}$ outputs exactly this pair, else it aborts.

This algorithm allows to expand digests, which contradicts the assumption made about the pre-image resistance of the accumulator. Note, the message needs to be in the transitive closure to be considered a valid expansion. Otherwise, we have a second pre-image, which belongs to case 2a. The actual pre-image of the hash-function can easily be extracted. Again, we can use $\mathcal{A}_{one}$ to win our Resistance game depicted in Fig. 7 resp. Fig. 8. Note, all messages are randomized due to the appended and randomized document identifier. Even if the queried message is in the transitive closure, the adversary is not able to give an already redacted message to the signer and then create a forgery by "expanding" them afterwards, since the adversary cannot guess the document identifier. This prevents framing attacks.

In all cases: $\mathcal{A}_{unf}$ is successful, iff it can break at least one of the underlying primitives. □

Theorem 4 (Construction 1 and 2 are transparent and private). *If $\mathcal{QCHF}_K$ and $\mathcal{AHF}_K$ always output uniformly distributed digests and are therefore indistinguishable, our scheme is transparent and due to Prop. 1, also private.*

Proof. This follows directly from the definitions, i.e., the uniform distribution of the digests. In particular, all digests are computationally indistinguishable from random. This implies that the output of Sign resp. Redact is also computationally indistinguishable from uniform, therefore hiding the secret bit b with overwhelming probability. In other words, an adversary breaking transparency is able to distinguish between random and computed digests, which has been assumed to be infeasible. An additional note: This is the reason why we require the fake-digests to be chosen at random. Otherwise, an adversary could just recalculate the digest or distinguish between fake and merged digests. $\square$