Efficient signature schemes supporting redaction, pseudonymization, and data deidentification

Stuart Haber[*]
Hewlett-Packard
stuart.haber@hp.com

Yasuo Hatano[†]
Hitachi Ltd.
yasuo.hatano.bn@hitachi.com

Yoshinori Honda[†]
Hitachi Ltd.
yoshinori.honda.tb@hitachi.com

William Horne[*]
Hewlett-Packard
william.horne@hp.com

Kunihiko Miyazaki[†]
Hitachi Ltd.
kunihiko.miyazaki.zt@hitachi.com

Tomas Sander[*]
Hewlett-Packard
tomas.sander@hp.com

Satoru Tezoku[†]
Hitachi Ltd.
satoru.tezoku.mg@hitachi.com

Danfeng Yao[‡]
Rutgers University
danfeng@cs.rutgers.edu

ABSTRACT

In this paper we give a new signature algorithm that allows for controlled changes to the signed data. The change operations we study are removal of subdocuments (redaction), pseudonymization, and gradual deidentification of hierarchically structured data. These operations are applicable in a number of practically relevant application scenarios, including the release of previously classified government documents, privacy-aware management of audit-log data, and the release of tables of health records. When applied directly to redaction, our algorithm improves on [18] by reducing significantly the overhead of cryptographic information that has to be stored with the original data.

Categories and Subject Descriptors

H.2.7 [**Database Management**]: Database Administration—*Security, integrity, and protection*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Security, Algorithms

[*]HP Labs, 5 Vaughn Drive, Suite 301, Princeton, NJ 08540, USA

[†]Hitachi Ltd., Systems Development Laboratory 292, Yoshida-cho, Totsuka-ku, Yokohama-shi, Kanagawa-ken, 244-0817, Japan

[‡]Department of Computer Science, Rutgers University, 110 Frelinghuysen Road, Piscataway, NJ 08854-8019, USA. Work done while at HP Labs.

ASIACCS Tokyo, March 2008

Keywords

data integrity, redaction, data privacy, pseudonyms, digital signatures, audit logs

1. INTRODUCTION

Traditional digital signature schemes can be used to check that the authenticity and integrity of data are maintained, *i.e.* that the data has not been modified since it was signed. However, in some instances modification of the data is not only allowable, but desirable. In this paper we consider three important, practical types of modifications: redaction, pseudonymization and data deidentification.

Consider the Freedom of Information Act in the US and similar laws in other countries which enable citizens to request the disclosure of government documents. Often in such cases sensitive information, such as names of individuals, are blacked out or removed prior to release. This process of removal is called *redaction*.

In other applications, it is not appropriate to redact, but rather to replace data tokens with pseudonyms consistently throughout the document. Pseudonymization provides the end user with the ability to make certain kinds of structural correlations among the data without having access to sensitive information.

Finally, in other applications, such as those involving census data and healthcare records, it may be appropriate to gradually deidentify the data. For example, a birth date of an individual, such as *June 7, 1980* could be replaced by successive generalizations, such as *June 1980*, *1980*, *the 1980s*, etc. A city name could be replaced by its state or province, then by its country, etc. An income value could be replaced by nested salary ranges. Such successive generalization has been studied, for example, by Sweeney [23, 24]. Sweeney has further shown that health records commonly released to the public, and assumed to be anonymous, could be successfully linked to individuals in a number of cases, and therefore that deidentification is indeed necessary to preserve medical privacy.

In each of these cases, if the original data were signed, then traditional digital signatures could not be used to check the

integrity of the transformed data. While one could re-sign the modified data, establishing the integrity of the modified dataset, this would destroy any connection to the integrity of the original data. In this paper, we are interested in techniques that cryptographically link the integrity of the two datasets.

