The State of the Union: Union-Only Signatures for Data Aggregation

Diego F. Aranha[1], Felix Engelmann[2], Sebastian Kolby[1], and Sophia Yakoubov[1]

[1] Department of Computer Science, Aarhus University, Denmark
`{dfaranha,sk,sophia.yakoubov}@cs.au.dk`
[2] IT University of Copenhagen, Denmark `fe-research@nlogn.org`

Abstract. A *union-only signature (UOS) scheme* (informally introduced by Johnson *et al.* at CT-RSA 2002) allows signers to sign sets of messages in such a way that (1) any third party can merge two signatures to derive a signature on the union of the message sets, and (2) no adversary, given a signature on some set, can derive a valid signature on any strict subset of that set (unless it has seen such a signature already).
Johnson *et al.* originally posed building a UOS as an open problem. In this paper, we make two contributions: we give the first formal definition of a UOS scheme, and we give the first UOS constructions. Our main construction uses hashing, regular digital signatures, Pedersen commitments and signatures of knowledge. We provide an implementation that demonstrates its practicality. Our main construction also relies on the hardness of the short integer solution (SIS) problem; we show how that this assumption can be replaced with the use of groups of unknown order. Finally, we sketch a UOS construction using SNARKs; this additionally gives the property that the size of the signature does not grow with the number of merges.

Keywords: homomorphic signatures, union-only signature schemes, history-hiding, software implementation

1 Introduction

A set-homomorphic digital signature is a signature scheme which supports the computation of set operations — for example union and difference — over signed messages. Let $\mathsf{Sign}(\mathsf{sk}, M)$ be the signing operation for such a signature scheme, for some private signing key sk and a set of messages M.

For sets of messages $X = \{x_1, \dots, x_k\}$ and $Y = \{y_1, \dots, y_n\}$ which were signed as $\mathsf{Sign}(\mathsf{sk}, X)$ and $\mathsf{Sign}(\mathsf{sk}, Y)$, any third party can compute the signature on their union as $\mathsf{Sign}(\mathsf{sk}, X) \times \mathsf{Sign}(\mathsf{sk}, Y)$ (we use $\times$ to denote the homomorphic union operation). If the homomorphic operation $\times$ is invertible, and $X \subseteq Y$, one can also compute the signature on their difference $\mathsf{Sign}(\mathsf{sk}, X \setminus Y) = \mathsf{Sign}(\mathsf{sk}, X) \times \mathsf{Sign}(\mathsf{sk}, Y)^{-1}$. The notion was initially introduced by Johnson *et al.* [18], together with a practical construction based on RSA accumulators, but there are instantiations based on hardness assumptions other than integer factorization [1,19].

A union-only signature (UOS) scheme is a special case of a set-homomorphic signature scheme where the homomorphic operation is *not* invertible (that is, it is one-way). In a UOS scheme, computing a signature on a set of messages is *easy* given signatures on subsets of those messages, but computing the signature on the difference of two sets (the inverse operation) is *hard*. Constructing union-only signatures was first posed as an open problem in the seminal work of Johnson *et al.* [18]. Because previous set-homomorphic constructions represent signatures as multiplicative structures (rings and multiplicative groups), computing the difference operation amounts to inverting elements in such structures, which can be done efficiently. For this reason, building a UOS based on those constructions is challenging, and implies the existence of groups with infeasible inversion (GIIs) [23], a powerful algebraic structure that further implies strong associative one-way functions [28], efficient two-party secret key agreement protocols, and direct transitive signatures [17]. In a GII, computing the inverse of a group element is required to be *hard*, while performing the group operation is computationally efficient. While GIIs are not known to exist, there are recent candidate constructions based on self-bilinear maps assuming indistinguishability obfuscation [31] and isogeny graphs [5].

Contributions. In this paper, we make two contributions. First, we take the opportunity to formalize the definition of a secure UOS scheme. Second, we present two constructions which circumvent the roadblock described above by choosing the signature format to *not* have a multiplicative structure. Our first construction is based on hashing, Schnorr signatures, Pedersen commitments and signatures of knowledge instantiated with elliptic curves. It also relies on the hardness of the short integer solution (SIS) problem [3,22]. We show how to replace this assumption with the use of groups of unknown order in a variant. A second construction based on SNARKs appears in the Appendix.

All of our constructions support multiple signers and offer a notion of *privacy* which precludes an adversary from learning how the signatures were derived (i.e., which subsets were actually signed by the signer, and which order the signatures were merged in). The first construction performs much better, so we explore it in detail; the SNARK-based construction produces constant-size signatures, offering a trade-off between performance and compactness.

In our first construction, we employ *multisets*; we design a scheme that preserves duplicates in the intersection of the merged sets instead of removing them, which technically makes it homomorphic with respect to multiset *sum*. However, this naturally coincides with the union operation for disjoint input sets and satisfies the original intuition of UOS given by Johnson *et al.* [18]. From this point on, we abstract this technicality away in the scheme's interface and refer only to the set union operation for simplicity. We provide a proof-of-concept implementation in `Rust` that shows that the construction is indeed *efficient and scalable.*

Applications. To the best of our knowledge, the literature does not contain concrete applications specific to UOS schemes. We devise application scenarios considering a minimum of 4 parties:

Signer(s): Assumed to be *honest*, one or more **Signers** sign(s) sets of messages with their signing keys, and make them available to a **Merger**. Importantly, signatures over sets containing individual messages should not be publicly available; otherwise, a third party (e.g. **Prover**) can remove them from a merged signature simply by re-executing the merge on all the *other* signatures.

Merger: a **Merger** merges signatures following a public procedure and forwards the resulting signature to a **Prover**. The merger must be independent of the signer(s), and is trusted only to discard the merge history (when hiding the history is desirable for privacy reasons).

Prover: a **Prover** trusts the public key(s) belonging to the **Signer(s)**, and therefore can be convinced that signatures are valid. It attempts to convince a **Verifier** that all relevant messages are included in the merged signature provided by the **Merger**.

Verifier: a **Verifier** verifies the signature and wants to check that the prover did not exclude any messages from the set.

Figure **??** illustrates the workflow. In terms of incentives, signers want to publish their signed messages for credibility or to achieve some common goal. They do not trust each other unconditionally (e.g. they still want privacy against one another) and thus have their own key pairs. Signatures are merged by the merger, and used by the prover to convince external parties (verifiers) that uncomfortable messages or data points were not omitted on purpose.

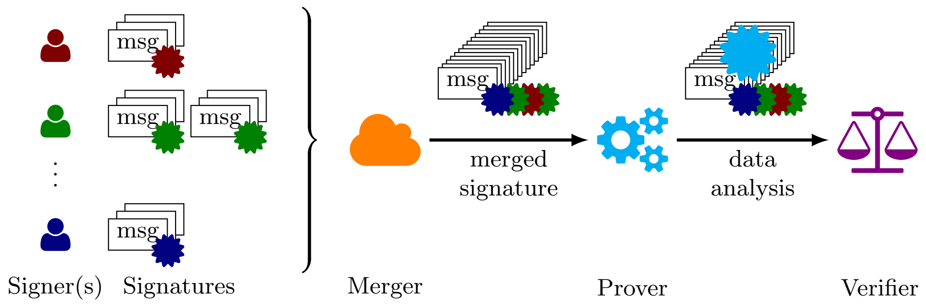

Fig. 1. Application scenario illustrating how authenticated data flows from Signer(s) to the interaction between Prover and Verifier, after being merged by the Merger.

The general framework motivates a few scenarios. In epidemiology studies, healthcare providers sign patient medical records (e.g. containing vaccination or infection information) that are of interest to researchers. An independent government entity merges the medical records from multiple providers, with an incentive to have as many as possible for credibility, even if specific data points are not desirable. In this case, the government does not have a trusted signing key to include new authenticated records (otherwise, we could imagine the government using this key to inflate the number of records). Researchers can then perform statistical analysis of the records and verifiably convince the public that all data points available were taken into consideration to reach the conclusions. Privacy guarantees are important to prevent the public from matching certain patient data to a given healthcare provider, increasing their chances of de-anonymizing the patient in question.

In a biodiversity setting, databases are jointly built[3] by multiple curators contributing data points (e.g. wildlife sightings for species in danger of extinction). The database infrastructure merges the individual datasets with an incentive to have as much data as possible available to researchers, who can then analyse records and provide guarantees to the public that all data was considered. We argue that privacy guarantees are important to collect data and perform impactful research in these domains, as to prevent the public from being able to track down and further jeopardize endangered species.

Related work. Homomorphic signatures were initially proposed by Johnson *et al.* [18]. Many possibilities for the homomorphic operation were further developed in the literature [11,12,30,7], but the original paper already gave constructions for redactable and set-homomorphic signatures. For the latter, assume that N is an RSA modulus for which only the signer knows the factorization $\mathsf{sk} = (p, q)$. For $X = \{x_1, \ldots, x_k\}$, define $\mathsf{Sign}(\mathsf{sk}, X) = v^{1/d} \bmod N$ for random $v \in \mathbb{Z}_N^*$ and a hash function $h(\cdot)$ that maps elements to a set of primes and $d = \prod_{i=0}^{k} h(x_i)$. Given a signature σ on X, one can compute a signature on a subset $X \setminus \{x_i\}$ by computing $\sigma^{h(x_i)} \bmod N$. For d, X as above and $Y = \{y_1, \ldots, y_n\}$, $e = \prod_{j=0}^{n} h(y_j)$, a signature on $X \cup Y$ can be computed as $v^{1/lcm(d,e)} = (v^{1/d})^a (v^{1/e})^b \bmod N$ with integers (a, b) such that $ad + be = gcd(d, e)$. The multiplicative structure of the signatures allows efficient union/difference by respectively adding/ removing elements from the accumulator, but complicates the design of UOS. Because of the set difference operation, this does *not* imply UOS.

Abiteboul *et al.* [1] study homomorphic signatures for modifiable collections, with applications to access control and secure data aggregation. They first recast the scheme above as *delete-only*, for which an initial signature is computed over the entire universe of elements and individual elements can be progressively removed. They also give privacy definitions in a single-signer context and propose an *insert-only* signature scheme satisfying computational privacy that supports both insertion of individual elements and computation of a set intersection. The scheme is constructed from a cryptographically-enforced “write-only” memory and the delete-only scheme as building blocks, and imposes an upper bound on the collection size, forcing a key refresh when the limit is reached. Additional constructions based on zero-knowledge proofs are given for both delete- and insert-only schemes, but without privacy claims.

Kaaniche *et al.* [19] propose pairing-based set-homomorphic signatures supporting union, difference and intersection for secure data aggregation within

[3] Global Biodiversity Information Facility: `https://www.gbif.org/`

the Internet of Things. The construction builds on efficient accumulators from bilinear maps instead of RSA accumulators. Privacy is informally defined as the adversary not being able to link signatures to signers or to detect removed content. In terms of performance, pairing-based schemes rely on rather special pairing-friendly curves, which are known to be less efficient than plain elliptic curves due to their larger parameters [8].

A generalized notion introduced by Ahn et al. in [2] is *P-homomorphic* signatures, in which anyone can derive a signature for a message m' from the signatures of a set of messages M as long as the predicate $P(M, m')$ holds. The paper puts forward not only the abstract notion, but also a few concrete constructions for fixed predicates, including string subset. Many homomorphic, transitive and redactable signature schemes can be recast under this notion, but it is not clear how to extend such constructions for the union-only predicate efficiently. Another related notion is *mergeable* signatures [26,27], which are defined for redactable signature schemes which allow a subset operation by design. A construction based on signed RSA accumulators is also given, together with applications such as merging databases while redacting specific entries.

A last related notion is that of extendable threshold ring signatures [6], where signatures can be thought of as homomorphic with respect to the set of signers (i.e., given a message signed by an anonymous subset of a set of signers, new signers can contribute their support and potentially expand the anonymity set). However, extendable threshold ring signatures do not offer any homomorphism when it comes to the message in question. They also offer a very different notion of privacy: they protect the identities of the signers, while our history-hiding property protects the history and origins of the message set.

The schemes discussed previously do not satisfy the UOS requirement of allowing union of messages as the only homomorphic operation, since they support deletion and/or set intersection, or operate over the set of signers.

Organization. The paper is organized as follows. Section 2 defines the syntax for UOS, and Section 3 defines the desired security notions. Section 4 presents our main and most efficient construction. The experimental results supporting our efficiency claims are discussed in Section 5, followed by the conclusion. An interested reader can find our SNARK-based construction in the Appendix.

2 Syntax

As described above, in a UOS scheme, signers sign *sets* of messages, and any third party can merge signatures on such sets.

