Digital Signatures for Modifiable Collections

Serge Abiteboul
INRIA & LRI-Univ. of Paris-Sud
serge.abiteboul@inria.fr

Bogdan Cautis
INRIA & LRI-Univ. of Paris-Sud
bogdan.cautis@inria.fr

Amos Fiat
Tel-Aviv Univ.
fiat@tau.ac.il

Tova Milo
Tel-Aviv Univ.
milo@cs.tau.ac.il

Abstract

The common assumption about digital signatures is that they disallow any kind of modification on signed data. However, a more flexible approach is often needed and has been advocated lately, one in which some restricted modifications may still occur, without invalidating the data. This is made possible by offering signatures which are homomorphic *with respect to some operation on the message domain. Starting from the signature(s) of some data instance(s), computed by the data owner, anybody else can derive the signature corresponding to a new data instance, if obtained only via some accepted operation from the previous one(s). More, updated signatures should be* indistinguishable *from the ones computed by the data owner and this updating step should be applicable as many times as needed. This paper deals with the signing of* insert-only collections*, in which element insertions are accepted but no removals should occur. Newly inserted elements do not have to be signed or known by the initial signer. We propose two techniques: one which transposes the* insert-only *problem into a* delete-only *one (which is already solved), and another technique based on zero-knowledge proofs. We also give performance measures and discuss applications.*

1 Introduction

It is conventional wisdom that a main drawback of information exchange (e.g., on the Web or in peer-to-peer) is the lack of security. A partial answer to the problem is provided by secure protocols such as *SSL/TLS* [9, 7] that in particular support the encryption and authentication of information that is transferred. However, in many information exchange scenarios, a *much finer control* of security is required, where information may be partially modified on the way.

Consider a patient medical record that moves from one party to another (e.g., nurses, doctors, clerks). First, it may be critical for someone who receives the patient record to understand the provenance of data. Such provenance guarantees can indeed be provided by standard digital signatures. However, someone participating in the exchange of this record may have to modify it, e.g., add some annotation; or she may have to filter out some classified information. Still, even if such modifications can fit in the picture, a source of information may typically want to control how its data is modified along the way. For instance, a doctor may not allow other parties to remove her diagnosis. Equally, one may want exchange parties to be able to verify data validity without being able to detect the modifications that may have happened. For instance, some sensitive data concerning a particular medical test may have been removed from a patient record. The disclosure of the fact that something was deleted may in itself be an information leak.

Such scenarios combining security and privacy issues could be easily supported in the presence of some trusted authority which controls and authenticates everything that goes on. The obvious challenge is to support them in a general, unrestricted case, where no such suppositions are made. For that, we should rely uniquely on digital signature techniques which, as usual, provide integrity and authenticity guarantees, but achieve these guarantees in the presence of some accepted “legal” modifications.

Our contribution We present cryptographic techniques for signing modifiable collections, enforcing authorship while enabling us to verify that only valid modifications may have been performed. Collection signatures will be homomorphic with respect to two basic operations, element insertion (i.e., set to superset transformation) and element removal (i.e., set to subset transformation). In this context, by authenticity we mean that a signer’s identity is associated with the collection (but not necessarily to individual elements), while integrity will refer to the collection as a whole, guaranteeing only that a current collection is a superset (respectively a subset) of an initially signed one. It is rather straightforward to handle delete-only collections, using a known signature scheme. We present it here not only to give a complete picture, but also because this technique becomes useful when dealing with insert-only collections. Handling insertion is more challenging and represents the core of this paper. We present a novel, conceptually simple, yet effective technique for which communication costs are very low. Our measures prove that the technique is usable in practice, even for large sets. We also discuss a zero-knowledge based protocol which can enforce insert-only or delete-only collections in a restricted sense. We present two potential applications which can greatly benefit from insert-only collections and for which no satisfactory solutions were previously known.

Related work The usefulness of homomorphic signatures schemes has been firstly advocated by Rivest in [19]. In [14], Micali et al present a signature scheme for graph edges which is homomorphic w.r.t. transitive closure: given the signatures for two edges (u, v) and (v, w), one can derive the signature for the edge (u, w). Chari et al ([6]) introduce a signature schemes for tree nodes, such that given the signature of a child node one can derive the signature of its parent. Using this scheme, they obtain an efficient solution for the problem of signing aggregated routes in hierarchical routing protocols. Johnson et al ([13]) propose a set signing scheme which is homomorphic with respect to set difference (given the signature for a set, one can derive the signature of any subset) and union (give the signatures of two sets, one can derive the signature of their union). Also, they present a scheme which deals with message redaction (given a signed string, one can derive a signature for any redacted version of it). A signature scheme homomorphic with respect to bitwise AND and XOR was proposed in [20]. The need for such techniques for secure exchange of modifiable information was also raised in the database community. In [16, 15], Miklau et al provide a formal model for integrity in data exchange and identify the challenge we address here (see also the references therein).