Integrity-preserving redaction, pseudonymization, and deidentification are useful tools for other types of data besides documents. For example, auditing and event logging are considered to be best practices in complying with corporate governance regulations such as SOX or privacy regulations such as HIPAA, to pick two examples from the United States. Such logs may contain sensitive information. In addition, audit logs are often voluminous, so in practice it is more practical to deal with a subset of the data than with the entire data set. Taking a subset of the data is equivalent to redacting all of the data that is not in the subset. As a result, redaction can be used to link the integrity of a subset to the integrity of the master data. We believe that a flexible redaction algorithm that can ensure the integrity of the redacted data will be a useful tool for supporting privacy-aware management of audit logs.

It is further desirable to have a scheme that can ensure that certain parts of the document *cannot* be further redacted, pseudonymized or deidentified. This is important because it serves as a safeguard to prevent certain essential meanings in a document from being changed. This additional requirement for the redaction problem is called "disclosure control," and was introduced by Miyazaki *et al.*, whose solution increased the size of the associated signature by a factor linear in the number of portentially redactable subdocuments [18].

In this paper we introduce an efficient algorithm to establish and verify the integrity of signed data subject to redaction, pseudonymization, and deidentification, which supports disclosure control in all three cases. Although for the first of the three problems, redaction, several solutions have been described in the literature, we are not aware of previous work on pseudonymized or deidentified data. Our algorithm significantly increases the efficiency of the previously known algorithm of [18] for redaction with disclosure control. We do this by adapting to our scenario the tree-building techniques of Johnson *et al.* in [12].

For redaction and pseudonymization the size of the cryptographic data overhead we have to store with the original document increases, in the worst case, by a logarithmic factor for each sequence of consecutive subdocuments that are redacted or marked as nonredactable.

Applying this algorithm directly to the deidentification problem would lead to an additional overhead which is linear in the length of the chains of generalizations. This can be significant in a healthcare example, where a typical table may contain millions of records and the chain of generalization for dates may be of length four. To handle this problem, we give a variant of the algorithm for the deidentification problem which avoids this blow-up, in case the successive generalizations follow a uniform pattern across the document or table, as is the case for dates, for example.

1.1 Organization of the paper

The paper is organized as follows. In Section 2 we describe related work on redaction and then a motivating example for some of the problems arising for deindentification; and finally we describe our model, list cryptographic preliminaries, and review a redactable signature algorithm from [12] that we use. In Section 3 we describe our new redaction algorithm with disclosure control, and in Section 4 we use this algorithm to solve the pseudonymization and deindentification problems efficiently. We give our security definitions and proof in the in Section 5.

2. BACKGROUND

2.1 Related work

The redaction problem has been independently considered by several groups of authors, under different names: as "digitally signed document sanitizing schemes" by Miyazaki *et al.* [17, 18], as "content extraction signatures" by Steinfeld *et al.* [21], and as "redactable signatures" by Johnson *et al.* [12].

Documents often contain information that should never be modified. A legal statement that a document was prepared in accordance with law would be an example of something a signer may wish to appear throughout the lifecycle of the document. Moreover, for many applications there are only limited parts of the document that would even make sense to redact. For example, in a structured form it might be acceptable to redact the data filled into the form, but there is no reason why the form itself should ever be redacted. Therefore, it is important to consider cryptographic redaction mechanisms that can also prevent parts of a document from being further redacted. This problem was introduced in [18] as redaction with "disclosure control". Earlier work [21] gave a solution to the related problem of giving the signer more control over which subsets can be redacted and which cannot.

Recently the redaction problem has also been studied by various other authors. Ateniese *et al.* proposed sanitizable signatures that use chameleon hash functions instead of the usual hash functions, and allow redactors having their own secret key to modify some portions of the originally signed document where this was designated by the signer [1, 2]. Suzuki *et al.* proposed another construction called "sanitizable signatures with secret information" [22]. In their scheme, a signer can assign and change a condition for each portion of the document stating whether it can be redacted or not. Izu *et al.* proposed a scheme called PIATS, which addresses further redaction of the document in a different way [10]. In PIATS, not only the signer but also the redactor signs the redacted document. Here verifiers can learn who has redacted the document and avoid malicious additional redaction, although additional redaction is not prohibited directly.