Definition 1 (Union-only Signature Scheme). *A* $\mathsf{UOS}=(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Merge}, \mathsf{Verify})$ *consists of five algorithms with the following syntax:*

$\mathsf{Setup}(1^\lambda) \rightarrow \mathsf{p}$ *takes the security parameter* λ *and outputs public parameters* p *which are implicitly given to all algorithms.*

$\mathsf{KeyGen}() \rightarrow (\mathsf{pk}, \mathsf{sk})$ *generates a public-secret key pair.*

$\mathsf{Sign}(\mathsf{sk}, M) \to \sigma$ *takes a secret key* sk *and a set of messages M. Outputs a signature σ.*

$\mathsf{Merge}(\{(\sigma_i, M_i, PK_i)\}_{i=1}^{\mu}) \to \sigma$ *takes a set of μ tuples, where each tuple consists of a set of messages M_i, a signature σ_i and a set public keys PK_i. Outputs a new signature σ (which should verify for the message set $M = \cup_{i=1}^{\mu} M_i$ under the public key set $PK = \cup_{i=1}^{\mu} PK_i$).*

$\mathsf{Verify}(\sigma, M, PK) \to \{\mathtt{accept}, \mathtt{reject}\}$ *takes a signature σ, a set of messages M and a set of public keys PK, and outputs either* `accept` *or* `reject`*.*

These algorithms should satisfy the natural notion of correctness; that is, the output of any sequence of honestly executed Sign *and* Merge *operations should be a verifying signature. They should also satisfy Definition 4 and Definition 3, described in Section 3.*

3 Security Definitions

We require two properties of a UOS: *unforgeability* and *history-hiding*. Informally, unforgeability demands that an adversary not be able to sign on behalf of a set of signers none of whom are corrupt. History-hiding demands that an adversary not be able to determine how a given signature was derived.

3.1 Notation

We formalise the power of the adversary through four oracles, with access to some common state, described in Figure 4. The first three oracles provide bookkeeping of identities: $\mathsf{KeyGen}\mathcal{O}$ adds a new honest party to the system, while $\mathsf{Corrupt}\mathcal{O}$ corrupts an existing party, and $\mathsf{RegKey}\mathcal{O}$ registers a new corrupt party. The keys in the system are stored by the oracles in a list L_K; the indices corresponding to honest and corrupt parties in the key list are tracked in the sets H and C respectively.

The final oracle we define is the signing oracle, denoted $\mathsf{Merge}\mathcal{O}$. It takes a sequence of sign and merge operations described by the adversary, and outputs the resulting signature. The oracle maintains a set of past queries L_T, and a counter q_s for the number of queries made. For ease of notation, we define a *query tree* T representing the kind of query an adversary can submit to a signing oracle in our security games. (Since a signature can be derived by merging other signatures, our signing oracle takes queries that are more complex than a single message set and the signer the adversary wishes to see a signature from.)

In a query tree T, each leaf l represents a signature on a set of messages $l.M$ by signers in $l.PK$. If a leaf does not contain a signature provided by the adversary and $l.PK$ is a singleton containing the public key $\mathsf{pk}_{l.i}$ of an honest signer $l.i$, we call this an honest leaf. The leaves of T may be partitioned into the set of honest leaves $hl(T)$ and the set of corrupt leaves $cl(T)$. Corrupt leaves may have an arbitrary $l.PK$ and can contain a signature provided by the adversary. The signing oracle $\mathsf{Merge}\mathcal{O}$ only answers queries where corrupt leaves contain

```
$\mathsf{VerifyLeaves}(T)$:
  for $l \in cl(T)$ do
    if $\mathsf{Verify}(l.\sigma, l.M, l.PK) = \bot$ then
      return $\bot$
```

Fig. 2. VerifyLeaves. This algorithm checks that all signatures included in a query tree by the adversary verify. It is used in Figure 4. The set of public keys PK contains all public keys pk_i associated with the tree. Note that. if $l.\sigma$ is not provided, verification will always return $\bot$.

verifying signatures. To enforce this we introduce the VerifyLeaves procedure, which checks that all corrupt leaves verify.

Each internal node n of T represents the signature derived by merging its children. So, each node n has associated message and signer sets $n.M$ and $n.PK$ respectively. These are defined as the union of the corresponding sets across the children of n. We denote the set of signers at the root of the tree as $T.PK$.

When given a query tree T, the challenger signs the appropriate leaves on behalf of the honest signers, and merges the nodes as specified by the tree structure, until it derives the signature associated with the tree root. That signature will be the query answer. The $\mathsf{Merge}\mathcal{O}$ oracle produces signatures following the SignAndMerge procedure, described in Figure 3. It uses a depth first approach to signing and merging following the structure of the tree T. During the traversal signatures are produced for each honest leaf, while corrupt leaves already contain a signature provided by the adversary. The signatures at leaves are progressively merged towards the root, such that each node contains the signature produced by merging the signatures of its children.

3.2 Unforgeability

We define unforgeability with respect to the security game described in Figure 6. Informally, we don't want an adversary to be able to forge a signature on behalf of a set of honest signers, as long as that signature is on a set of messages that could not have been obtained through taking unions of message sets on which the signing oracle was queried. To formalize this, we define an *outside span* algorithm (Definition 2, Figure 5) that determines whether a given set of messages and honest signer identities are outside the span of the set of signing oracle queries.

Definition 2 (Outside span). *We define the predicate*

$$\mathsf{OutsideSpan}(M^*, PK^*, \{(M_i^h, PK_i^h)\}_{i\in[n]})$$

for sets of messages and public keys M_i^h, PK_i^h as

$$\nexists S \subseteq [n] : \Big(M^* = \bigcup_{i\in S} M_i^h\Big) \wedge \Big(PK^* = \bigcup_{i\in S} PK_i^h\Big).$$

```
$\mathsf{SignAndMerge}(T, SK)$:
  $Q := \{T.root\}$                                          $\triangleright$ new stack
  while $\neg Q.isEmpty$ do
      $n := Q.pop()$
      if $n.isLeaf \wedge n.\sigma = \emptyset$ then
          $n.\sigma := \mathsf{Sign}(\mathsf{sk}_{n.i}, n.M)$
      else if $n.childrenQueued$ then
          $n.\sigma := \mathsf{Merge}(\{(c.\sigma, c.M, c.PK)\}_{c \in n.children})$
      else
          $Q.push(n)$
          for $c \in n.children$ do
              $c.childrenQueued := \bot$
              $Q.push(c)$
          $n.childrenQueued := \top$
  return $n.\sigma$
```

Fig. 3. $\mathsf{SignAndMerge}$. The set of secret keys SK contains all honest parties' secret keys sk_i.

This can be efficiently checked as described in the algorithm in Figure 5.

For instance, imagine that we asked the signing oracle the set of queries $S = \{(\{m_1\}, \{\mathsf{pk}_1\}), (\{m_2\}, \{\mathsf{pk}_2\})\}$; in other words, we requested a signature on m_1 under pk_1, and a signature m_2 under pk_2 from the signing oracle. Then, a signature on $\{m_1, m_2\}$ under $\{\mathsf{pk}_1, \mathsf{pk}_2\}$ should be computable in a UOS scheme; so, it is within the span (in other words, $\mathsf{OutsideSpan}(\{m_1, m_2\}, \{\mathsf{pk}_1, \mathsf{pk}_2\}, S) = \bot$). However, anything which is not computable via union operations is outside the span. As an example, a signature on $\{m_1, m_2\}$ under pk_1 alone should not be computable (so, $\mathsf{OutsideSpan}(\{m_1, m_2\}, \{\mathsf{pk}_1\}, S) = \top$). A signature on $\{m_1, m_2, m_3\}$ under $\{\mathsf{pk}_1, \mathsf{pk}_2\}$ should also not be computable (so, $\mathsf{OutsideSpan}(\{m_1, m_2, m_3\}, \{\mathsf{pk}_1, \mathsf{pk}_2\}, S) = \top$).

Definition 3 (Unforgeability). *A UOS scheme is* unforgeable *if for all PPT adversaries $\mathcal{A}$ it holds that*

$$|\Pr[\mathcal{A} \textit{ wins } \mathrm{EUF}(\lambda)]| \leq \mathsf{negl}(\lambda)$$

with the game EUF defined in Figure 6.

3.3 History Hiding

We define history-hiding for UOS schemes to require that no adversary can tell the difference between signatures on trees which are "similar enough". We leave the definition of "similar enough" as a parameter of the history-hiding property, formalized as an equivalence relation $\equiv_T$. We describe two options for $\equiv_T$ here.

```
KeyGen$\mathcal{O}(i)$
1 : if $(i,\cdot,\cdot) \in \mathsf{L}_K$ : return $\perp$
2 : $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}()$
3 : $\mathsf{L}_K \leftarrow \mathsf{L}_K \cup \{(i, \mathsf{pk}_i, \mathsf{sk}_i)\}$
4 : $H \leftarrow H \cup \{i\}$
5 : return $\mathsf{pk}_i$

Corrupt$\mathcal{O}(i)$
1 : if $(i,\cdot,\cdot) \notin \mathsf{L}_K$ : return $\perp$
2 : retrieve $(i, \mathsf{pk}_i, \mathsf{sk}_i)$ from $\mathsf{L}_K$
3 : $H \leftarrow H \setminus \{i\}$
4 : $C \leftarrow C \cup \{i\}$
5 : return $\mathsf{sk}_i$

RegKey$\mathcal{O}(i, \mathsf{pk}_i)$
1 : if $(i,\cdot,\cdot) \in \mathsf{L}_K$ : return $\perp$
2 : $\mathsf{L}_K \leftarrow \mathsf{L}_K \cup \{(i, \mathsf{pk}_i, \perp)\}$
3 : $C \leftarrow C \cup \{i\}$

Merge$\mathcal{O}(T_j)$
1 : $q_s \leftarrow q_s + 1$
2 : $\mathsf{L}_T \leftarrow \mathsf{L}_T \cup \{(q_s, T_j)\}$
3 : if $(T_j.PK \not\subset \{\mathsf{pk}_i \mid i \in H \cup C\}) \vee$
      $(\mathsf{VerifyLeaves}(T_j) = \perp)$ :
4 :   return $\perp$
5 : $\sigma_j \leftarrow \mathsf{SignAndMerge}(T_j, \{\mathsf{sk}_i \mid i \in H\})$
6 : return $\sigma_j$
```

Fig. 4. Oracles for key generation, signing and corruption, used in the Unforgeability and History Hiding games.

```
OutsideSpan$(M^*, PK^*, \{(M_i, PK_i)\}_{i=1}^{\mu})$
  $S := \{i \in [n] \mid (M_i \setminus M^* = \emptyset) \wedge (PK_i \setminus PK^* = \emptyset)\}$
  $M := \bigcup_{i \in S} M_i$
  $PK := \bigcup_{i \in S} PK_i$
  return $\neg\big((M^* = M) \wedge (PK^* = PK)\big)$
```

Fig. 5. OutsideSpan

$\equiv_{strong}$: This equivalence relation deems two trees T_0 and T_1 equivalent if the union of their leaf message sets, and the union of their leaf identities, are the same.

$\equiv_{weak}$: This equivalence relation deems two trees T_0 and T_1 equivalent if:

1. The set of corrupt leaves on the two trees are the same.
2. The number of leaves per honest signer is equal.
3. The multiset union of the *honest* leaf message sets are equal.

The use of both these relations within the history-hiding definition demands that honest signers who contribute signatures to a merged signature cannot be linked to a specific subset of the signed messages, even in the presence of malicious signers. The use of $\equiv_{strong}$ additionally demands that even corrupt parties cannot be linked to a specific subset of messages.

```
EUF(λ)
1 :  $H \leftarrow \emptyset;\ C \leftarrow \emptyset;\ \mathsf{L}_K \leftarrow \emptyset;\ \mathsf{L}_T \leftarrow \emptyset; q_s \leftarrow 0$
2 :  $\mathsf{p} \leftarrow \mathsf{Setup}(1^\lambda)$
3 :  $\mathcal{O} \leftarrow \{\mathsf{KeyGen}\mathcal{O}, \mathsf{Corrupt}\mathcal{O}, \mathsf{RegKey}\mathcal{O}, \mathsf{Merge}\mathcal{O}\}$
4 :  $(\sigma^*, M^*, PK^*) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{p})$
5 :  for $(\cdot, T_j) \in \mathsf{L}_T$ :
6 :      $PK_j^h := \{\mathsf{pk}_{l.i}\}_{l \in hl(T_j)}$
7 :      $M_j^h := \bigcup_{l \in hl(T_j)} l.M$
8 :  if $\mathsf{Verify}(\sigma^*, M^*, PK^*) = \perp$ : $\mathcal{A}$ loses
9 :  if $\mathsf{OutsideSpan}(M^*, PK^*, \{(M_j^h, PK_j^h)\}_{j=1}^{q_s}) = \perp$ :
10 :     $\mathcal{A}$ loses
11 : if $PK^* \setminus \{\mathsf{pk}_i\}_{i \in H} \neq \emptyset$ : $\mathcal{A}$ loses
12 : else $\mathcal{A}$ wins
```

Fig. 6. The Unforgeability Game

```
HH_b(λ)
1 :  $H \leftarrow \emptyset;\ C \leftarrow \emptyset;\ \mathsf{L}_K \leftarrow \emptyset;\ \mathsf{L}_T \leftarrow \emptyset; q_s \leftarrow 0$
2 :  $\mathsf{p} \leftarrow \mathsf{Setup}(1^\lambda)$
3 :  $\mathcal{O} \leftarrow \{\mathsf{KeyGen}\mathcal{O}, \mathsf{Corrupt}\mathcal{O}, \mathsf{RegKey}\mathcal{O}, \mathsf{Merge}\mathcal{O}\}$
4 :  $(T_0, T_1, \mathsf{aux}) \leftarrow \mathcal{A}_1^{\mathcal{O}}(\mathsf{p})$
5 :  if $\mathsf{VerifyLeaves}(T_0, T_0.PK) = \perp$ : $\mathcal{A}$ loses
6 :  if $\mathsf{VerifyLeaves}(T_1, T_1.PK) = \perp$ : $\mathcal{A}$ loses
7 :  $\sigma_b \leftarrow \mathsf{SignAndMerge}(T_b)$
8 :  $b' \leftarrow \mathcal{A}_2^{\mathcal{O}}(\sigma_b, \mathsf{aux})$
9 :  if $\neg(T_0 \equiv_T T_1)$ : $\mathcal{A}$ loses
10 : if $b \neq b'$ : $\mathcal{A}$ loses
11 : else $\mathcal{A}$ wins
```

Fig. 7. The History-Hiding Game

Definition 4 (History Hiding). *A UOS scheme is* history-hiding with respect to equivalence relation $\equiv_T$ *if for all PPT adversaries* $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2\}$ *it holds that*

$$|\Pr[\mathcal{A}\ \textit{wins}\ \mathrm{HH}_0(\lambda)] - \Pr[\mathcal{A}\ \textit{wins}\ \mathrm{HH}_1(\lambda)]| \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

with the game HH_b *defined in Figure 7.*

Remark 1. Note that the equivalence of the trees ($T_0 \equiv_T T_1$) depends on which signers are corrupt; for that reason, the equivalence check is performed after the adversary (possibly using access to the corruption oracle) produces the bit b'.