The paper is organized as follows. The next section introduces notions used in the rest of the paper. Section 3 briefly discusses delete-only collections and Section 4 presents a first solution for insert-only collections, based on a reduction to the delete-only case. Section 5 discusses an alternative solution based on zero-knowledge proofs. We consider applications in Section 6 and Section 7 concludes.

2 Preliminaries

This section introduces the notions used in the rest of this paper, and it may be skipped at a first reading. To ease the reading, we present simplified versions of these rather standard definitions and refer the interested reader to [11] for more rigorous ones.

Negligible functions We say that a function $f(l)$ is *negligible in l* if it decreases to zero faster than any polynomial in $1/l$, for l sufficiently large.

Efficient algorithms An *efficient (or feasible)* algorithm is one that is polynomially bounded in the size of its input.

One-way functions A function f is *one-way* if any efficient algorithm succeeds in inverting f with negligible probability in the length of the input. A one-way function is (t, ϵ)-*secure* if, for any given image y, an attacker limited to running in time t, succeeds in finding $f^{-1}(y)$ with probability at most ϵ.

Public-key signature schemes A public-key signature scheme is a pair of efficient algorithms $(S,\ V)$, with $S : \mathcal{D} \times SK \rightarrow \mathcal{S}$ being a (probabilistic) one-way signing algorithm and $V : \mathcal{D} \times \mathcal{S} \times PK \rightarrow \{true, false\}$ being a signature verification algorithm. $\mathcal{D}$ represents the domain of signable values (not necessarily distinct from $\mathcal{S}$), by SK we denote the domain of secret keys (know only by the signer), while PK represents their matching public keys[1] (which can be know by any verifier). We will note $S_{sk}(..)$ (resp. $V_{pk}(..)$) for $S(.., sk)$ (resp. $V(.., pk)$). As a soundness requirement, for a matching pair of public-private keys (pk, sk) and any $s = S_{sk}(d)$, $V_{pk}(d, s)$ should always output $\{true\}$. We note that since the signing algorithm may be probabilistic (making random choices during its execution), multiple signatures may be valid for one particular domain value.

Forgeries Given a pair $(d, s) \in \mathcal{D} \times \mathcal{S}$, with s obtained without knowledge of the secret key (in particular, s is not the result of computing $S_{sk}(d)$), we say that the pair (d, s) represents a *forgery* if $V_{pk}(d, s)$ outputs $\{true\}$.

Cryptographic/collision-free hashing In what is known as the hash-and-sign paradigm, before the signing step, a cryptographic hash function is used to transpose elements from a general message domain (say $\{0, 1\}^*$) into some specific domain required by the signing step. A family of collision-free hash functions $H_k : \{0, 1\}^* \rightarrow \mathcal{D}$ is an easy to sample (keyed) family of efficient functions, such that it is unfeasible (for keys k sufficiently large) to find two distinct pre-images mapping to the same output.

Random oracles are a useful abstraction for cryptographic hash functions, often used when analyzing schemes where strong randomness assumptions are required (more on this paradigm in [3]). A random oracle h (or an *ideal hash function*) outputs, for a given new query x, a value $h(x)$ chosen uniformly at random from the oracle's output domain, independent of the output for any other previous input $x' \neq x$. If a query x has been previously asked, the oracle responds with the same value it gave before. Unless stated otherwise, hash functions will be considered ideal.

Exact security We consider the *exact security* of a signing scheme by quantifying the probability of finding a forgery with limited time and chosen-message queries [4]. We say that a public-key signature scheme is $(t, q_{sig}, q_{hash}, \epsilon)$-*secure* if an adversary, provided the public key, running in time t, asking for at most q_{sig} chosen message-signature pairs and invoking at most q_{hash} times an ideal hashing function, succeeds in finding a forgery (i.e., a valid signa-

[1]We ignore here the key generation part of a signing scheme and assume that the domain of keys is sufficiently large.

ture for whatever new message) with probability at most ϵ. This represents the notion of *existential forgery under an adaptive chosen message attack*.

Homomorphic signatures We say that a signature scheme (S, V) defined as above is homomorphic with respect to some binary operation $\odot : \mathcal{D}^2 \to \mathcal{D}$, if there exists an efficient signature update algorithm $UP : \mathcal{D}^2 \times \mathcal{S}^2 \times PK \to \mathcal{S}$ (or $UP : \mathcal{D}^2 \times \mathcal{S} \times PK \to \mathcal{S}$) such that

$$UP_{pk}(d_1, s_1, d_2, [s_2]) = S_{sk}(\odot(d_1, d_2))$$

for any pair of matching public-private keys (pk, sk) and pairs (d_i, s_i) such that $V_{pk}(d_i, s_i)$ outputs $\{true\}$. An important requirement is that signatures obtained via the (secret) signing algorithm and the ones obtained via the (public) updating algorithm should be indistinguishable (as defined below). For such signature schemes, we need to adapt the above notion of security with respect to forgery, since new signatures can be efficiently computed by anyone knowing the public key, via the signature updating procedure. A forgery will now refer to a message which is not obtainable via $\odot$ from already signed messages.