Miyazaki *et al.* proposed "invisible sanitizable signatures" based on the aggregate signatures derived from bilinear maps [14, 16]. Their scheme enables a redactor to hide the number of redacted subdocuments as well as to assign disclosure conditions to each portion of the document. After their scheme was proposed, many other redactable signature schemes based on aggregate signature have been proposed, including that of Sano *et al.*, who proposed a visible and invisible redactable signature scheme called "sanitizable and deletable signature scheme" in [11, 20].

2.2 A motivating example

In this section we describe an example using healthcare records in which our techniques for redaction, pseudonym-

ization and deidentification are applicable. The problem of deidentification of health records has been studied by Sweeney [23, 24]. She has shown that in many cases it is possible to link presumably anonymous health records to the real world identity of the individuals. This linking of health records was possible using the attributes of zip code (postal code), birth date, and gender alone. These three attributes are sufficient to uniquely identify approximately 87% of the population of the United States, and are contained, along with the names of the corresponding individuals, in publicly available voter registration lists. The medical records so reidenitified included those of the Governor of Massachusetts. Thus it often desirable to deidentify health records before releasing them to the public.

The key tools for deidentifying tables of records used by Sweeney are *suppression* and *generalization* of data fields. Suppression corresponds to our notion of redaction. In generalization, a data field is made successively more general. For example, a zip code 32578 could be generalized to 3257*, and further to 325**, etc; a date could be generalized from *June 10, 1968* to *June 1968*, to *1968*, then to *1960-1969*. Generalization and suppression can be modeled as trees, where a suppressed attribute corresponds to a NULL element at the root of the tree.

Sweeney describes (heuristic) algorithms that can be used to deidentify tables using generalization and suppression to achieve a privacy property she calls *k-anonymity*. We are not concerned with the specifics of these algorithms and definitions here, but note that they are fully compatible with the integrity mechanisms we discuss in this paper.[1]

Consider the example healthcare data shown in Table 1a. For this data it is plausible that we may wish to:

- pseudonymize the patient name, in order to allow cross correlation of records,
- deidentify the race, birth dates, gender or zip code from some of the entries in order to prevent reidentifications as described above, *e.g.* using one of Sweeney's k-anonymity algorithms, and
- leave the medical diagnosis field unchanged.

The corresponding deidentitified table might now look as shown in Table 1b.

If we wish to establish the integrity of this kind of data using cryptographic techniques, then traditional digital signatures only establish the integrity of the data in isolation. Such techniques cannot be used to link the integrity of the pseudonymized and deidentified data to the original. In this paper, we show how to use redactable signatures to efficiently establish the integrity of both datasets in such a way that the integrity of the two can be linked.

2.3 Preliminaries

[1]Which deidentification method is appropriate in a given situation is often a difficult problem itself. Solutions needs to balance the utiltiy of the deidentified data with an assessment of which data are publicly known and with which the released data could be cross-linked, and they should take into account that the very fact that data have been removed from a specific record could leak information about that record.

2.3.1 Model

There are three sorts of players in our model: *signers*, *redactors*, and *users*.

The signer prepares and authenticates a document or data set once, producing an ordinary digital signature along with some auxiliary information; we will call the signature together with the auxiliary information an *extended signature* for the original document. The signature depends on the original data, on a list of operations to be allowed on parts of the data, and on the signer's private signing key.

The data and extended signature may be given to a redactor. The redactor may modify the data, according to the signer's list of operations allowed. Among the modifications that we treat in this paper are the redaction of parts of the data, marking certain parts of the data as nonredactable, and the replacement of parts of the data by pseudonyms or deidentifying generalizations. We will refer to a set of any of these allowed modifications as a *(generalized) redaction operation*. The redactor makes certain changes to the auxiliary information, and combines it with the original signature value to form a modified extended signature, which, together with the modified data, may then be "published", sent to another redactor, or sent to a user.