4 A UOS Scheme

In this section, we present a concrete UOS scheme. The construction uses signatures, Pedersen commitments and signatures of knowledge. The proof of security additionally uses additive and multiplicative secret sharing, the discrete logarithm assumption, and either the short integer solution (SIS) assumption or groups of unknown order. We present formal preliminaries in Appendix A; here, we go directly to the construction. We first describe an intuitive construction similar the anonymously aggregatable signature of SwapCT [15] which doesn't quite give us the unforgeability guarantees we would like. We then describe the two ways to obtain those guarantees.

4.1 Initial Construction

When signing a set of messages $\{m_j\}_{j=1}^{\mu}$, the signer assigns each message m_j a random proxy $s_j \in \mathbb{G}$.[4] The signer commits to the proxy as $C_j = G^{s_j} H^{r_j}$, using a freshly chosen random witness r_j. To ensure that the signer knows the commitment opening (i.e. to ensure that she didn't simply copy someone else's commitment without knowing its contents), and to bind the commitment to the message m_j, the signer creates a signature of knowledge π_j of m_j, using s_j and r_j as the witness.

To complete the signature, the signer adds up all of the proxies as $s := \sum_{j=1}^{\mu} s_j$, and commits to s as $D := G^s$. She then signs D using a regular signature scheme Sig, resulting in σ'.

The resulting UOS signature σ contains three parts:

1. The value D and the associated signature σ' (together with the public signature verification key of the signer, for ease of notation). We denote this part of the signature as
$$L = \{(\sigma', D, \mathsf{pk})\}.$$
(We write this as a set to facilitate notation for merging later.)
2. The set of messages m_j, commitments C_j and associated signatures of knowledge π_j. We denote this part of the signature as
$$R = \{(\pi_j, C_j, m_j)\}_{j=1}^{\mu}.$$
3. The *sum* $r := \sum_{j=1}^{\mu} r_j$ of all the values r_j used in the commitments.[5]

[4] A signer might want to sign an empty message set, if she is contributing the signature solely for the purposes of expanding the others' anonymity set. If this is the case, and the message set is empty, a placeholder message $\perp$ outside of the message space is added.

[5] If the order of the group $\mathbb{G}$ is known, the sum can be computed modulo that order.

To merge two signatures $\sigma_1 = (L_1, R_1, r_1)$ and $\sigma_2 = (L_2, R_2, r_2)$, any third party can simply compute $\sigma = (L_1 \cup L_2, R_1 \cup R_2, r = r_1 + r_2)$.

Note that the public verification key pk is included in L, and the messages m_j are included in R, even though the public keys and the messages are independently given to verification algorithm. This is done to make clear that a given public key should be used to verify σ' on D, and that a given message corresponds to commitment C_j. This mapping is necessary for verification of $\sigma = (L, R, r)$, during which the verifier checks the following three things:

1. For $(\sigma', D, \mathsf{pk}) \in L$, the signature σ' on D verifies under the key pk.
2. For $(\pi, C, m) \in R$, the signature of knowledge π verifies on the message m for the corresponding statement C.
3. Let $D' = \prod_{(\cdot,D,\cdot)\in L} D$, and $C' = \prod_{(\cdot,C,\cdot)\in R} C$. Then, it holds that $H^r D' = C'$.

Informally, we get the weak history hiding property because the items in L correspond to signers and the items in R correspond to messages, but there is nothing to link a given item in L to a given item in R. Unforgeability is a bit trickier to argue. There are several kinds of forgeries we would like to prevent:

1. One where the attacker adds a new value D.
2. One where the attacker uses a subset of D's produced by the honest signers, but changes the set of corresponding C's.

We can rule out the first kind of forgery simply by relying on the unforgeability of the underlying digital signature scheme: the attacker cannot sign a new value D on behalf of any of the honest signers. Within the second kind of forgery, we must consider the case where the attacker uses a strict subset of the honestly produced C's, and the case where the attacker adds new values of C. In the first case, we can use the attacker to find the discrete logarithm relationship between G and H.

However, in the second case, there is a trivial attack — the attacker can add the message m to any signature $\sigma = (L, R, r)$ by (1) choosing r' randomly, (2) choosing $s = 0$ modulo the order of the group $\mathbb{G}$, (3) computing (π, C, m) using those values of s and r', and (4) setting the new signature to be

$$\sigma' = (L, R \cup \{(\pi, C, m)\}, r + r').$$

This new signature will verify, since we have not changed the exponent of G on either side, and have made sure that the exponents of H change consistently on the two sides (by adding r' to r).

We can preclude this kind of attack in one of two ways: by relying on groups of unknown order, or by relying on the hardness of the short integer solution problem. We describe our construction formally in Figure 8, and informally below.

```
$\mathsf{Setup}(1^\lambda)$:
    $(\mathbb{G}, q, G) \leftarrow \mathsf{GroupGen}(1^\kappa)$, where $\kappa$ is the bit-length of the group order $q$, and $G$ is a generator of $\mathbb{G}$.
    Pick a second generator $H$ of $\mathbb{G}$.
    for all $k \in [w]$ do
        $G_k \xleftarrow{\$} \mathbb{G}, \mathsf{H}_k \xleftarrow{\$} \mathcal{H}$
    $\mathtt{crs} \leftarrow \mathsf{SoK}[\mathcal{L}^{\mathsf{ped}}].\mathsf{Setup}(1^\lambda)$
    return $\mathsf{p} := (H, G, (G_1, \ldots, G_w), (\mathsf{H}_1, \ldots, \mathsf{H}_w), \mathtt{crs})$

$\mathsf{KeyGen}()$:
    $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Sig.KeyGen}(1^\lambda)$
    return $(\mathsf{pk}, \mathsf{sk})$

$\mathsf{Sign}(\mathsf{sk}, M = \{m_j\}_{j=1}^{\mu})$:
    if $\mu = 0$ then $\mu := 1$; $m_1 := \perp$
    for all $j \in [\mu]$ do
        $\triangleright$ Let $q$ be the order of $\mathbb{G}$ or $2^{2\kappa}$ chosen such that $2^\kappa$ is bigger than the order of $\mathbb{G}$
        $s_j, r_j \xleftarrow{\$} \mathbb{Z}_q$
        $C_j := H^{r_j} G^{s_j}$
        $\pi_j \leftarrow \mathsf{SoK}[\mathcal{L}^{\mathsf{ped}}$ or $\mathcal{L}^{\mathsf{ped'}}].\mathsf{Sign}(\mathsf{x} := C_j, \mathsf{w} := (s_j, r_j), m_j)$
        for all $k \in [w]$ do $h_{k,j} := \mathsf{H}_k(C_j)$
    $s := \sum_{j=1}^{\mu} s_j \pmod{q}$
    for all $k \in [w]$ do $h_k = \sum_{j=1}^{\mu} h_{k,j} \pmod{q}$
    $D := G^s \prod_{k=1}^{w} G_k^{h_k}$
    $\sigma' \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}, D)$
    $r := \sum_{j=1}^{\mu} r_j \pmod{q}$
    return $\sigma := (\{(\sigma', D, \mathsf{pk})\}, \{(\pi_j, C_j, m_j)\}_{j \in [\mu]}, r)$, where $\mathsf{pk}$ is the public key corresponding to $\mathsf{sk}$.

$\mathsf{Merge}(\{\sigma_i, M_i, PK_i\}_{i=1}^{n})$:
    parse $\sigma_i$ as $(L_i, R_i, r_i)$
    $L := \bigcup_{i \in [n]} L_i, R := \bigcup_{i \in [n]} R_i$
    $r := \sum_{i=1}^{n} r_i \pmod{q}$
    return $\sigma := (L, R, r)$ (where sets are represented in lexicographic order to hide how they were formed).

$\mathsf{Verify}(\sigma, M, PK)$:
    parse $\sigma$ as $(L, R, r)$, with $L := \{(\sigma'_i, D_i, \mathsf{pk}_i)\}_{i \in [n]}$ and $R := \{(\pi_j, C_j, m_j)\}_{j \in [\mu]}$
    if $\{m_j\}_{j \in [\mu]} \neq M$ then return $0$
    if $\{\mathsf{pk}_i\}_{i \in [n]} \neq PK$ then return $0$
    if $C_j = C_{j'}$ for $j \neq j'$ then return $0$
    for all $j \in [\mu]$ do
        if $\mathsf{SoK}[\mathcal{L}^{\mathsf{ped}}$ or $\mathcal{L}^{\mathsf{ped'}}].\mathsf{Verify}(\mathsf{x} = C_j, \pi_j, m_j) = 0$ then return $0$
    for all $i \in [n]$ do
        if $\mathsf{Sig.Verify}(\sigma'_i, D_i, \mathsf{pk}_i) = 0$ then return $0$
        for all $k \in [w]$ do $h_{k,j} := \mathsf{H}_k(C_j)$
    return $H^r \prod_{i=1}^{n} D_i = \prod_{j=1}^{\mu} (C_j \prod_{k=1}^{w} G_k^{h_{k,j}})$
```

Fig. 8. Constructions for UOS. We mark steps only present in the variant based on a group of unknown order in blue; and steps only present in the lattice variant in teal.

4.2 Secure Variant from Groups of Unknown Order

If we require that every signature of knowledge π additionally prove that the s contained in the witness is positive, then to carry out the above attack, the adversary would need to find a (set of) value(s) s whose sum is positive, but *is zero modulo the order of* $\mathbb{G}$. Then, we can use any adversary who can carry out the attack described above to take roots in the group $\mathbb{G}$, which should be hard to do in a group of unknown order. However, using groups of unknown order is very costly in terms of modulus size, since the parameters scale closer to RSA than to elliptic curves [14].

4.3 Secure Variant from Lattices

We could instead make sure that we can use any adversary who carries out the attack described above to solve the *short integer solution* (SIS) problem. We can embed a SIS instance by using several additional generators $G_1, \ldots, G_w$ and several hash functions $\mathsf{H}_1, \ldots, \mathsf{H}_w$, and by modeling each hash function as a random oracle.

During signing, the signer does almost everything as before. However, she additionally computes $h_{k,j} = \mathsf{H}_k(C_j)$ for each $k \in [1, \ldots, w]$ and $j \in [1, \ldots, \mu]$, sets $h_k = \sum_{j=1}^{\mu} h_{k,j}$, and changes D to be $D = G^s \prod_{k=1}^{w} G_k^{h_k}$. The third step of the verification algorithm now consists of checking that

$$H^r \prod_{i=1}^{n} D_i = \prod_{j=1}^{\mu} (C_j G_1^{\mathsf{H}_1(C_j)} \ldots G_w^{\mathsf{H}_w(C_j)}).$$

If an adversary now succeeds in adding new commitments, either we can use her to solve for the discrete logarithm relationship of some of the generators, or we can use her to solve an instance of the SIS problem which we can embed into the hash values (by means of the random oracle assumption).

4.4 Security Analysis

In Figure 8, we describe our constructions formally.

Theorem 1. *The construction based on lattices described in Figure 8, including teal steps, is unforgeable (Definition 3) assuming that (a) the* $\mathsf{SIS}(w, m, q, \sqrt{m})$ *problem is hard (where the parameters are w hash functions, m random oracle queries, and group order q), (b) the discrete logarithm problem is hard in for group generation algorithm* GroupGen*, (c) the signature scheme* Sig *is secure, and (d) the signature of knowledge scheme* SoK *is secure.*

We prove Theorem 1 in Section 4.4.1.

Theorem 2. *Both variants of the construction described in Figure 8 are history-hiding (Definition 4) with respect to equivalence relation $\equiv_{weak}$ assuming that the signature of knowledge scheme* SoK *is secure.*

We prove Theorem 2 in Section 4.4.2.