Indistinguishability We consider the privacy of signature updates by relating the learning advantage of an adversary to the notion of indistinguishability of distributions. Intuitively, two probability distributions $\mathcal{D}_1$, $\mathcal{D}_2$ over the same probability space $S \in \{0,1\}^l$ are *computationally indistinguishable* if no efficient algorithm can tell them apart. Formally, for any family of efficient boolean algorithms A_l,

$$\epsilon(l) = |Pr[A_l(x) = 1 | x \sim \mathcal{D}_1] - Pr[A_l(x) = 1 | x \sim \mathcal{D}_2]|$$

is negligible in l, where by $x \sim \mathcal{D}_i$ we denote that x is distributed according to $\mathcal{D}_i$. We say that a homomorphic signature scheme achieves *perfect* (resp. *computational*) *privacy* if, for any matching pair of keys (pk, sk), the distributions induced by $S_{sk}(..)$ and $UP_{pk}(..)$ are equivalent (resp. computationally indistinguishable).

Pseudo-randomness We say that a distribution D on $\{0,1\}^n$ is *pseudo-random* if it is computationally indistinguishable from the uniform distribution U_n. An efficient function $g : \{0,1\}^n \to \{0,1\}^m$, $m > n$, is called a *pseudo-random generator* if $g(U_n)$ is pseudo-random.

3 Delete-only collections

In this section, we recall standard results for delete-only collections. Consider a delete-only collection U (we will use the terms *collection* and *set* interchangeably in the rest of the paper), with initial value $\{x_1, \ldots, x_n\}$, and let (S, V) be some public-key signature scheme. A naive solution is to sign every element in the original collection and use the set of signatures as the collection signature. Anyone who wants to remove elements needs just to remove their corresponding signatures. Verification amounts to checking that all the current elements in the set have matching signatures.

A more efficient signature scheme can be achieved using the set-homomorphic signing scheme of [13]. Starting from an RSA-like scheme, we obtain a public-key set-signing scheme (S, V) that is homomorphic with respect to both set *difference* and *union*. More specifically, it provides two (public) updating functions $\widetilde{UP}$ and $\widehat{UP}$, such that:

$$(1)\ \forall U \ and\ U' \subseteq U,\quad \widetilde{UP}_{pk}(U, S_{sk}(U), U') = S_{sk}(U')$$

$$(2)\ \forall U_1, U_2,\ \ \widehat{UP}_{pk}(U_1, S_{sk}(U_1), U_2, S_{sk}(U_2)) = S_{sk}(U_1 \cup U_2)$$

Now consider that a set $U = \{a_1, ..., a_n\}$ and the signature $s = S_{sk}(U)$ (for some secret key sk) are given. Someone who wants to remove elements, for instance update U to the subset U', will be able to sign U' by using (1). Note that, by using (2), we can also obtain signatures of unions of signed sets. The mathematical details may be found in [13].

In this paper, we will call the above signature scheme *Delete Homomorphic* (*DH* in short). We say that an adversary produces an *insert forgery* if, after obtaining signatures for some sets $U_1, \ldots, U_n$, signed by the same secret key, he can produce a valid signature for some set $U \not\subseteq \bigcup_i U_i$. We have the following result:

Theorem 1 *[13] Assuming that RSA is a (t, ϵ_r)-secure one-way function, the DH signature scheme is (t', q_s, q_h, ϵ)-secure against insert forgery, where $\epsilon \approx q_h \epsilon_r \lg n + q_h^2 \lg n/n$, $t \approx t'$, $q_s \le q_h$ and n is an RSA modulus.*

The signatures obtained by the homomorphic property are history independent and this scheme clearly achieves *perfect privacy*, since an updated signature is exactly the one that the signing procedure would yield. Also, the signature size does not depend on the number of elements in the collection (see also Section 4.1 on experiments).

Observation Although it may seem that equation (2) affects our aim for delete-only collections (since one can also perform insertions if they refer to elements from another signed set), homomorphism w.r.t. union can have little impact if distinct (fresh) public-private key pairs are used each time a delete-only collection is signed.

4 Insert-only collections

This section introduces a novel technique for signing insert-only collections. We first give the intuition on how the technique can be built using the DH scheme, by relying on a “write-only ” memory. We show how to enforce naively a “write-only” memory. Then, we improve the technique by using an encoding that is provably optimal. We then detail the signing of insert-only collections.

Naive write-only Consider a memory M of N bits, all initially set to undefined (neither 0 nor 1). To make M write-only, we represent it as an array A_M of $2N$ bits as follows. An undefined bit position is represented by 00; the value 1 by 01 and the value 0 is encoded by 10. To enforce a write-only behavior of this memory, we sign the 0s in A_M using the DH scheme. Then, writing in M amounts to flipping bits from 0 to 1 (transforming memory positions of A_M from undefined to encodings of 0 or 1). This operation, seen as a removal of 0s, can be supported and made irreversible by the DH scheme; once one writes a 1 in place of a 0 it cannot be changed back (unless DH is broken), whereas 0 to 1 is allowed. Note that this bit-flipping technique presents some strong analogy with "write-once" disk technology. These are disks where a 0 may be turned into a 1 (by piercing a hole) and the operation is irreversible.