There may be more than one redaction operation, performed by more than one redactor, where subsequent operations will only be verifiable if they are performed according to the current modified form of the data and its extended signature (so that, for example, a pseudonym cannot be changed back to the subdocument that it replaced in a previous redaction operation).

A user is able to verify the correctness of the modified data using the (modified) extended signature, capturing the property that the data should only be modified by the redactor according to the specifications of the signer. Unlike the situation with ordinary signature schemes, where any change to the data should cause the signature verification to fail, here we want to allow a carefully specified set of changes to the data, while disallowing all other changes. We give a formal definition of security for redactable-signature schemes in §5.1, and prove that our algorithms satisfy the definition in §5.2.

All of the algorithms discussed in this paper can be stated in terms of any proofs of integrity that begin by hashing their inputs with a one-way hash function, including both digital signatures and time-stamp certificates. Precise definitions of the security of time-stamping schemes are not yet clear in the cryptographic literature (see [8, 3, 4]), and therefore we state all our security results in terms of digital signatures.

We describe our cryptographic algorithms in terms of how they apply to simple documents, viewed as strings of characters over an alphabet. Let m denote a document to be signed, segmented into a sequence of subdocuments m_1, m_2, ..., m_n. In the case of ordinary text documents, these might correspond to words, sentences, or paragraphs, depending on the level of granularity desired. We will use $\perp$ as a standard symbol denoting a redacted subdocument, agreed upon by convention by users of the system. In §4.1 below, we sketch an application where the "document" consists of a snapshot of the contents of a database, and subdocuments are records or fields of records in the database.

As a practical consideration, in any implementation the rendering algorithm that displays the document would have to decide how to display modified versions. For example, in

Name	Race	Birth Date	Gender	ZIP	Medical Diagnosis
Frank Miller	white	June 2, 1970	male	45873	chest pain
Mary Ross	white	Apr 10, 1964	female	45875	obesity
Howard Wu	Asian	Jan 17, 1958	male	45875	hypertension
Frank Miller	white	June 2, 1970	male	45873	HIV-related symptoms
Cathy Dunne	black	Sep 20, 1975	female	45874	short of breath

(a) unredacted

Name	Race	Birth Date	Gender	ZIP	Medical Diagnosis
Patient 1	white	1970		4587*	chest pain
Patient 2	white	1964	female	45875	obesity
Patient 3	Asian	1958		4587*	hypertension
Patient 1		1970	male	4587*	HIV-related symptoms
Patient 4		1975	female	45874	short of breath

(b) redacted

Table 1: Example healthcare data.

our algorithm redacted subdocuments may be represented in our data structure as, say, 160-bit apparently random bit-strings. In the simplest case, the rendering algorithm could display each one with 27 base-64 encoded characters. A slightly more complicated rendering algorithm might display them with more user-friendly values (*e.g.* "pseudonym1", "pseudonym2", etc.), and give the user the options of clicking on that value or hovering over it to reveal the actual value.

2.3.2 Cryptographic building blocks

The security of our algorithms relies on several cryptographic assumptions.

Let H denote a particular choice of collision-free hash function. (See [13], Chapter 9, for more details).

Let S be a digital signature scheme that is secure against existential forgery attacks by an adaptive chosen-message adversary [7]. (Strictly speaking, we assume—and in practice this is completely without loss of generality—that the signature scheme starts by hashing its input, and we use S to denote the operations after the computation of a hash value.)

Let $C(\cdot,\cdot)$ be a secure randomized *commitment* scheme, as can be constructed based on the existence of collision-free hash functions [9]. (In practice, one might implement C by simply taking $C(m,r) = H(0,m,r)$ with a collision-free hash function H, with 0 serving as a tag indicating input for the commitment scheme.) The output $x = C(m,r)$ of any invocation of the commitment function does not leak any information about the particular committed value m.