Theorem 3. *The construction based on groups of unknown order described in Figure 8, including blue steps, is unforgeable (Definition 3) assuming that (a) the discrete logarithm problem is hard in group $\mathbb{G}$, (b) finding roots is hard for group generation algorithm* GroupGen *(as it produces groups off unknown order), (c) the signature scheme* Sig *is secure, and (d) the signature of knowledge scheme* SoK *is secure.*

The proof of Theorem 3 may be found in the full version of this paper.

4.4.1 Proof of Theorem 1 To prove this theorem we rely on reducing to discrete log in the final step. Achieving a tight reduction proves difficult, since if the reduction included the discrete log challenge in any single commitment the adversary would have to exclude that specific commitment while keeping the corresponding left side fixed for it to be possible to solve the challenge. Similarly, trying to embed the challenge in any one leaf of the tree when producing a signature does not yield a tight reduction. To avoid this we must instead take special care to embed the challenge throughout an entire signature.

Lemma 1 shows how a discrete log challenge P may be embedded throughout the values produced by signing while maintaining the same distribution as honestly produced signatures. Given this lemma, we can exploit the properties of additive secret sharing to construct the values we will need for our reduction.

Lemma 1. *Consider the distribution*

$$((D'_1, \dots, D'_n), ((C_1^1, \dots C_1^{\mu_1}), \dots, (C_n^1, \dots C_n^{\mu_n})), r)$$

where

- *P, G and H are generators in a group of prime order q,*
- *s, t, r are independent uniformly random values modulo q,*
- *$D'_i = G^{s_i^{\mathsf{left}}} P^{t_i^{\mathsf{left}}}$ for additive secret sharings $\langle s \rangle = (s_i^{\mathsf{left}})_{i \in [n]}$ and $\langle t \rangle = (t_i^{\mathsf{left}})_{i \in [n]}$,*
- *$C_i^j = G^{s_{i,j}^{\mathsf{right}}} P^{t_{i,j}^{\mathsf{right}}} H^{r_{i,j}}$ for fresh additive secret sharings $(s_i^{\mathsf{right}})_{i \in [n]}$, $(t_i^{\mathsf{right}})_{i \in [n]}$ and $(r_i)_{i \in [n]}$ of s, t and r, with secondary sharings $\langle s_i^{\mathsf{right}} \rangle = (s_{i,j}^{\mathsf{right}})_{j \in [\mu_j]}$, $\langle t_i^{\mathsf{right}} \rangle = (t_{i,j}^{\mathsf{right}})_{j \in [\mu_j]}$ and $\langle r_i \rangle = (r_{i,j})_{j \in [\mu_j]}$.*

Then, the following properties hold:

1. *For any strict subset $I \subset [n]$, $\{t_i^{\mathsf{left}}\}_{i \in I}$ is independent of $\{t_{i,j}^{\mathsf{right}}\}_{i \in [n], j \in [\mu_i]}$.*
2. *For any strict subset $I \subset \{(i,j) | i \in [n], j \in [\mu_i]\}$, $\{t_{i,j}^{\mathsf{right}}\}_{(i,j) \in I}$ is independent of $\{t_i^{\mathsf{left}}\}_{i \in [n]}$.*
3. *The distribution is independent of t and the subsequent choices of $\{t_i^{\mathsf{left}}\}_{i \in [n]}$ and $\{t_{i,j}^{\mathsf{right}}\}_{i \in [n], j \in [\mu_i]}$.*

4. The distribution conditioned on t, s and subsequent choices of $\{t_i^{\mathit{left}}\}_{i\in[n]}$, $\{t_{i,j}^{\mathit{right}}\}_{i\in[n],j\in[\mu_i]}$ is independent of the choices of $\{s_i^{\mathit{right}}\}_{i\in[n]}$.

Note that (3) and (4) imply that the described values would be indistinguishable to those produced by the construction in Figure 8 (where each D_i is set to $D'_i \prod_{j=1}^{\mu_i} G^{\mathsf{H}(C_i^j)}$). The construction fits the case where all t_i^{left} and $t_{i,j}^{\mathit{right}}$ are 0 and $\{s_i^{\mathit{right}}\}_{i\in[n]} = \{s_i^{\mathit{left}}\}_{i\in[n]}$.

Proof. We prove the four parts of the lemma separately. First, we note that (1) and (2) follow directly from the properties of additive secret sharing.

(3) follows from the fact that for every fixed value $((D'_1,\ldots,D'_n),((C_1^1,\ldots C_1^{\mu_1}),\ldots,(C_n^1,\ldots C_n^{\mu_n})),r)$ in the distribution and for every choice of values $\{t_i^{\mathsf{left}}\}_{i\in[n]}$, $\{t_{i,j}^{\mathsf{right}}\}_{i\in[n],j\in[\mu_i]}$ and $\{r_{i,j}\}_{i\in[n],j\in[\mu_i]}$ (consistent with r), there exists a unique choice of values $\{s_i^{\mathsf{left}}\}_{i\in[n]}$ and $\{s_{i,j}^{\mathsf{right}}\}_{i\in[n],j\in[\mu_i]}$ that explains $((D'_1,\ldots,D'_n),((C_1^1,\ldots C_1^{\mu_1}),\ldots,(C_n^1,\ldots C_n^{\mu_n})),r)$.

(4) follows from the fact that for every fixed t, s, r and subsequent choice of $\{t_{i,j}^{\mathsf{right}}\}_{i\in[n],j\in[\mu_i]}$, for every choice of $\{s_{i,j}^{\mathsf{right}}\}_{i\in[n],j\in[\mu_i]}$ and every set $\{C_i^j\}_{i\in[n],j\in[\mu_i]}$ such that $G^sP^tH^r = \prod_{i\in[n],j\in[\mu_i]} C_i^j$, there exists a unique choice of $\{r_{i,j}\}_{i\in[n],j\in[\mu_i]}$ such that $G^{s_{i,j}^{\mathsf{right}}}P^{t_{i,j}^{\mathsf{right}}}H^{r_{i,j}} = C_i^j$.

Now, we move on to prove Theorem 1.

Proof (of Theorem 1). We reduce the unforgeability of our lattice-based construction to the assumptions enumerated in the theorem. The construction relies on the hardness of the SIS and discrete log problems, as well as the security of the underlying signature scheme and the signature of knowledge scheme.

In the following hybrids we will consider an adversary producing a verifying forgery σ^*, M^*, PK^*, where σ^* is of the form $(L^*,R^*,r^*) = (\{(\sigma_z,D_z,\mathsf{pk}_z)\}_{z=1}^{n^*}, \{(\pi_j,C_j,m_j)\}_{j=1}^{\mu^*}, r^*)$. We define a reduction R in a sequence of hybrids:

1. In this hybrid, the reduction runs the challenger according to the instructions in Figure 6.
2. In this hybrid, the reduction aborts if it gets a forged (underlying) signature. This hybrid is indistinguishable from the previous one by the unforgeability of the underlying signature scheme.
   At this point, since PK^* must all belong to honest parties, all signatures must be generated by the challenger, allowing us to only consider honestly generated D_z.
3. In this hybrid, the reduction uses a trapdoor to simulate the SoKs.
   This hybrid is indistinguishable from the previous one by the zero knowledge property of the SoK scheme.
4. In this hybrid, the reduction aborts if it cannot extract a witness from any signature of knowledge.
   This hybrid is indistinguishable from the previous one by the simulation extractability of the SoK scheme. At this point the reduction knows a witness for each commitment which is part of a verifying forgery.

5. In the following hybrid the reduction aborts if the adversary provides a verifying forgery where $C_j = G^{s_j} H^{r_j}$ for each $j \in [\mu^*]$, but $r^* \neq \sum_{j=1}^{\mu^*} r_j$. That is, the reduction aborts if the provided r^* does not correspond appropriately to the witnesses extracted from the sigantures of knowledge.
   This is indistinguishable from the previous hybrid as an adversary providing a forgery where this is the case may be used to solve a discrete log challenge H base G.
   Each G_k for $k \in [w]$ may be chosen at setup by the reduction as a uniform power of G. Since the forgery is valid, $H^{r^*} \prod_{z=1}^{n^*} D_z = \prod_{j=1}^{\mu^*} (C_j \prod_{k=1}^{w} G_k^{\mathsf{H}_k(C_j)})$ must hold. The reduction knows a witness for each commitment, and the exponents for D_z, which it itself must have produced in response to signing queries. It can then find the discrete logarithm of H, since $r^* - \sum_{j=1}^{\mu^*} r_j \neq 0$.
6. In the following two hybrids we will address the case where the forgery is a valid merging of the outputs of the signing oracle queries, with some extra commitments $C_1, \ldots, C_v$ on the right-hand side. As all other commitments are part of a verifying signature and $r^* = \sum_{j=1}^{\mu^*} r_j$ the extra commitments must not affect the product in the verification formula, i.e. $\prod_{j=1}^{v} (G^{s_j} \prod_{k=1}^{w} G_k^{\mathsf{H}_k(C_j)}) = 1$.
   In this hybrid, the reduction will abort if either $\prod_{j=1}^{v} G^{s_j} \neq 1$ or $\prod_{k=1}^{v} G_k^{\mathsf{H}_k(C_j)} \neq 1$ for any $k \in [w]$. In the next hybrid we will exploit the separation of these generators to embed an instance of the SIS problem, where each generator G_k may be used for a separate dimension of the problem. This hybrid is indistinguishable from the previous by a reduction to a discrete log challenge. For notational convenience, throughout the rest of this hybrid, we will denote G as G_0 and define $s_{0,j} = s_j$ and $s_{k,j} = \mathsf{H}_k(C_j)$. Note the reduction knows both the powers s_j and $\mathsf{H}_k(C_j)$. To find the discrete log of a challenge Y base X the reduction proceeds by first choosing G_k for $k \in [w] \cup \{0\}$ as $X^{a_k} Y^{b_k}$ where a_k and b_k are uniformly random integers modulo q.
   For an adversary successfully producing a forgery where
   $(\prod_{k \in [w] \cup \{0\}} \prod_{j=1}^{v} G_k^{s_{k,j}}) = 1$ but there is an i such that:

$$\prod_{j=1}^{v} G_i^{s_{i,j}} = Z \neq 1, \quad \prod_{\substack{k \in [w] \cup \{0\} \\ k \neq i}} (\prod_{j=1}^{v} G_k^{s_{k,j}}) = Z^{-1} \neq 1,$$

   the reduction may find the discrete log with probability at least 1 - 1/q. The only case where a discrete log cannot be found is when

$$a_i(\sum_{j=1}^{v} s_{i,j}) = \sum_{\substack{k \in [w] \cup \{0\} \\ k \neq i}} a_k(\sum_{j=1}^{v} s_{k,j}).$$

   Consider the distribution of $G_0, \ldots, G_k$ provided to the adversary by the reduction. A fixed G_i may have been produced by any choice of $a_i \in \mathbb{Z}_q$ along with the one possible corresponding b_i, where each of these cases is indistinguishable to the adversary. When all other values are fixed the above equality may only hold for exactly one value of a_i, thus the reduction will only fail to find a discrete log of Y with probability $1/q$.

7. In this hybrid, the reduction embeds an instance of the $\mathsf{SIS}(w, m, q, \sqrt{m})$ problem by programming the random oracle. Recall we are still focusing on the case where the forgery is a valid merge of the outputs of the signing oracle queries, with some extra commitments $C_1, \ldots, C_v$ on the right-hand side. This hybrid now aborts if this is the case.
The indistinguishability of this hybrid from the previous will follow from the hardness of the $\mathsf{SIS}(w, m, q, \sqrt{m})$ problem. For a bound m on the number of random oracle queries made by the adversary, we may consider a uniformly random matrix $\mathbf{A} \in \mathbb{Z}_q^{w \times m}$ provided by an $\mathsf{SIS}(w, m, q, \sqrt{m})$ challenger. We let $\mathbf{A}_{i,j}$ denote the jth entry of the ith row of $\mathbf{A}$.
For each distinct query to the random oracle we may embed one column of the matrix $\mathbf{A}$ in the oracle responses. Specifically, for the ith unique commitment C queried to the random oracle we define $\mathsf{H}_k(C) = \mathbf{A}_{k,i}$ for each $k \in [w]$. The output distribuition of the random oracle is unchanged by this as each entry $\mathbf{A}_{i,j}$ is uniform and independent.
Due to the previous hybrid we know that verification with only extra commitments $C_1, \ldots, C_v$ implies $\prod_{k=1}^{v} G_k^{\mathsf{H}_k(C_j)} = 1$ for all $k \in [w]$. Thus the adversary has provided commitments such that $\sum_{j=1}^{v} \mathsf{H}_k(C_j) = 0 \mod q$ for each $k \in [w]$.
Let $t(j)$ be the index of the first query of C_j to the oracle. The adversary will then have found indices $t(j)$, such that $\sum_{j=1}^{\mu} \mathbf{A}_{k,t(j)} = 0$ for each $k \in [w]$. This provides a solution to the $\mathsf{SIS}(w, m, q, \sqrt{m})$ instance defined by $\mathbf{A}$, as the vector $\mathbf{v}$ which is one exactly for each index in $\{t(j)\}_{j \in [v]}$ and zero otherwise satisfies $\mathbf{A} \cdot \mathbf{v} = \mathbf{0}$. Note $\mathbf{v}$ satisfies the length requirements as each commitment must be unique and the ℓ_2-norm of a zero-one vector of dimension m is bounded by $\sqrt{m}$.
8. In this hybrid, the reduction embeds a discrete log challenge P base G. An adversary breaking the unforgeability of our UOS may be used to find the discrete log with high probability. In this hybrid the H and G_k for $k \in [w]$ are chosen as uniform powers of G. The reduction proceeds as follows:
   (a) For each query to the oracle $\mathsf{Merge}\mathcal{O}(T_i)$, instead of using the $\mathsf{SignAndMerge}$ procedure in Figure 3 the challenger generates values for the honest leaves following the structure of Lemma 1. This produces $((D'_1, \ldots, D'_n),$ $((C_1^1, \ldots, C_1^{\mu_1}), \ldots, (C_n^1, \ldots, C_n^{\mu_n})), r)$, and the reduction sets
   $D_i := D'_i \prod_{j=1}^{\mu_i} \prod_{k=1}^{w} G_k^{\mathsf{H}_k(C_i^j)}$. Now, D_i and $(C_i^1, \ldots C_i^{\mu_i})$ correspond to the values of the ith honest leaf. The reduction produces the necessary signatures and simulates each required SoK.
   From properties (3) and (4) of Lemma 1, it follows that this is indistinguishable from the original distribution. The distribution produced by the $\mathsf{SignAndMerge}$ procedure corresponds to the case where all t_i^{left} and $t_{i,j}^{\mathsf{right}}$ are 0, and $(s_i^{\mathsf{left}})_{i \in [n]} = (s_i^{\mathsf{right}})_{i \in [n]}$.
   (b) Say the adversary produces some forgery σ^*, M^*, PK^* winning the EUF game, where σ^* is of the form $(L^*, R^*, r^*) = (\{(\sigma_z, D_z, \mathsf{pk}_z)\}_{z=1}^{n^*}, \{(\pi_j,$