Observe also that this provides a means to enforce insert-only collections. Such a collection would have an associated memory (bit vector) in which memory blocks record elements of some fixed length (say from $\{0,1\}^k$), such that once an element is written into a block it should be unfeasible to write it off. We will come back to this in more detail further on.

Optimal write-only Observe that for now a memory position used for recording a k-bit value needs $2k$ bits. This encoding is hardly optimal, even though it satisfies the crucial requirement $(\dagger)$ that passing from one encoded value to a different one requires introducing at least one 0 (hence breaking the DH scheme). We can do significantly better by representing a number from 1 to 2^k by v bits where v is the minimal integer s.t. $\binom{v}{\lfloor v/2\rfloor} \geq 2^k$.

An element encoding (on v bits) is given by $\lfloor v/2 \rfloor$ bits set on 1 and the remaining bits set on 0. For a given number $x \in \{1,\ldots,2^k\}$, its encoding is the one pointing to the x'th combination (in lexicographic order) of $\lfloor v/2 \rfloor$ out of v. So $(\dagger)$ is preserved since (i) no two values have the same encoding; and (ii) it is not possible to move from one encoding to another because at least one 0 would be (re)introduced. We call the above the *half-half encoding*. Given a value x, its mapping into a v-bit representation can be constructed in linear time. Via Sperner's theorem [5], we can prove the following[2]:

Proposition 2 *Under the* $(\dagger)$ *requirement, the* half-half encoding *is* optimal *and yields* $k + \mathrm{O}(\lg k)$ *long encodings.*

Overview of the signing technique We can now describe the signing technique for insert-only collections. For now, we assume that the size of the insert-only collection is bounded by some limit l (we will see further how we can effectively overcome this restriction). Consider given a family of hash functions H_k mapping elements into $\{0,1\}^k$ and a function h chosen uniformly at random from this family. Let v_k be the length of the half-half encoding for k-bit values. Consider starting with an empty insert-only collection and an associated, empty (all bits are set to 0), "write-only" memory of total bit length $l \cdot v_k$. This array is partitioned into l contiguous blocks of length v_k. As above, to enforce the "write-only" behavior of the collection, the set of 0s is

[2]For space reasons all formal proofs are omitted and can be found in[1].

```
     Input: $U = \{e_1,\ldots,e_{l'}\}$, parameters $(l,k,v_k)$, $l' \leq l$
1.   construct the bit vector of $v_k \cdot l$ bits, initialized to $0$
2.   choose at random a key pair $(pk, sk)$ for the DH scheme
3.   choose at random the hash function $\{h\}$
4.   for each $e_i$ in turn
     4.1. choose at random an available block $b_i$ between $1$ and $n$
     4.2. "write" the hash value $h(e_i)$ in the selected block
5.   let $P_0$ be the set of remaining $0$ positions in the bit vector
6.   sign with the DH scheme $P_0$ using $sk$, obtain $\sigma$
7.   output as a public key for the IH scheme the tuple
     $(l, k, v_k, h, pk)$
```

```
     Input: $U = \{e_1,\ldots,e_{l'}\}$, $(l,k,v_k,h,pk)$,
     a DH signature $\sigma$ and $U' = \{e'_1,\ldots,e'_{l''}\}$, with $l' + l'' \leq l$
1.   reconstruct the bit vector and obtain $P_0$
2.   verify $\sigma$ for $P_0$ using the DH public key $pk$
3.   for each $e'_j$ in turn
     3.1. choose at random an available block $b_j$ between $1$ and $n$
     3.2. "write" the hash value $h(e'_j)$ in the selected block
4.   let $P'_0 \subset P_0$ be the set of remaining 0 positions
5.   compute the DH signature for $P'_0$ by updating $\sigma$ to $\sigma'$
```

Figure 1. Initial signing and updating

signed as a delete-only collection. Then, for a new element e to be inserted in the collection, we choose uniformly at random an available bit block and write into it the hash value $h(e)$ (and update the memory signature accordingly).

Note that, by symmetry, we obtain signatures that are homomorphic for both element insertion and set intersection.

The signing and updating procedures are summarized in Figure 1. By analogy with the DH scheme, we call the above scheme *Insert Homomorphic* (IH). The main compution cost is the computation of signatures, which is linear in the number of elements. The communication overhead is just a fixed part consisting of σ, h and the memory parameters. As we will see this scheme turns out to be usable in practice even for large sets.

Security analysis The security of this scheme is considered with respect to a different kind of forgery, namely *delete forgery*. By symmetry with insert forgery, we say that an adversary finds a delete forgery when, given signatures for sets $U_1,\ldots,U_n$, for the same signing key, he computes a valid signature for a set U s.t. $\bigcap_i U_i \nsubseteq U$.