Let G be a secure length-doubling *pseudorandom generator*, as used in the GGM construction of pseudorandom functions [5]. Since we use it repeatedly throughout this paper, we sketch the construction here. Beginning with a single random seed s, the construction computes a list of pseudorandom values, by building a binary tree from the root to the leaves. Specifically, suppose that s is k bits long. The signer uses the pseudorandom generator to expand s to a $2k$-bit string, and lets the first and second k bits form, respectively, the left and the right children of s. (In practice, this could be implemented by computing $H(1,s)$ for the left child and $H(2,s)$ for the right child.) Continuing in this manner, we obtain n leaves.

2.3.3 A redactable signature algorithm

In this section we describe in some detail a particular redactable signature algorithm, due to [12]. In this algorithm, the additional data added to form the extended signature for the original document is of constant size, and only grows logarithmically with the number of sequences of consecutively redacted subdocuments.

The algorithm is as follows:

Setup: Given a security parameter, the signer chooses a collision-free hash function H, a secure pseudorandom generator G, a secure commitment scheme C, and a secure signature scheme S. The signer then generates a public-private key pair (PK, SK), publishes the public parameters (PK, H, C, S), and keeps the private key secret.

Sign: Given a document $m = (m_1, \ldots, m_n)$, the signer chooses a random seed s, and computes an n-leaf GGM tree, using G. Let $(r_1, \ldots, r_n)$ denote the list of these leaves. For each subdocument m_i, the signer computes $x_i = C(m_i, r_i)$. Next, the signer builds a Merkle hash tree from the list of leaves $(x_1, \ldots, x_n)$ to form the root h, and signs it with the private key SK, to get a signature σ. The extended signature for m is (s, σ). The signer sends the following information to the redactor in a secure channel: (m, s, σ). We assume that an adversary cannot obtain the information transmitted in the secure channel.

Redact: Given the document m, the redactor chooses L, the set of subdocument in m to redact, and proceeds as follows.

The redactor constructs the GGM tree from random seed s, and obtains n pseudorandom values $r_1, \ldots, r_n$. Let $m' = (m'_1, \ldots, m'_n)$ where

$$m'_i = \begin{cases} m_i & i \notin L \\ \perp & i \in L \end{cases}$$

and let

$$\begin{aligned} R &= \{r_i \mid i \notin L\}, \text{and} \\ M &= \{x_i \mid i \in L\}, \end{aligned}$$

so that R is the set of GGM leaves corresponding to non-redacted subdocuments and M is the set of commitment

values corresponding to redacted subdocuments.

Let S_G be the minimum set of subroots of the GGM tree that covers R. Let S_M be the minimum set of subroots of the Merkle tree that covers M.

The extended signature for m' is (S_G, S_M, σ). The redactor then sends the following information to the user over a secure channel: (m', S_G, S_M, σ).

The communication overhead of this algorithm can be compared to the baseline case where the redactor simply redacts subdocuments and re-signs the resulting document (with resulting signature σ', say). In such a case, the redactor would have to send to the user the following information: (m', σ'). Thus, the communication overhead from the redactor to the user of this algorithm is due to S_G and S_M, whose size we can bound as follows.

In general, this cost can be as large as $O(n)$. (For example, if L consists of exactly the set of even-numbered subdocuments, then S_G and S_M are each of size $n/2$.) However, in practice, especially for text documents and images, it is often the case that redactions consist of sequences of consecutive subdocuments; when this occurs, there will be considerable savings.

Consider first the case where L consists of a single sequence of j consecutive subdocuments to be redacted ($j \leq n$). The total number of tree nodes required—pseudorandom tree nodes included in S_G plus hash values included in S_M—is at most $O(\lg j + \lg(n-j)) = O(\lg n)$. If L contains s sequences of consecutive subdocuments, then the total cost is $O(s \lg n)$ tree nodes (each of length depending on the security parameter governing the lengths of the outputs of the pseudorandom generator and the hash function).

Verify: From the location of the $\perp$ symbols in m', the user can determine the indices that each root in S_G and S_M cover. The user expands each of the subroots in S_G to reconstruct R. From these values, the user can compute the set of commitments $\{C(m_i, r_i) \mid i \notin L\}$. The user combines these commitments with the subroots in S_M to compute the root of the Merkle tree, and verifies the correctness of σ as a signature on that root with respect to the public key PK of the signer.