$C_j, m_j)\}_{j=1}^{\mu^*}, r^*)$. We may start by removing elements from L^* and R^* where the adversary has used the entire content of a signature σ_i produced by the reduction on $M_i^h \subseteq M^*$, such that $L_i^h \subseteq L^*$ and $R_i^h \subseteq R^*$. The forged signature σ^* is updated to $(L^* \setminus L_i^h, R^* \setminus R_i^h, r^* - r_i^h)$. Messages which no longer appear in R^* are removed from M^*, and public keys which no longer appear in L^* are removed from PK^*. This must leave R^* non-empty, as a winning forgery must satisfy $\mathsf{OutsideSpan}(M^*, PK^*, \{(PK_i^h, M_i^h)\}_{i=1}^{q_s})$, i.e. M^* cannot be produced as the union of honestly signed message sets. Importantly, if the original forgery verified, then $H^{r^*} \prod_{D \in L^*} D = \prod_{C \in R^*} C \prod_{k=1}^{w} G_k^{\mathsf{H}_k(C)}$ is maintained. We update $n^* := |L^*|$ and $\mu^* := |R^*|$.

There are three cases for the contents of L^* and R^*; we state them now and analyze them below.

i. L^* is empty and R^* contains only fresh C's (not produced by the reduction).
ii. At least one C was produced by the reduction (in response to a signing oracle query), but is now linked to a new message via a new signature of knowledge.
iii. The above don't hold, and for at least one signing oracle query, either a D is missing from L^* or a C is missing from R^*.

We consider these cases separately.

(i) This option is excluded, as the reduction did not abort in hybrid (7).

(ii) The adversary reuses a commitment $C_j = G^{s_j^{\mathsf{right}}} P^{t_j^{\mathsf{right}}} H^{r_j}$ with a new signature of knowledge (on a new message) separately. Extracting the witness from the SoK would give s, r such that $C_j = G^s H^r$. This clearly allows finding the discrete log of P base G when $t_j^{\mathsf{right}} \neq 0$ which is the case except with probability $1/q$. The distribution seen by the adversary is independent of t_j^{right} by property (3) of Lemma 1

(iii) Now, we move on to consider the case where the adversary *did not* reuse a commitment with a new signature of knowledge. Verification requires $H^{r^*} \prod_{z=1}^{n^*} D_z = \prod_{j=1}^{\mu^*} (C_j \prod_{k=1}^{w} G_k^{\mathsf{H}_k(C_j)})$. The commitments C_j were either produced earlier by the challenger (such that $C_j = G^{s_j^{\mathsf{right}}} P^{t_j^{\mathsf{right}}} H^{r_j}$ for known s_j^{right}, t_j^{right} and r_j) or constructed by the adversary with an accompanying signature of knowledge π (such that $s_j^{\mathsf{right}}, r_j$ satisfying $C_j = G^{s_j^{\mathsf{right}}} H^{r_j}$ can be extracted; this may be regarded as a special case where $t_j^{\mathsf{right}} = 0$).
Since we did not abort in hybrid (2), and the forgery may not contain any corrupt signer public keys, we may be certain all $D_z \in L^*$ were signed by the reduction. Therefore the reduction knows s_z^{left}, t_z^{left} such that $D_z = G^{s_z^{\mathsf{left}}} P^{t_z^{\mathsf{left}}}$ for each $z \in [n^*]$.
The reduction may find the discrete log of P as long as $\sum_{j \in [n^*]} t_j^{\mathsf{left}} \neq \sum_{j \in [\mu^*]} t_j^{\mathsf{right}}$, we argue that the adversary cannot avoid this with non-negligible probability.

For any $L_i^h \subset L^*$ at least one element of R_i^h must not be in R^*. We denote the included subset as $R' \subset R_i^h$. Due to property (2) from Lemma 1, the difference in powers of P between L_i^h and R', $\sum_{D_z \in L_i^h} t_z^{\mathsf{left}} - \sum_{C_j \in R'} t_j^{\mathsf{right}}$ will be uniformly random and independent. Therefore even if the adversary knew all remaining t_z^{left}, t_j^{right} they would not be able to make $\sum_{j\in[n^*]} t_j^{\mathsf{left}} = \sum_{j\in[\mu^*]} t_j^{\mathsf{right}}$ with probability better than chance, $1/q$.
The case where $R_i^h \subset R^*$ and $L_i^h \not\subset L^*$ is largely analogous, but using property (1). If both L_i^h and R_i^h are partially included the difference in powers of P will also be uniform and independent following both properties (1) and (2).

4.4.2 Proof of Theorem 2

Proof (of Theorem 2). We reduce the history hiding of our UOS construction to the simulatability property of the signature of knowledge scheme or the hiding property of the commitment scheme. (Since the Pedersen commitment scheme used in our construction is perfectly hiding, this does not require an additional assumption.) We define a reduction R in a sequence of hybrids as follows:

1. In this hybrid, the reduction runs the challenger according to the instructions in Figure 7.
2. In this hybrid, when answering a SignAndMerge query, the reduction uses a trapdoor to simulate SoKs.
   This hybrid is indistinguishable from the previous one by the simulatability of the SoK scheme.
3. In this hybrid, when answering a SignAndMerge query, for each honest leaf $i \in H$ containing μ_i messages, the reduction computes the right side R of the signature as follows:
   - For $j \in [\mu_i]$, it picks the commitment $C_{i,j}$ as a random element of $\mathbb{G}$.

   It then independently computes r_i as follows:
   - For $j \in [\mu_i]$, it picks $r_{i,j}$ at random (from the appropriate space; either at random modulo q, or as a random 2κ-bit integer).
   - It computes r_i as the modular or integer sum of the $r_{i,j}$'s. (Note that when the modular sum is used, this is the same as choosing r_i at random directly, without going through the step of choosing the $r_{i,j}$'s.)

   It computes the final R and r as per the merge algorithm (including the values from the corrupt leaves). Finally, the reduction produces the left side L of the signature as follows, to ensure a total lack of coupling between elements of R and elements of L:
   - It computes D as $(\prod_{C\in R} C \prod_{k=1}^{w} G_k^{\mathsf{H}_k(C)})/H^r$, and computes the individual D_i's (for honest leaves) as a random factoring of D (divided by corrupt leaf D_i's). That is, it produces all but the last honest D_i as random elements of $\mathbb{G}$; it picks the last honest D_i as D divided by all other D_i's (including the corrupt ones).

– It signs the honest D_i's on behalf of the honest parties.

This hybrid is indistinguishable from the previous one because the responses to the challenge $\mathsf{SignAndMerge}$ query are indistinguishable (identical) in the two hybrids, by the perfect hiding property of the commitment scheme; for each choice of factoring of D, there exists a unique consistent multiplicative decomposition of H^r.

Observe that now, there is no link at all between the D_i's (linked to party identities) and the C_j's (linked to messages). So, the outputs of $\mathsf{SignAndMerge}$ on T_0 and T_1 are identically distributed as long as $T_0 \equiv_{weak} T_1$.

5 Performance

To evaluate the performance of our signature scheme, we provide an implementation in `Rust` based on the `curve25519-dalek` crate implementing the `edwards25519` elliptic curve [9] and the Ristretto encoding for points. The signature scheme used is a Schnorr signature. As hash function, we use SHA256 which is hardware accelerated on modern processors. Most of the signature generation and verification is independent of each other. This allows parallel execution on multi-core processors. As our benchmarking system, we use an Intel Core i7-6820HQ CPU at 2.70 GHz for a total of 8 threads.

Figure 9 shows the signing and verification time for an increasing number of messages and different number of hash functions. There are clearly visible and linear steps at multiples of 8 messages, supporting our claim of efficient parallelization. To achieve a difficulty for the SIS problem similar to the discrete logarithm in `edwards25519`, we require 478 dimensions. We used the approach described in [20] with the model put forward by Albrecht *et al.* [4] to estimate the bit-security of $\mathsf{SIS}(w, m, q, \sqrt{m})$. In this case $m = 2^{128}$ is a bound on the number of oracle queries, and q is the order of the `edwards25519` curve, allowing us to find the smallest dimension w providing the necessary difficulty. This results in signatures which require 1 group element and the size of the Schnorr signature (1 group element and 1 scalar) for each merged leaf plus an additional 2 group elements and 2 scalars for each message. Concretely, 100 merged signatures, over 100 messages each for a total of 10,000 messages, requires 1.2 MB.

The merging of signatures consists of adding the randomness, which is independent of the messages and linear in the number of parts. To maintain history hiding, the inputs and outputs need to be sorted, which is possible in $\mathcal{O}(n \log k)$ where n is the total number of messages and k is the number of already ordered signatures to be merged. There are always less or equal signatures to be sorted.

Figure 10 shows the time required for merging signatures. The two dependencies are the number of signatures on the x-axis and we measured this for different numbers of messages per signature. For all experiments with more than one message per signature, we notice that the majority of time is required copying memory and sorting the messages and adding the randomness is marginal. This observation is derived from the fact, that merging with an equal number of messages have the same timings. Each step right are 10 times more signatures

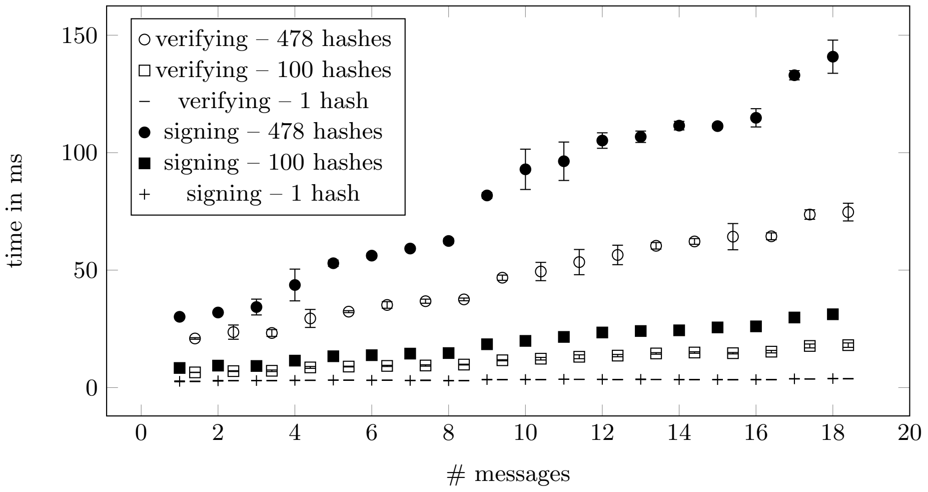

Fig. 9. Signing and verification time depending on number of messages with 8 threads.

but a 10th of the messages per signature, resulting in the same total number. For single message signatures, adding the randomness is noticeable and they are a bit slower than the experiment with an equal number of messages distributed in a 10th of the signatures.

In absolute terms, our implementation handles merging a million messages in less than half a second, making it usable for large datasets.

6 Conclusion

We presented two constructions for UOS schemes, closing the open problem posed by Johnson *et al.* [18]. This is made possible by the fact that our signatures do not have a multiplicative structure. Our first construction is experimentally evaluated as computationally efficient when instantiated with a state-of-the-art elliptic curve implementation, but not compact in terms of signature size. Our second construction is based in SNARKs and produces constant-size signatures, but with a significant performance penalty due to the inherent cost of SNARKs. We finish by pointing out that our first step may lead to more efficient UOS constructions, and hope that it provides techniques useful to close the other open problem of *concatenable* signatures posed in the same paper.