In our analysis, we use the following game played between a *Signer* $(\mathcal{S})$ and an *Attacker* $(\mathcal{A})$. For an arbitrary maximal set size l and security level K:

1. $\mathcal{S}$ initializes the game by:
   - generating a DH public-private key pair (pk, sk);

- choosing a random (secret) hash function h from a sufficiently large family of hash functions;
- setting the bit array parameters (k, v_k) based on n and K;
- publishing the array parameters and pk;

2. $\mathcal{A}$ asks for the signature of a first set U_1. $\mathcal{S}$ responds by sending the signature of U_1 and by disclosing h (practically, this can be achieved by viewing h as part of the signing public key);
3. $\mathcal{A}$ can now either ask for signatures of other sets U_i or evaluate $h(u_j)$, for any sets and elements of his choice;
4. $\mathcal{A}$ wins if he can output a signature for a set U_f of size at most n, such that $\bigcap_i U_i \nsubseteq U_f$.

This adversary game is almost identical to the standard one of producing existential forgery under adaptive chosen message attacks. The notable difference is that the hash function h are hidden until a first signature is published. It may seem that, by imposing this, we affect the chosen-message attack philosophy (we basically avoid the birthday paradox and prevent the adversary from taking advantage of *any* hash collisions). However, this is a perfectly reasonable assumption here since the space for forgery will firstly be limited by the attacker's initial signature query. Obviously, this is not the case in other (classic or homomorphic) signing schemes where the signing queries that are being asked do not limit in any way the subsequent forging options of an attacker. There, it makes no difference if the attacker has already access to the hash oracle (before asking for chosen message signatures). It would not affect in any way their proven security guarantees or the attacker chances in producing a forgery. We also stress that the this condition affects our analysis only in a quantitative way. When this is not the case, it just becomes slightly easier to find a forgery, resulting in an increased (but still feasible) signing complexity.

The IH scheme is (t, q_s, q_h, ϵ)*-secure* for deletion forgery if an adversary running in time t, asking for at most q_{sig} chosen set signatures and invoking at most q_h times an ideal hash functions, succeeds in the game above.

Theorem 3 *Assuming that RSA is a (t, ϵ_r)-secure one-way function, the IH signature scheme of parameters (l, k, v_k) is (t', q_s, q_h, ϵ)-secure against delete forgery, where $t \approx t'$, $\epsilon \approx \beta \cdot \epsilon_r \cdot \lg n + \beta^2 \cdot \lg n/n + l \cdot q_h/2^k$, $\beta = l \cdot v_k$, $q_s \le q_h$ and n is an RSA modulus.*

Privacy analysis We next consider the privacy guarantees of updated signatures in this scheme. Indistinguishability is not immediate anymore, since the signing and updating algorithms are now probabilistic (in the choice of array blocks). Two distinct ways (by sequences of insertions) of computing a set signature will very likely yield different corresponding bit vectors and thus signatures. We need to make sure that the various signatures cannot be distinguished by an adversary. We have the following result.

Proposition 4 *Assuming that element representations and IDs are pseudo-randomly generated, the IH scheme achieves computational privacy.*

Overcoming the size limitation Observe that for now a size limit l is fixed in advance, and the bit vector used in the signing procedure is dimensioned accordingly. What happens if a modifier wants to add more than l elements? Since a public key can be part of a collection, just as any other element, he can take the following steps:

- generate a new public-private key pair (pk', sk') for the DH scheme.
- introduce pk' in the collection and its current vector.
- create a new bit vector initially signed by the newly obtained secret key (sk') (matching the public one introduced in the collection and its existing bit vector).
- gather the new vector and its signature to the existing one(s) and their signature(s).
- insert whatever element needs to be inserted in the collection using the new bit vector.

This means that by a priori choosing the parameters (l, k, v_k) we are now merely setting a basic bit vector unit, which does not restrict the overall size of the insert-only collection (which can rely on as many such units as needed). We call this extension *IHC* (IH with *chaining*). A minor drawback is that we may now disclose a partial order on the insertions. We can state the following corollary.

Corollary 5 *Under the same assumptions of Theorem 1, if the IH scheme of parameters (l, k, v_k) is (t, q_s, q_h, ϵ)-secure against delete forgery, then the IHC scheme is $(t', q_s, q_h, \epsilon')$-secure, for $\epsilon \approx \epsilon'$ and $t \approx t'$.*

4.1 Experimental evaluation

We analyze now the costs of signing delete-only and insert-only collections. We used a straightforward C implementation for the two schemes (DH and IH), using the GMP library [10], and tested them on a low-end machine (2.4GHz, 1Gb-RAM, Pentium IV). For DH signatures we used 1024-bit RSA moduli and 200-bit exponents. We first consider the DH scheme. We looked at the overhead of initiating a signature for a given set, the overhead of verification (depending on how many elements are in the set) and the overhead of removing some existing elements (see Table 1). These costs do not depend on the number of elements in the collection. We used a standard cryptographic hash function that takes about 1μsec per call.