3. NEW ALGORITHM: PROHIBITING REDACTION EFFICIENTLY

In this section we show how to extend the algorithm of §2.3.3 above in order to be able specify that any subdocument is *nonredactable*. This implements the capability first introduced by the authors of [18], while improving the efficiency of their algorithm.

The **Setup** operation is as before.

Once again, the **Sign** operation begins with the random choice of a seed value s. Instead of using s to build a GGM tree with n leaves, we now build a tree with $2n$ leaves, $(r_1, \ldots, r_{2n})$. Now each subdocument m_i is associated with a pair of pseudorandom values, r_i and r_{n+i}. We use r_i as before, to compute the commitment value $x_i = C(m_i, r_i)$; and we use the second value, r_{n+i}, to compute a hash value $y_i = H(3, r_{n+i})$, where 3 (or any standard constant) is a tag to indicate input to computations of H for use in exactly this place in the overall signature scheme. Finally, we build a Merkle tree from the list of $2n$ values $(x_1, \ldots, x_n, y_1, \ldots, y_n)$, and sign the root of this tree, with resulting signature σ. As before, the extended signature for m is (s, σ).

We use the second set of n leaves to extend the **Redact** operation to allow for the marking of subdocuments as nonredactable. As before, let L be the set of indices of subdocuments to be redacted, and now let L' be the set of indices of subdocuments that are nonredactable. Note that L and L' must be disjoint, as a subdocument cannot be both redacted and nonredactable. The interpretation of the status of m_i is summarized as follows.

		semantics
$i \notin L$	$i \notin L'$	ith subdocument can be redacted
$i \in L$	$i \notin L'$	ith subdocument is redacted
$i \notin L$	$i \in L'$	ith subdocument is nonredactable
$i \in L$	$i \in L'$	invalid

Figure 1: Semantics of verification algorithm.

We define m' as before, but now define R and M as follows:

$$\begin{aligned} R &= \{r_i \mid i \notin L, 1 \leq k \leq n\} \\ &\quad \cup \{r_{n+k} \mid i \notin L', 1 \leq k \leq n\}, \\ M &= \{x_i \mid i \in L\} \cup \{y_i \mid i \in L'\}. \end{aligned}$$

As before, we let S_G be the minimum set of subroots of the GGM tree that covers R, and let S_M be the minimum set of subroots of the Merkle tree that covers M. Here, the location of the $\perp$ symbols in m' are not sufficient to describe the range of indices covered by S_G and S_M. Therefore, each subroot must be prepended with a description of its path to the root. Such an encoding has length at most logarithmic in n.

The extended signature for m' is (S_G, S_M, σ). The redactor then sends the following information to the user over a secure channel: (m', S_G, S_M, σ).

The receiver of this information could also be another redactor who performs an additional round of redaction on the document. This intermediate redactor can further redact subdocuments that have not been previously marked as nonredactable. This is enforced via the semantics in Figure 1, and by the fact that this redactor cannot compute the necessary preimages in the Merkle and GGM trees to change a subdocument from nonredactable to redactable. Similarly, the redactor can mark subdocuments as nonredactable if and only if they have not been previously redacted.

As before, the communication overhead of this algorithm is due to S_G and S_M. Once again, we can expect that for many sorts of documents both the redacted parts and the parts to be marked as nonredactable will occur in sequences of consecutive subdocuments. If $L \cup L'$ contains s sequences of consecutive subdocuments, then we incur a cost of $O(s \lg n)$ tree nodes. Assuming tree nodes of length k, the extended signature is of size $O(|\sigma| + ks \lg n)$. When s is not too large, this compares favorably to the algorithm of [18], whose extended signature is of size $O(|\sigma| + kn)$, both for the original document as well as for any redacted versions of it, independent of the number or the distribution of the redacted and nonredactable subdocuments.