Acknowledgements

This work was partially funded by the Carlsberg Foundation under the Semper Ardens Research Project CF18-112 (BCM), the Sapere Aude: DFF-Starting Grant number 0165-00079B "Foundations of Privacy Preserving and Accountable Decentralized Protocols" and by the European Research Council (ERC)

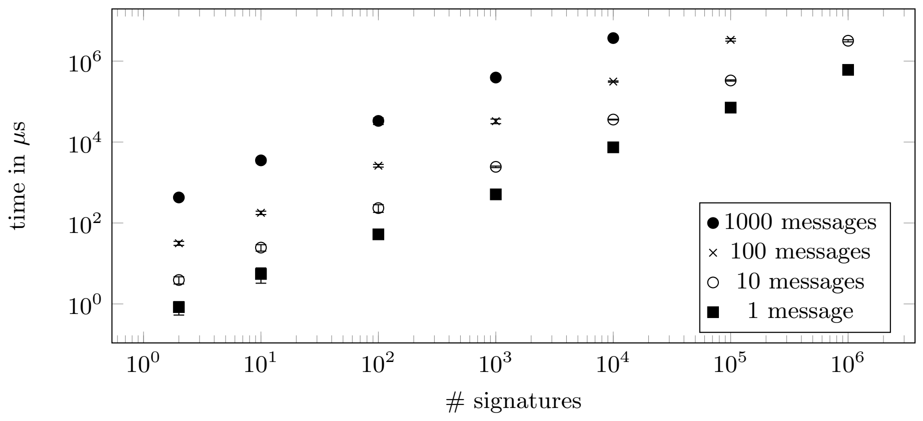

Fig. 10. Merging of signatures with n messages in each signature. The operation is mostly memory bound and depends on the amount of signatures copied. It is independent on the security parameter of the number of hash functions

under the European Unions's Horizon 2020 research and innovation programme under grant agreement No 669255 (MPCPRO) and No 803096 (SPEC). The first author acknowledges support from the Concordium Blockchain Research Center (COBRA) and the DIGIT Centre for Digitalisation, Big Data and Data Analytics at Aarhus University.

References

1. Abiteboul, S., Cautis, B., Fiat, A., Milo, T.: Digital signatures for modifiable collections. In: ARES. pp. 390–399. IEEE Computer Society (2006)
2. Ahn, J.H., Boneh, D., Camenisch, J., Hohenberger, S., shelat, a., Waters, B.: Computing on authenticated data. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 1–20. Springer, Heidelberg (Mar 2012). https://doi.org/10.1007/978-3-642-28914-9_1
3. Ajtai, M.: Generating hard instances of lattice problems (extended abstract). In: 28th ACM STOC. pp. 99–108. ACM Press (May 1996). https://doi.org/10.1145/237814.237838
4. Albrecht, M.R., Cid, C., Faugère, J.C., Fitzpatrick, R., Perret, L.: On the complexity of the BKW algorithm on LWE. Cryptology ePrint Archive, Report 2012/636 (2012), https://eprint.iacr.org/2012/636
5. Altug, S.A., Chen, Y.: Hard isogeny problems over RSA moduli and groups with infeasible inversion. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019, Part II. LNCS, vol. 11922, pp. 293–322. Springer, Heidelberg (Dec 2019). https://doi.org/10.1007/978-3-030-34621-8_11
6. Aranha, D.F., Hall-Andersen, M., Nitulescu, A., Pagnin, E., Yakoubov, S.: Count me in! extendability for threshold ring signatures. Cryptology ePrint Archive, Report 2021/1240 (2021), https://ia.cr/2021/1240
7. Aranha, D.F., Pagnin, E.: The simplest multi-key linearly homomorphic signature scheme. In: Schwabe, P., Thériault, N. (eds.) LATINCRYPT 2019. LNCS,

vol. 11774, pp. 280–300. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-030-30530-7_14
8. Barbulescu, R., Duquesne, S.: Updating key size estimations for pairings. Journal of Cryptology **32**(4), 1298–1336 (Oct 2019). https://doi.org/10.1007/s00145-018-9280-5
9. Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B.Y.: High-speed high-security signatures. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 124–142. Springer, Heidelberg (Sep / Oct 2011). https://doi.org/10.1007/978-3-642-23951-9_9
10. Boneh, D., Bünz, B., Fisch, B.: Batching techniques for accumulators with applications to IOPs and stateless blockchains. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part I. LNCS, vol. 11692, pp. 561–586. Springer, Heidelberg (Aug 2019). https://doi.org/10.1007/978-3-030-26948-7_20
11. Boneh, D., Freeman, D., Katz, J., Waters, B.: Signing a linear subspace: Signature schemes for network coding. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 68–87. Springer, Heidelberg (Mar 2009). https://doi.org/10.1007/978-3-642-00468-1_5
12. Catalano, D., Fiore, D.: Using linearly-homomorphic encryption to evaluate degree-2 functions on encrypted data. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015. pp. 1518–1529. ACM Press (Oct 2015). https://doi.org/10.1145/2810103.2813624
13. Chase, M., Lysyanskaya, A.: On signatures of knowledge. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 78–96. Springer, Heidelberg (Aug 2006). https://doi.org/10.1007/11818175_5
14. Dobson, S., Galbraith, S.D., Smith, B.: Trustless unknown-order groups. Mathematical Cryptology **1**(2), 25–39 (2021), https://journals.flvc.org/mathcryptology/issue/view/6013
15. Engelmann, F., Müller, L., Peter, A., Kargl, F., Bösch, C.: SwapCT: Swap confidential transactions for privacy-preserving multi-token exchanges. PoPETs **2021**(4), 270–290 (Oct 2021). https://doi.org/10.2478/popets-2021-0070
16. Groth, J., Maller, M.: Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part II. LNCS, vol. 10402, pp. 581–612. Springer, Heidelberg (Aug 2017). https://doi.org/10.1007/978-3-319-63715-0_20
17. Hohenberger, S.R.: The cryptographic impact of groups with infeasible inversion. Master's thesis, Massachusetts Institute of Technology (2003), available at http://hdl.handle.net/1721.1/87357
18. Johnson, R., Molnar, D., Song, D.X., Wagner, D.: Homomorphic signature schemes. In: Preneel, B. (ed.) CT-RSA 2002. LNCS, vol. 2271, pp. 244–262. Springer, Heidelberg (Feb 2002). https://doi.org/10.1007/3-540-45760-7_17
19. Kaaniche, N., Jung, E., Gehani, A.: Efficiently validating aggregated iot data integrity. In: BigDataService. pp. 260–265. IEEE Computer Society (2018)
20. Kosba, A., Zhao, Z., Miller, A., Qian, Y., Chan, H., Papamanthou, C., Pass, R., Shi, E., et al.: C $\emptyset$ c $\emptyset$: A framework for building composable zero-knowledge proofs. Cryptology ePrint Archive (2015)
21. McCurley, K.S.: The discrete logarithm problem. In: Proc. of Symp. in Applied Math. vol. 42, pp. 49–74. USA (1990)
22. Micciancio, D., Regev, O.: Worst-case to average-case reductions based on Gaussian measures. In: 45th FOCS. pp. 372–381. IEEE Computer Society Press (Oct 2004). https://doi.org/10.1109/FOCS.2004.72

23. Molnar, D.: Homomorphic signature schemes (2003), bSc. Senior thesis. Harvard College. Available at https://www.dmolnar.com/papers/papers.html
24. Nitulescu, A.: zk-snarks: A gentle introduction. https://www.di.ens.fr/~nitulesc/files/Survey-SNARKs.pdf (2021)
25. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO'91. LNCS, vol. 576, pp. 129–140. Springer, Heidelberg (Aug 1992). https://doi.org/10.1007/3-540-46766-1_9
26. Pöhls, H.C., Samelin, K., Posegga, J., de Meer, H.: Transparent mergeable redactable signatures with signer commitment and applications. Inst. IT-Security Security-Law, Univ. Passau, Passau, Germany (2012)
27. Pöhls, H.C., Samelin, K.: On updatable redactable signatures. In: Boureanu, I., Owesarski, P., Vaudenay, S. (eds.) ACNS 14. LNCS, vol. 8479, pp. 457–475. Springer, Heidelberg (Jun 2014). https://doi.org/10.1007/978-3-319-07536-5_27
28. Rabi, M., Sherman, A.T.: Associative one-way functions: A new paradigm for secret-key agreement and digital signatures. Tech. rep., University of Maryland Institute for Advanced Studies (1993), cS-TR-3183/UMIACS-/R-93-124
29. Rivest, R.L., Shamir, A., Adleman, L.M.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the Association for Computing Machinery **21**(2), 120–126 (1978)
30. Traverso, G., Demirel, D., Buchmann, J.: Homomorphic Signature Schemes - A Survey. Springer Briefs in Computer Science, Springer (2016)
31. Yamakawa, T., Yamada, S., Hanaoka, G., Kunihiro, N.: Self-bilinear map on unknown order groups from indistinguishability obfuscation and its applications. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 90–107. Springer, Heidelberg (Aug 2014). https://doi.org/10.1007/978-3-662-44381-1_6

A Construction Preliminaries

For our construction, we require the following tools. In the description of the assumptions we make about the groups we use, we employ an algorithm $\mathsf{GroupGen}$ which takes in the security parameter and produces a group.

A.1 Discrete Logarithm Assumption

Let $\mathsf{GroupGen}(1^\lambda)$ produce a group $\mathbb{G}$ together with it's order q (s.t. q is a κ-bit integer where $\kappa \in \mathsf{poly}(\lambda)$ such that the group achieves the necessary level of security) and a generator G.

Definition 5 (Discrete Logarithm Assumption [21]). *The discrete logarithm assumption hold for* $\mathsf{GroupGen}$ *if for any PPT adversary* $\mathcal{A}$*, it holds that*

$$\Pr\left[\begin{matrix}\gamma' \leftarrow \mathcal{A}(\mathbb{G}, q, G, H);\\ \gamma' = \gamma\end{matrix} : \begin{matrix}(\mathbb{G}, q, G) \leftarrow \mathsf{GroupGen}(1^\lambda);\\ \gamma \overset{\$}{\leftarrow} [0, \ldots, q-1];\\ H := G^\gamma\end{matrix}\right] \leq \mathsf{negl}(\lambda).$$

A.2 Groups of Unknown Order, and the RSA Assumption

Let $\mathsf{GroupGen}(1^\lambda)$ produce a group $\mathbb{G}$ of order q (s.t. q is a κ-bit integer where $\kappa \in \mathsf{poly}(\lambda)$ such that the group achieves the necessary level of security) and a generator G, *without* revealing q. In one of our constructions, we require that it be hard to find roots in the groups $\mathsf{GroupGen}$ produces; that is, we require that the RSA assumption holds.

Definition 6 (The RSA Assumption [29]). *The RSA assumption holds for a given $e \in \mathbb{Z}$, $e \geq 2$ for* $\mathsf{GroupGen}$ *if for any PPT adversary $\mathcal{A}$, it holds that*

$$\Pr\left[\begin{matrix} R \leftarrow \mathcal{A}(\mathbb{G}, G, H); \\ R^e = H \end{matrix} : \begin{matrix} (\mathbb{G}, G) \leftarrow \mathsf{GroupGen}(1^\lambda); \\ H \xleftarrow{\$} \mathbb{G} \end{matrix}\right] \leq \mathsf{negl}(\lambda).$$

Notice also that in the Definition 6, we need to sample uniform elements in the group. It is possible to sample an element which is statistically indistinguishable from uniform in the group by sampling $x \xleftarrow{\$} \{0, \dots, 2^{2\lambda}\}$ and computing G^x.

A.3 The Short Integer Solution Assumption

For $n, q, m \in \mathbb{N}$, a random matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ defines the lattice $\Lambda_q^\perp(\mathbf{A}) = \{\mathbf{v} \in \mathbb{Z}^m | \mathbf{A}\mathbf{v} = \mathbf{0} \mod q\}$. Given a norm bound $1 \leq \beta < q$, the $\mathsf{SIS}(n, m, q, \beta)$ problem is to find $\mathbf{v} \in \Lambda_q^\perp(\mathbf{A})$ such that $0 < ||\mathbf{v}||_2 \leq \beta$.

Definition 7 (SIS Assumption [3,22]). *Given $n, m, q, \beta \in \mathbb{N}$ The* $\mathsf{SIS}(n, m, q, \beta)$ *assumption holds if for any PPT adversary $\mathcal{A}$, it holds that*

$$|\Pr[\mathbf{v} \leftarrow \mathcal{A}(\mathbf{A}); \mathbf{A} \cdot \mathbf{v} = \mathbf{0} \wedge \mathbf{v} \neq \mathbf{0} \wedge ||\mathbf{v}||_2 \leq \beta : \mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}]| \leq \mathsf{negl}(\lambda).$$

The bit security of $\mathsf{SIS}(n, m, q, \beta)$ may be estimated following the approach described by Kosba *et al.* [20].