Turning now to the IH scheme, we need first to set the minimal array parameter v_k. In what follows we want to achieve a forgery probability of as low as 2^{-60} for $q_h \approx 2^{60}$ (usual recommended values). For instance, for hash values of $k = 128$ bits, we obtain that $v_k = 132$. The number of array bits per potential collection element (v_k) gives

Initialization time per element	120 μsecs
Verification time per (existing) element	2.4 msecs
Updating time per removed element	2.4 msecs

Table 1. DH scheme: computation costs

the computation overhead per element. From Table 1, we can derive the overhead of initiating a signature, the one of verification (depending on how many elements can still be inserted), and the overhead of adding new elements to the signed collection (depending on the number of elements to be inserted). See the results of Table 2.

Initialization time per (potential) element	15 msecs
Verification time per (potential) element	0.31 secs
Updating time per inserted element	0.31 secs

Table 2. IH scheme: computation costs

Let us consider now the size of the signature, expressed also in bits per potential collection element. Since the DH scheme signature size is fixed, regardless of the collection size, this relative cost decreases with the number of elements. We stress that the size of the bit array does not affect in any way communication.

Initialization time per (potential) element	1 msec
Verification time per (potential) element	19 msecs
Updating time per inserted element	19 msecs

Table 3. Parallelized IH: computation costs

Further optimization Observe that although much higher than for the DH scheme, time remains reasonable and in particular usable in practice. Furthermore, the IH scheme can be easily parallelized. For instance, we can sign a set of maximal size 16384 by using 16 bit vectors, each of them containing at most 1024 elements (see Table 3). Of course, this results in a larger overall signature (one per bit vectior), leading to a computation/communication trade-off.

5 A zero-knowledge protocol

In the case of large collections, the IH scheme may require an amount of time considered too important for some applications. That is why in the following we discuss a zero-knowledge based protocol that may be computationally lighter, but presents other drawbacks.

- First, it can be employed only in 1-step updating setting, involving 3-parties: we have a *Signer* who signs an initial collection, a *Modifier* who adds some elements and a *Verifier* who verifies that only insertions occurred, without being able to distinguish the *Modifier*'s added elements from the *Signer*'s initial ones.
- Second, the basic protocol is *interactive* and we deviate significantly from the notion of signature scheme as used until now. The signing and verifying steps will be interactive processes.
- Third, communication becomes the main bottleneck.
- And fourth, in its initial form, the protocol discloses the number of elements introduced by the *Modifier*.

The protocol in itself may not be entirely novel, similar approaches have been used before in the zero-knowledge area (see for example [18]), but the problem it solves and setting are new. We start by a brief introduction on 2-party interactive and zero-knowledge proofs. For space reasons we omit formal definitions and refer the interested reader to [11].

Let us assume two machines that can interact and have a common input: a prover (P) and a verifier (V) (which is polytime bounded). The interacting machines run on their common input and at the end of the run the verifier decides to *accept* or *reject* it. We consider that the prover's goal is to convince the verifier into accepting the input. The pair (P,V) is called an *interactive proof system* for a language L if there is a non-negligible gap between the acceptance probability of (P, V) (considered for the prover P) on valid inputs ($x \in L$) and the one of any pair (P^*, V)(for any prover P^*) on invalid inputs ($x \notin L$).

Loosely speaking, a zero-knowledge proof of a statement is an interactive proof that doesn't disclose anything besides the validity of the statement. We can see that this fits nicely our aim here, since we want the *Modifier* to prove to the *Verifier* that he produced a superset of what the *Signer* gave him, without actually disclosing the *Signer*'s elements. Intuitively, an interactive proof system (P, V) is (*computational) zero-knowledge* if for any verifier V^*, there exists an efficient simulator (Sim^*) that, without having access to P, is able to simulate V^*'s interaction with P. This means that V^* doesn't learn anything besides the validity of the prover's claim, since the same output could have been obtained without interacting with P. For that, the two ensembles below must be computationally indistinguishable.

- $\{(P, V^*)(x)\}_{x \in L}$ (the output of V^* after interacting with P on input x).
- $\{Sim^*(x)\}_{x \in L}$ (the output of Sim^* on input x).

An auxiliary concept on which the protocol relies is the notion of *commitment* [11]. A (multi-bit) commitment scheme can be seen as a two-phase protocol between a *Sender* and a *Receiver*. In the first phase (*commit*), starting from a bit string x and a public key pk, the Sender picks a random value $r_x \in \{0, 1\}^l$ (called the *proof* or *witness*), computes a commitment to x ($c_x = commit_{pk}(r_x, x)$) and sends it to the Receiver. In the second phase (*opening*), the Receiver demands the opening of the commitment, receiving the proof r_x from which he can (efficiently) learn x. We will say that the commitment scheme is *secure* if it satisfies the two following requirements:

- *Computationally binding*: the probability that a poly-time bounded Sender finds two distinct pairs (x, r_x), (y, r_y) s.t. $c_x = c_y$ is negligible in l.
- *Unconditionally hiding*: for any x, y, the distributions of $\{c_x\}_{r\in\{0,1\}^l}$ and $\{c_y\}_{r\in\{0,1\}^l}$ are statistically indistinguishable.