The **Verify** operation is changed to account for the changes in the data structure. As before, given (m', S_G, S_M, σ), the user recomputes the root of the Merkle tree, using m', S_G, and S_M, and then checks that σ is a correct signature for this root.

The user also makes another check, verifying that L and L'

are disjoint, *i.e.* verifying the correctness of the semantics in Figure 1 above. This prevents an adversary from redacting a subdocument that has been marked nonredactable.

4. APPLICATIONS

In this section we apply the algorithm of §3 to treat the the pseudonymization and deindentification problems motivated in §2.2, and to solve them efficiently.

4.1 Subsets of tables

Guaranteeing the integrity of the contents of a database is similar to the problem of secure document redaction, as follows. Computing an integrity certificate (for example, a digital signature) for a subset of the database that somehow ties to a signature for the entire database is exactly analogous to computing a signature for a redacted document that ties to a signature for the original document, where the subset corresponds to the nonredacted portion of the document.

Consider a 2-dimensional array of entries, consisting of r rows and c columns. We will consider the data in the array to constitute a single "document" whose rc entries are its subdocuments, taken row by row (or column by column, depending on the application). Building the GGM tree in order to sign the array according to the algorithm of §3 above, we take care to group the r pseudorandom leaves corresponding to the entries of each row into individual subtrees, and build the Merkle tree in a similar manner. When r is not an even power of 2, this will result in somewhat larger data structures for both the GGM and Merkle trees, but considerably reduces the size of the auxiliary data in the extended signature when entire rows of the array are redacted.

This application is of particular interest in the case of audit logs, which can be considered to be append-only databases. An audit report, computed as the response to a database query, often consists of a subset of the entire audit log, sometimes with certain entries redacted. In this case, our algorithm applies directly as a solution to the problem of accompanying the audit report with a proof of its integrity. For the case where as an additional constraint certain entries must be pseudonymized or deidentified, see below.

4.2 Efficient pseudonymization and deidentification

Once again, we will describe this algorithm in terms of a document $m = (m_1, \ldots, m_n)$. With the illustrative tables of §2.2 in mind, let us suppose that each subdocument m_i has an associated list of pseudonyms or deidentifying generalizations, denoted $p_i = (p_{i1}, \ldots, p_{il_i})$. For certain subdocuments, this list may be empty.

Now the signer can apply the algorithm of §3, not to the original document itself, but rather to an augmented form of the document, which is unambiguously encoded as each subdocument followed by a list of its successively more general pseudonyms. For example we might represent the first row of our table of healthcare data as

```
[ 'Frank Miller', 'Patient 1' ]
[ 'white' ]
[ 'June 2, 1970', 'June 1970', '1970', '1970-1979' ]
[ 'male' ]
[ '45873', '4587*', '458*', '45*', '4*' ]
[ 'chest pain' ].
```

We could then treat each bracket and token within the brackets in the above representation as an independent subdocument that can be redacted or marked for no further redaction. The signer could distribute such a version with each bracket pre-marked as nonredactable to preserve the relationship between the subdocuments and its pseudonyms.

Display conventions may vary according to the application. For example, we might require that if m_i is redacted (*i.e.* $m'_i = \perp$ in the modified version m'), then only the lexicographically first nonredacted pseudonym p_{ij} $(1 \le j \le l_i)$ is displayed by the rendering algorithm.

A variation that is especially appropriate for hierarchical generalization following Sweeney's approach (see §2.2) would require that only prefixes of the list $[\, m_i, (p_{i1}, \ldots, p_{il_i})\,]$ can be redacted (where the pseudonyms are listed in order from specific to general, so that each p_{ij} is more specific than $p_{i,j+1}$). Furthermore, this requirement can be enforced by adding this to the "semantic" checks that are validated by the **Verify** procedure.

To estimate the cost of this algorithm, let k be the length of the commitment values and hash values, and let $p = l_1 + \ldots + l_n$ denote the total number of pseudonyms. The augmented document has $n + p$ subdocuments, and and its initial extended signature is of size $O(|\sigma| + k)$.