A.4 Pedersen Commitments

Pedersen commitments [25] are perfectly hiding and computationally binding commitments which are additively homomorphic. Given a group $\mathbb{G}$ where the discrete logarithm problem is hard, and given two generators G and H, the algorithm $\mathsf{PC.Commit}$ is as follows:

$\mathsf{PC.Commit}(a; r) \rightarrow \mathsf{com}$**:** takes a value a and randomness r and outputs a commitment $\mathsf{com} = G^a H^r$.

Pedersen commitments satisfy the following properties:

Perfect Hiding: a commitment perfectly hides the committed value a.
Computational Binding: no PPT adversary $\mathcal{A}$ can produce a commitment as well as $a_0 \neq a_1$, r_0, r_1 s.t. $\mathsf{PC.Commit}(a_0; r_0) = \mathsf{PC.Commit}(a_1; r_1)$.
Additive Homomorphism: For an efficient operation $\odot$, it holds that

$$\mathsf{PC.Commit}(a, r) \odot \mathsf{PC.Commit}(b, s) = \mathsf{PC.Commit}(a + b, r + s).$$

A.5 Signatures

A signature scheme consists of four algorithms $\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}$ defined as follows:

$\mathsf{Sig.Setup}(1^\lambda) \to \mathsf{p}$ takes the security parameter λ and outputs public parameters p which are implicitly given to all algorithms.
$\mathsf{Sig.KeyGen}() \to (\mathsf{pk}, \mathsf{sk})$ generates a public-secret key pair.
$\mathsf{Sig.Sign}(\mathsf{sk}, m) \to \sigma$ takes a secret key sk and a message $m \in \{0,1\}^*$ and outputs a signature σ.
$\mathsf{Sig.Verify}(\sigma, \mathsf{pk}, m) \to \{0,1\}$ takes a signature σ a public key pk and a message m and outputs a bit.

We require the signature scheme to have existential unforgeability under chosen message attack, meaning that an adversary should have negligible probability of fabricating a valid signature for a message and signer pair not seen before.

A.6 Signatures of Knowledge

Signatures of Knowledge (SoK, [13]) are Arguments of Knowledge which are bound to a message. An SoK scheme for an NP language $\mathcal{L}$ consists of three algorithms $\mathsf{SoK.Setup}, \mathsf{SoK.Sign}, \mathsf{SoK.Verify}$ defined as follows:

$\mathsf{SoK.Setup}(1^\lambda, \mathcal{L}) \to \mathsf{p}$ takes the security parameter λ together with a language $\mathcal{L}$ and outputs the public parameters p.
$\mathsf{SoK.Sign}(x, w, m) \to \pi$ takes a statement x for a witness w and a message m and outputs a signature π.
$\mathsf{SoK.Verify}(x, \pi, m) \to \{0,1\}$ takes a statement x and a signature π with a message and outputs a bit.

A Signature of Knowledge satisfies the properties which are detailed in [16]:

Correctness: any valid witness enables creating a valid signature.
Soundness: it should not be possible for a prover to produce a verifying proof for a statement without knowledge of a corresponding witness.
Zero knowledge: if the public parameters p are generated with a trapdoor, that trapdoor should enable the generation of proofs *without* knowledge knowledge of a witness; these proofs should be indistinguishable from real proofs.
Simulation extractability: this is a stronger notion of simulation soundness, which demands that an adversary shouldn't be able to produce a new verifying proof without knowledge of a witness even if she is given access to simulated proofs.

Note in particular that simulation extractability implies that a signature of knowledge for a given statement and message cannot be used to obtain a valid signtature of knowledge for the same statement and a different message.

Instantiation In our construction we make use of the relation defined by the following NP language:

$$\mathcal{L}^{\mathsf{ped}} := \{C : \exists(s, r) \text{ s.t. } C = H^r G^s\}.$$

We use a Schnorr-style SoK for this relation. In more detail, $\mathsf{SoK.Sign}(\phi = C, w = (r, s), m)$ does the following:

```
$s', r' \xleftarrow{\$} \mathbb{F}$
$C' = H^{r'} G^{s'}$
$c \leftarrow \mathsf{H}(C, C', m)$
$z_s = s' + cs, z_r = r' + cr$
verify $C^c C' = H^{z_r} G^{z_s}$
```

We also make use of an augmented relation $\mathcal{L}^{\mathsf{ped}'}$, where the proof additionally shows that s is positive. In groups of unknown order, we rely on the efficient sigma protocol $\mathsf{ZKPoKRep}$ from [10]. It provides soundness of $\frac{1}{2}$ which requires λ repetitions to achive security.

A.7 Additive and Multiplicative Secret Sharing

In our proofs of security we leverage additive and multiplicative secret sharing. We denote an additive secret sharing modulo q of an element x with n shares, as $\langle x \rangle = (x_1, \ldots, x_n)$ (where q will be obvious from context). it can be produced by selecting $(x_i)_{i \in [n-1]}$ uniformly in $\mathbb{Z}_q$ and setting $x_n = x - \sum_{i=1}^{n-1} x_i \pmod q$. Any vector of $n - 1$ shares will be uniform in $\mathbb{Z}_q$ and independent of x. The secret may be reconstructed as $x = \sum_{i=1}^{n} x_i$.

Similarly, we can define multiplicative secret sharing of an element X in a group $\mathbb{G}$. The first $n - 1$ shares are random elements $X_i \in \mathbb{G}$; X_n is computed as $X/(\prod_{i=1}^{n-1} X_i)$.

B Proof of Theorem 3

Similar to Lemma 1, we provide Lemma 2 to show how a discrete log challenge P may be embedded throughout the values produced by signing while maintaining the same distribution as honestly produced signatures.

Lemma 2. *Consider the distribution*

$$((D_1, \ldots, D_n), ((C_1^1, \ldots C_1^{\mu_1}), \ldots, (C_n^1, \ldots C_n^{\mu_n})), r)$$

where

- *P, G and H are generators in a group $\mathbb{G}$ of unknown order q (where q bounded between 2^κ and $2^{\kappa+1}$),*
- *$C_i^j = G^{s_{i,j}^{\mathsf{right}}} P^{t_{i,j}^{\mathsf{right}}} H^{r_{i,j}}$ for $i \in [n]$ and $j \in [\mu_i]$ with randomly sampled $s_{i,j}^{\mathsf{right}}, t_{i,j}^{\mathsf{right}}, r_{i,j} \in \{0, \ldots, 2^{2\kappa}\}$. Define $s_i^{\mathsf{right}} = \sum_{j=1}^{\mu_i} s_{i,j}^{\mathsf{right}}$ and analogously $t_i^{\mathsf{right}} = \sum_{j=1}^{\mu_i} t_{i,j}^{\mathsf{right}}$ and $r_i = \sum_{j=1}^{\mu_i} r_{i,j}$. Subsequently, define $s = \sum_{i=1}^{n} s_i^{\mathsf{right}}$, $t = \sum_{i=1}^{n} t_i^{\mathsf{right}}, r = \sum_{i=1}^{n} r_i$. Note that these are integer sums.*

– *$D_i = G^{s_i^{\textit{left}}} P^{t_i^{\textit{left}}}$ for $i \in [n]$. To achieve $s = \sum_{i=1}^{n} s_i^{\textit{left}}$ and $t = \sum_{i=1}^{n} t_i^{\textit{left}}$ with $s_i^{\textit{left}}, t_i^{\textit{left}} \geq 0$, sample all uniformly at random from $\{0, \ldots, 2^{2\kappa}\}$. If the sampling undershot and the sum is smaller $\sum_{i=1}^{n} s_i^{\textit{left}} < s$, add the difference to the last element $s_n^{\textit{left}} = s_n^{\textit{left}} + s - \sum_{i=1}^{n} s_i^{\textit{left}}$. If the sampling overshot and the sum is larger than s, add the difference to $s_{n,\mu_n}^{\textit{right}}$ and increase s. Proceed analogously for t. This sampling process ensures that all values are at least as big as if they were sampled uniformly from $\{0, \ldots, 2^{2\kappa}\}$.*

Then, the following properties hold:

1. *For any strict subset $I \subset [n]$, the distribution of $\{P^{t_i^{\textit{left}}}\}_{i \in I}$ is indistinguishable from uniform in $\mathbb{G}$ and independent of $\{P^{t_{i,j}^{\textit{right}}}\}_{i \in [n], j \in [\mu_i]}$.*
2. *For any strict subset $I \subset \{(i,j) | i \in [n], j \in [\mu_i]\}$, the distribution of $\{P^{t_{i,j}^{\textit{right}}}\}_{(i,j) \in I}$ is indistinguishable from uniform in $\mathbb{G}$ and independent of $\{P^{t_i^{\textit{left}}}\}_{i \in [n]}$.*
3. *For any pair of choices of $\{t_i^{\textit{left}}\}_{i \in [n]}, \{t_{i,j}^{\textit{right}}\}_{i \in [n], j \in [\mu_i]}$ and t, the resulting distributions are indistinguishable.*
4. *Consider a fixed $\{t_i^{\textit{left}}\}_{i \in [n]}$, $\{t_{i,j}^{\textit{right}}\}_{i \in [n], j \in [\mu_i]}$, t, $\{s_i^{\textit{left}}\}_{i \in [n]}$ and s. For any pair of choices of $\{s_i^{\textit{right}}\}_{i \in [n]}$, the distributions are indistinguishable.*

Note that (3) and (4) imply that the described values would be indistinguishable to those produced by the construction in Figure 8 (including blue steps). The construction fits the case where all $t_i^{\textit{left}}$ and $t_{i,j}^{\textit{right}}$ are 0 and $\{s_i^{\textit{right}}\}_{i \in [n]} = \{s_i^{\textit{left}}\}_{i \in [n]}$.

Proof (of Lemma 2). We prove the four parts of the lemma separately. First, we note that for any generator G, sampling an exponent x uniformly in $\{0, \ldots, 2^{2\kappa}\}$ yields a value G^x that is statistically indistinguishable from uniformly random in the group $\mathbb{G}$. All values except $s_n^{\text{left}}, t_n^{\text{left}}$ are chosen uniformly from $\{0, \ldots, 2^{2\kappa}\}$; so, G and P to their power look random in $\mathbb{G}$. The final values $G_n^{s^{\text{left}}}$, $P_n^{t^{\text{left}}}$ are uniquely determined by the rest, as they must satisfy $G^s = \prod_{i=1}^{n} G^{s_i^{\text{left}}}$ and $P^t = \prod_{i=1}^{n} P^{t_i^{\text{left}}}$. If the rest of the values were truly uniformly random, $G_n^{s^{\text{left}}}$ and $P_n^{t^{\text{left}}}$ would be as well. So, they, too, must be statistically indistinguishable rom uniformly random; any distinguisher who could tell them apart from random could be used to tell the rest of the values apart from random. Properties (1) and (2) follow by similar logic.

Property (3) follows from the fact that for every fixed value $((D_1, \ldots, D_n), ((C_1^1, \ldots C_1^{\mu_1}), \ldots, (C_n^1, \ldots C_n^{\mu_n})), r)$ in the distribution and for every choice of values $\{t_i^{\text{left}}\}_{i \in [n]}$, $\{t_{i,j}^{\text{right}}\}_{i \in [n], j \in [\mu_i]}$ and $\{r_{i,j}\}_{i \in [n], j \in [\mu_i]}$ (consistent with r), there exists a unique set of values $\{G^{s_i^{\text{left}}}\}_{i \in [n]}$ and $\{G^{s_{i,j}^{\text{right}}}\}_{i \in [n], j \in [\mu_i]}$ that explains $((D_1, \ldots, D_n), ((C_1^1, \ldots C_1^{\mu_1}), \ldots, (C_n^1, \ldots C_n^{\mu_n})), r)$. Recall the values $\{G^{s_i^{\text{left}}}\}_{i \in [n]}$ and $\{G^{s_{i,j}^{\text{right}}}\}_{i \in [n], j \in [\mu_i]}$ are indistinguishable from uniform in $\mathbb{G}$.

Property (4) follows from the fact that for every fixed value $((D_1, \ldots, D_n), ((C_1^1, \ldots C_1^{\mu_1}), \ldots, (C_n^1, \ldots C_n^{\mu_n})), r)$ in the distribution and for every consistent choice of values $\{t_i^{\text{left}}\}_{i \in [n]}$, $\{t_{i,j}^{\text{right}}\}_{i \in [n], j \in [\mu_i]}$, $\{s_i^{\text{left}}\}_{i \in [n]}$ and $\{s_{i,j}^{\text{right}}\}_{i \in [n], j \in [\mu_i]}$, there exists a unique set of values $\{H^{r_{i,j}}\}_{i\in[n],j\in[\mu_i]}$ (consistent with r) that explains $((D_1,\dots,D_n),((C_1^1,\dots C_1^{\mu_1}),\dots,(C_n^1,\dots C_n^{\mu_n})),r)$.

Proof (of Theorem 3). We reduce the unforgeability of this scheme to the assumptions enumerated in the theorem; namely, to the assumptions that (a) the discrete logarithm problem is hard in group $\mathbb{G}$, (b) finding roots is hard in group $\mathbb{G}$ (as it has unknown order), (c) the signature scheme Sig is secure, and (d) the signature of knowledge scheme SoK is secure.