We first argue that enforcing insert-only collections is very similar to the following proof game (which we call *subset selection*) between the second party (*Modifier*) and the third one (*Verifier*), on a common input representing a set U: the *Modifier* picks a set of values U', claims that $U' \subseteq U$ and proves in zero-knowledge the claim.

With the (for now) artificial assumption that the *Modifier* chooses U' and then never changes it, the zero-knowledge proof of the above could be constructed as follows:

- *Modifier*: for each $x \in U'$, compute some fresh commitments c_x and c_{c_x}. Send (in any permuted order) the set of commitments $\{c_{c_x}\}$.
- *Modifier*: complete the set of commitments c_x with commitments for the elements in $U \setminus U'$. Let the entire set of element commitments be $\{c_y\}$. Send in any permuted order $\{c_y\}$, claiming they correspond to each $y \in U$.
- *Verifier*: make a random bit choice (0/1). If 1, ask for the opening of $\{c_y\}$. If 0, ask for the opening of the $\{c_{c_x}\}$ commitments.
- *Modifier*: sends the requested openings.
- *Verifier*: verifies the opened commitments and accepts if they are valid. More precisely, if he opens the $\{c_y\}$ commitments, he verifies they correspond to U's elements. Otherwise, he verifies that the $\{c_{c_x}\}$ commitments correspond to a subset of c_y.

If the *Modifier*'s claim is true, then the *Verifier* will always accept. However, if the claim is false, the overall probability of erroneous acceptance is less than $1/2$ plus the probability of finding some two openings for the same commitment. By sequentially repeating the protocol we can make the overall probability arbitrarily low.

Proposition 6 *Assuming that the commitment scheme is secure, the above protocol is a zero-knowledge interactive proof system for subset selection.*

Passing now to the 3-party setting, we need some minor modifications. The *Signer* will be the one choosing a set U' (the collection she sends to the *Modifier*), without knowing anything about U (besides supposing it will be a superset of U'). Basically, he will play the role of a trusted third party, since we can reasonably assume him to be honest (he doesn't take part in the *Modifier*'s cheating attempt, so U' is fixed, and he doesn't disclose her elements to the *Verifier*).

We can avoid interactions *Signer*↔*Verifier* or *Modifier* →*Signer* by assuming that the *Signer* is the one computing the $\{c_{c_x}\}$ and $\{c_x\}$ commitments. He signs the ones that he has to send to the *Verifier* (with any standard signing scheme) and sends them through the *Modifier*. We'll need two-way interaction only between the *Modifier* and the *Verifier*. The protocol's analysis is straightforward.

Unfortunately, this technique reveals the exact size of the original set. If hiding this size is important, one can use a "padding" strategy, even though the modified technique would still leak some bound on the size of the original set.

Initialization per existing element (*Signer*)	0.04 secs
Updating per existing element (*Modifier*)	0.04 secs
Updating per inserted element (*Modifier*)	0.02 secs
Verification per element (*Verifier*)	0.02 secs
Communication costs per existing element	10^5 bits
Communication costs per inserted element	$7 \cdot 10^4$ bits

Table 4. Cost estimates

For now the protocol is interactive. First, we can collapse the sequential repetitions into a single, parallelizing one. It is well known that by doing this we do not necessarily obtain a zero-knowledge proof system. However, since here it is mainly important not to leak anything about the history of changes (what was added), we believe that our approach maintains privacy even in the parallelized version. Second, we can obtain in a rather standard way a non interactive protocol by using the Fiat-Shamir technique [8]. Finally note that an analogous zero-knowledge protocol can be employed for delete-only sets.

Practical instantiations We estimated the costs of an instantiation of the above protocol, when using a commitment scheme that is both unconditionally hiding and practical (produces $O(l)$ long commitments), namely the one of Halevi-Micali [12]. Its commit phase involves two collision-free hashes and one universal hash. The verification phase is similar. If the collision-free hash function is a standard cryptographic one, we can consider the commit/opening steps as fairly inexpensive (say $0.1ms$). Without presenting all the details of the cost model, in order to obtain the same security guarantees as before, we get the cost estimates of Table 4. We ignored the time overhead induced by communication. Note that computation costs are lower than the ones of the IH scheme but higher than the ones of its parallelized version. However, communication complexity, even if linear in the security parameters and set size, becomes the dominant factor.

A radical instantiation We conclude with a remark that it may be possible to lower these communication costs if we view collision-free hashing as a satisfactory commitment scheme. More precisely, we can commit to the ensemble $\{x_i\}$ by computing $h_r(x_i) = h(x_i||r)$, where r would be the unique witness for each and every x_i. This would lower significantly communication costs. However, while such a commit scheme would be both computationally binding and hiding under the random oracle assumption, the protocol would no longer meet the definition of zero-knowledge and its not clear how its privacy guarantees could be stated.