We define a reduction R in a sequence of hybrids as follows:

1.-4. Hybrids (1)-(4) proceed exactly as in the proof of Theorem 1.

5. As in hybrid (5) of the proof Theorem 1, the reduction aborts if the adversary provides a verifying forgery σ^*, M^*, PK^* where σ^* is $(L^*, R^*, r^*) = (\{(\sigma_z, D_z, \mathsf{pk}_z)\}_{z=1}^{n^*}, \{(\pi_j, C_j, m_j)\}_{j=1}^{\mu^*}, r^*)$, but $r^* \neq \sum_{j=1}^{\mu^*} r_j$. That is, the reduction aborts if the provided r^* does not correspond appropriately to the witnesses extracted from the sigantures of knowledge.
   This is indistinguishable from the previous hybrid as an adversary providing a forgery where this is the case may be used to break the RSA assumption in $\mathbb{G}$. The reduction may choose the generator H in such a way that it knows e s.t. $H = G^e$.
   For a valid forgery, $H^{r^*}\prod_{z=1}^{n^*} D_z = \prod_{j=1}^{\mu^*} C_j$ must hold. As the reduction knows a witness for each commitment and all D_z must be produced by the reduction this corresponds to knowing powers a, b and $c = \sum_{j=1}^{\mu^*} r_j$ such that $H^{r^*}G^a = H^b G^c$. Thus the reduction may find powers $G^{a+er^*} = G^{c+eb}$ where $a + er^* \neq c + eb$. The positive difference between these powers is an integer which is divisible by the order of the group $\mathbb{G}$. Using this it is possible to find arbitrary roots by using the extended Euclidean algorithm.
6. In this hybrid, the reduction produces signatures according to a different but indistinguishable distribution using a group element P. The reduction may then use an adversary breaking the unforgeability of our UOS to break the RSA assumption with high probability. In this hybrid the reduction may choose the generators H and P in such a way that it their discrete logarithms base G. The reduction proceeds as follows:
   (a) For each query to the oracle $\mathsf{Merge}\mathcal{O}(T_i)$, instead of using the $\mathsf{SignAndMerge}$ procedure in Figure 3 the challenger generates values for the honest leaves following the structure of Lemma 2. This produces $((D_1,\dots,D_n),$ $((C_1^1,\dots,C_1^{\mu_1}),\dots,(C_n^1,\dots,C_n^{\mu_n})),r)$, where D_i and $(C_i^1,\dots C_i^{\mu_i})$ correspond to the values of the ith honest leaf. The challenger produces or simulates the necessary signatures and SoK.
   From properties (3) and (4) of Lemma 2, it follows that this is indistinguishable from the original distribution. The distribution produced by the $\mathsf{SignAndMerge}$ procedure corresponds to the case where all t_i^{left} and $t_{i,j}^{\mathsf{right}}$ are 0, and $(s_i^{\mathsf{left}})_{i\in[n]} = (s_i^{\mathsf{right}})_{i\in[n]}$.
   (b) Say the adversary produces some forgery σ^*, M^*, PK^* winning the EUF game, where σ^* is of the form $(L^*, R^*, r^*) = (\{(\sigma_z, D_z, \mathsf{pk}_z)\}_{z=1}^{n^*}, \{(\pi_j,$ $C_j, m_j)\}_{j=1}^{\mu^*}, r^*)$. We may start by removing elements from L^* and R^*

where the adversary has used the entire content of a signature σ_i produced by the reduction on $M_i^h \subseteq M^*$, such that $L_i^h \subseteq L^*$ and $R_i^h \subseteq R^*$. The forged signature σ^* is updated to $(L^* \setminus L_i^h, R^* \setminus R_i^h, r^* - r_i^h)$. Messages which no longer appear in R^* are removed from M^*, and public keys which no longer appear in L^* are removed from PK^*. This must leave R^* non-empty, as a winning forgery must satisfy $\mathsf{OutsideSpan}(M^*, PK^*, \{(PK_i^h, M_i^h)\}_{i=1}^{q_s})$, i.e. M^* cannot be produced as the union of honestly signed message sets. Importantly, if the original forgery verified, then $H^{r^*} \prod_{D \in L^*} D = \prod_{C \in R^*} C$ is maintained. We update $n^* := |L^*|$ and $\mu^* := |R^*|$.

Since we did not abort in hybrid (2), and since the forgery may not contain any corrupt signer public keys, we may be certain all $D_z \in L^*$ were signed by the reduction. So, $D_z = G^{s_z^{\mathsf{left}}} P^{t_z^{\mathsf{left}}}$ for values s_z^{left} and t_z^{left} known to the reduction; the reduction is therefore able to find t^{left} and s^{left} such that $G^{s^{\mathsf{left}}} P^{t^{\mathsf{left}}} = \prod_{z=1}^{n^*} D_z$.

There are three cases for the contents of L^* and R^*; we state them now and analyze them below.

i. L^* is empty and R^* contains only fresh C's (not produced by the reduction).
ii. At least one C was produced by the reduction (in response to a signing oracle query), but is now linked to a new message via a new signature of knowledge.
iii. The above doesn't hold, and for at least one signing oracle query, either a D is missing from L^* or a C is missing from R^*.

We consider these cases separately.

(i) When L^* is empty and R^* contains only fresh commitments $C_1, \ldots, C_v$ verification implies $H^{r^*} = \prod_{j=1}^{v} C_j$. As all sigantures of knowledge have been successfully extracted, the reduction now knows powers such that $H^{r^*} = \prod_{j=1}^{v} G^{s_j} H^{r_j}$. Due to hybrid (5) $r^* = \sum_{j=1}^{v} r_j$, implying $\prod_{j=1}^{v} G^{s_j} = 1$. As this construction uses $\mathcal{L}^{\mathsf{ped}'}$ we additionally know that each s_j is positive. The order of group $\mathbb{G}$ must divide $\sum_{j=1}^{v} s_j$ and may be used as previously to find arbitrary roots within the group, contradicting the RSA assumption.

(ii) Here we consider an adversary which reuses a commitment $C_j = G^{s_j^{\mathsf{right}}} P^{t_j^{\mathsf{right}}} H^{r_j}$ with a new signature of knowledge (on a new message) separately. Extracting the witness from the SoK would give s, r such that $C_j = G^s H^r$.

As P is chosen as a some known power e of G the simulator now knows a, b such that $G^a P^{t_j^{\mathsf{right}}} = G^b$. As P was similarly chosen as a known power e of G we now have $G^{a + e \cdot t_j^{\mathsf{right}}} = G^b$. As previously these known powers may be used to find roots of elements in $\mathbb{G}$ provided whenever $a + e \cdot t_j^{\mathsf{right}} \neq b$. An adversary choosing s, r such that $a + e \cdot t_j^{\mathsf{right}} = b$ with non-negligible probability could be used distinguish between chosen values of t_j^{right}, directly contradicting the statistical indistinguishability to a distribution independent of t_j^{right} in property (3) of Lemma 2.

(iii) Now, we move on to consider the case where the adversary *did not* reuse a commitment with a new signature of knowledge. Verification requires $H^{r^*} \prod_{z=1}^{n^*} D_z = \prod_{j=1}^{\mu^*} C_j$. The commitments C_j were either produced earlier by the challenger (such that $C_j = G^{s_j^{\mathsf{right}}} P^{t_j^{\mathsf{right}}} H^{r_j}$ for known s_j^{right}, t_j^{right} and r_j) or constructed by the adversary with an accompanying signature of knowledge π (such that $s_j^{\mathsf{right}}, r_j$ satisfying $C_j = G^{s_j^{\mathsf{right}}} H^{r_j}$ can be extracted; this may be regarded as a special case where $t_j^{\mathsf{right}} = 0$).
Since we did not abort in hybrid (2), and the forgery may not contain any corrupt signer public keys, we may be certain all $D_z \in L^*$ were signed by the reduction. Therefore the reduction knows s_z^{left}, t_z^{left} such that $D_z = G^{s_z^{\mathsf{left}}} P^{t_z^{\mathsf{left}}}$ for each $z \in [n^*]$. The verification equation ensures that the reduction knows two powers a, b such that $G^a = G^b$. The left power a is dependent on $\sum_{j \in [n^*]} t_j^{\mathsf{left}}$, while the right power b depends on $\sum_{j \in [\mu^*]} t_j^{\mathsf{right}}$. The reduction may use the adversary to break the RSA assumption by the approach described previously if $a \neq b$.
For any $L_i^h \subset L^*$ at least one element of R_i^h must not be in R^*. We denote the included subset as $R' \subset R_i^h$. The adversary may not ensure $a = b$ with non-negligible probability as this would contradict the statistical indistinguishability as described in property (2) of Lemma 1, to a distribution where any subset of $P^{t_i^{\mathsf{left}}}$ is independent of $\{P^{t_{i,j}^{\mathsf{right}}}\}_{i \in [n], j \in [\mu_i]}$.
The case where $R_i^h \subset R^*$ and $L_i^h \not\subset L^*$ is largely analogous, but using property (1). If both L_i^h and R_i^h are partially included the difference in powers of P will also be uniform and independent following both properties (1) and (2).

C A SNARK-Based UOS Scheme

In this section, we sketch a construction of union-only signatures based on SNARKs. Though it is more computationally expensive, this construction has the advantage of constant-size signatures (even after merges have taken place).

We will use two main building blocks: an unforgeable signature scheme $\mathsf{Sig} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ (described in Section A), and a zero-knowledge $SNARK = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ (described below in Section C.1).

C.1 Preliminaries

A *zero knowledge succinct non-interactive argument of knowledge (zk-SNARK)* allows a prover to create a short proof that she knows a witness w for a statement ϕ such that $\mathcal{R}(\phi, w) = \texttt{accept}$ for some relation $\mathcal{R}$. The proof should be short (i.e. succinct), and should not reveal anything about the witness (zero knowledge). A zk-SNARK scheme has the following syntax:

$\mathsf{Setup}(1^\lambda, \mathcal{R}) \to \mathsf{p}$**:** Takes the security parameter λ and a relation $\mathcal{R}$. Outputs public parameters p which are implicitly given to all algorithms.

$\mathsf{Prove}(\phi, w) \to \sigma$**:** Takes a statement ϕ and a witness w such that $\mathcal{R}(\phi, w) = \mathtt{accept}$. Outputs a proof π.

$\mathsf{Verify}(\phi, \pi) \to \{\mathtt{accept}, \mathtt{reject}\}$**:** Takes a statement ϕ and a proof π and either accepts or rejects the proof.

Informally, we require from our zk-SNARK scheme the properties of an SoK (Soundness, Zero knowledge, Simulation extractability as in Section A) and additionally Succinctnsess: the size of the proof should be sublinear in the sizes of the statement and witness.

We refer to [24] for formal definitions of zk-SNARKs.

C.2 Construction

A UOS on the set M consists either of a signature from the underlying signature scheme Sig, or an argument π proving knowledge of signatures on two sets M_1 and M_2 under sets of public keys PK_1 and PK_2 such that $M = M_1 \cup M_2$ and $PK = PK_1 \cup PK_2$. The statement for the SNARK is $\phi = (M, PK)$ — consisting of a set M of messages and a set PK of public keys for the signature scheme — while the witness is $w = \{M_i, PK_i, \sigma_i\}_{i\in[n]}$ — consisting of a set of (message set, public key set, signature) tuples. The relation $\mathcal{R}$ is:

$\mathcal{R}(\phi, w) =$

$$\left\{\begin{matrix} M = M_1 \cup \cdots \cup M_n \\ \wedge PK = PK_1 \cup \cdots \cup PK_n \\ \wedge \forall i \in [n], \\ \Big(SNARK.\mathsf{Verify}(\phi_i := (M_i, PK_i), \sigma_i) = \mathtt{accept} \\ \vee (PK_i = \{\mathsf{pk}_i\} \wedge \mathsf{Sig.Verify}(\sigma_i, M_i, \mathsf{pk}_i) = \mathtt{accept})\Big) \end{matrix}\right\}$$

Finally, our UOS scheme works as follows:

$\mathsf{Setup}(1^\lambda)$ returns $\mathtt{crs} \leftarrow SNARK.\mathsf{Setup}(1^\lambda, \mathcal{R})$.

$\mathsf{KeyGen}(1^\lambda)$ returns $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Sig.KeyGen}(1^\lambda)$.

$\mathsf{Sign}(\mathsf{sk}, M)$ returns $\sigma \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}, M)$.

$\mathsf{Merge}(\{\sigma_i, M_i, PK_i\}_{i\in[n]})$ returns $\sigma \leftarrow SNARK.\mathsf{Prove}(\mathtt{crs}, \phi := (\cup_{i\in[n]} M_i, \cup_{i\in[n]} PK_i), w := \{M_i, PK_i, \sigma_i\}_{i\in[n]})$.

$\mathsf{Verify}(\sigma, M, PK)$ returns either $\mathsf{Sig.Verify}(\sigma, M, \mathsf{pk})$ (where $PK = \{\mathsf{pk}\}$), or $SNARK.\mathsf{Verify}(\phi := (M, PK), \sigma)$, depending on whether the signature in question is fresh or merged.

Note that because of the succinctness of the zk-SNARK scheme, our signatures remain the same size even after multiple merge operations.