6 Some applications

Application1: Access Control Access control in general, and update control in particular, have been studied extensively for both relational and semi-structured data, yet mostly in highly centralized settings. The presented work has been motivated by the design of completely decentralized access control (with no strings attached). In such an approach, a document “travels” (e.g., being published or exchanged), carrying access rights specifications and all the needed means to enforce these specifications. This problem raises important new issues, especially for semi-structured data and XML [21], which are considered de facto standards for data exchange. Considering updates, and ignoring the somewhat orthogonal issue of read access, it becomes immediately clear that only via digital signatures one could constrain the ways in which published data is modified. The techniques presented here could form an essential component of a full-fledged access control mechanism in a non-centralized setting. For instance, they complement read access techniques such as the ones of [17], which are typically based on the encryption of subtrees.

Application2: Secure Data Integration Data integration is one of the most studied topics in data management recently. Some security aspects of data integration have been considered in this context, such as secure data mining or OLAP [2]. In secure data integration it is commonly required that an integrating site provides integrity guarantees about the unified data coming from various sources, without disclosing the actual provenance of data. For instance (see Fig. 2), a warehouse may need to prove that its data is a subset (or superset) of the overall one coming from some sources, without disclosing any association between data items and their corresponding sources. Previous solutions to this problem were built by assuming collaborating sources and complex randomized protocols among them. By using digital signatures for insert-only and delete-only collections the above integration scenario can be fully supported, without imposing any additional requirement on the sources.

Details may be found in [1].

7 Conclusion

We considered in this paper signing techniques for modifiable collections, in particular delete-only and insert-only collections. For the latter, we proposed a novel and unrestricted approach which can be efficiently employed even for large collections. The security and privacy guarantees, as well as the optimality of the element encoding involved, were formally proven. We also discussed a zero-knowledge based protocol which works in a restricted setting but presents some advantages regarding computation costs. We believe that many data management problems can greatly benefit from the techniques discussed here and we are currently exploring potential applications of digital signatures for modifiable collections.

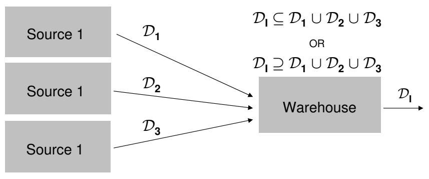
Source 1
Source 1
Source 1
$\mathcal{D}_1$
$\mathcal{D}_2$
$\mathcal{D}_3$
$\mathcal{D}_I \subseteq \mathcal{D}_1 \cup \mathcal{D}_2 \cup \mathcal{D}_3$
OR
$\mathcal{D}_I \supseteq \mathcal{D}_1 \cup \mathcal{D}_2 \cup \mathcal{D}_3$
Warehouse
$\mathcal{D}_I$

Figure 2. Secure data integration

References

[1] S. Abiteboul, B. Cautis, A. Fiat, and T. Milo. Secure exchange of modifiable collections. INRIA Technical Report. `ftp://ftp.inria.fr/INRIA/Projects/gemo/gemo/GemoReport-452.ps`.

[2] R. Agrawal. Data privacy. In *LNCS*, volume 3202, 2004.

[3] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *CCS*, 1993.

[4] M. Bellare and P. Rogaway. The exact security of digital signatures — how to sign with RSA and Rabin. In *LNCS*, 1996.

[5] B. Bollobás. Combinatorics: Set systems, hypergraphs, families of vectors, and combinatorial probability, 1986.

[6] S. Chari, T. Rabin, and R. Rivest. An efficient signature scheme for route aggregation. Technical report, 2002.

[7] T. Dierks and C. Allen. The TLS protocol. RFC 2246. 1999.

[8] U. Fiege, A. Fiat, and A. Shamir. Zero knowledge proofs of identity. In *STOC*, 1987.

[9] A. Frier, P. Karlton, and P. Kocher. The SSL 3.0 protocol. 1996.

[10] The GNU MP Bignum library. `http://www.swox.com/gmp/`.

[11] O. Goldreich. *Foundations of Cryptography*. Cambridge University Press, 2000.

[12] S. Halevi and S. Micali. Practical and provably-secure commitment schemes from collision-free hashing. *LNCS*, 1996.

[13] R. Johnson, D. Molnar, D. Song, and D. Wagner. Homomorphic signature schemes. In *CT-RSA*, 2002.

[14] S. Micali and R. Rivest. Transitive signature schemes. In *CT-RSA*, 2002.

[15] G. Miklau. Research problems in secure data exchange. U. of Washington, 2004.

[16] G. Miklau and D. Suciu. Modeling integrity in data exchange. In *SDM*, 2004.

[17] Gerome Miklau and Dan Suciu. Controlling access to published data using cryptography. In *VLDB*, 2003.

[18] R. Ostrovsky, C. Rackoff, and A. Smith. Efficient consistency proofs for generalized queries on a committed database. In *ICALP*, 2004.

[19] R. Rivest. Two signature schemes., 2000.

[20] T. Sander, A. Young, and M. Yung. Non-interactive cryptocomputing for NC 1. In *FOCS*, 1999.

[21] Extensible Markup Language 1.0 (2nd Edition). `http://www.w3.org/TR/REC-xml`.