After one or several redaction operations, let s denote the number of sequences of consecutive subdocuments in $L \cup L'$. The extended signature is then of size $O(|\sigma| + ks \lg n)$. The cost of signing or validating is $O(s \lg n)$ operations in addition to the "bare" digital-signature operation itself.

4.2.1 An efficiency improvement

The reader will observe that the algorithm just described requires the storage of an abundance of redundant data, especially in the case of certain data fields where the list of deidentifying pseudonyms for a data item is easily computable. The zip codes in §2.2 provide us a simple example, where 45873 has the possible pseudonyms 4587*, 458*, etc. In this case, we could simply make each of the five digits in the zip code a redactable character, instead of explicitly storing the list $p_i = (4587*, 458*, 45*, 4*)$ in the augmented document. Both the rendering algorithm and the **Verify** procedure must be suitably modified to handle data fields (subdocuments) containing zip codes. (For a completely different approach to this variation of the problem, see the techniques of [15], based on the Blum-Micali pseudorandom generator.)

Naturally, fields containing such data as dates or street addresses could be handled in a similar manner. But a similar approach can be used for more complicated data items, lacking a simple algorithm for computing the list of pseudonyms. For example, suppose we have data fields with items such as $m_i =$ `unicorn`, and $p_i =$ (`equine`, `ungulate`, `mammal`, `animal`). As long as there is a hierarchical classification of the deidentifying generalizations for the items in an identified subset of the data fields (subdocuments) of m, this classification of terms can be encoded in a "dictionary" that is appended to the document, marked as nonredactable, and the use of this dictionary can be suitably folded into the **Verify** procedure. If the dictionary is a completely standardized one, the augmented document only needs to include a persistent pointer to it (also marked as nonredactable).

Let d denote the size of the dictionary. Now the number of subdocuments in the augmented document can be reduced

from $n+p$ to $O(n+n\lg d)$, assuming the dictionary is standardized, and to $O(d+n+n\lg d)$ if the dictionary is sent along and signed with the document.

5. SECURITY

In this section we formally define our security requirements, and state and prove a theorem describing the security achieved by our algorithms. In order to analyze the security of our protocol, we give a formal game-based security definition, extending the usual definitions of security for encryption schemes and for signature schemes, that captures all of our desired security properties in a single game. Next we sketch a proof of security, reducing the existence of a successful adversary for our scheme to the existence of an adversary that successfully breaks one or more of the signature scheme, the pseudorandom generator, the commitment scheme, or the one-way hash function that our protocol uses.

5.1 Definitions

The principal requirement for any kind of signature scheme is that it should be computationally infeasible to forge illegitimate signatures. In contrast to conventional signature schemes, where no changes to a signed document are permitted, here we need a precise characterization of the class of modifications to the original document that we consider to be legitimate. Extending the definition used by [12], we define a partial order on redacted documents, as follows.

DEFINITION 1. *Let document m consist of n subdocuments $(m_1,\ldots,m_n)$. A* redacted version *of m is a sequence of n subdocuments $(m'_1,\ldots,m'_n)$ such that (for each $i=1\ldots n$), m'_i satisfies exactly one of the following conditions:*

1. *$m'_i = m_i$, indicating that m_i is (present and) redactable;*
2. *$m'_i = \perp$, indicating that m_i is redacted; or*
3. *$m'_i = \hat{m}_i$, indicating that m'_i is nonredactable (and identical to m_i).*

DEFINITION 2. *Let $p=(p_1,\ldots,p_n)$ and $q=(q_1,\ldots,q_n)$ be two redacted versions of m. We define a partial order on redacted documents by requiring that $p \prec q$ holds if and only if all of the following are satisfied, for $i=1\ldots n$:*

1. *if $p_i \neq \perp$, then $q_i \neq \perp$;*
2. *if $p_i = \hat{m}_i$, then either $q_i = m_i$ or $q_i = \hat{m}_i$; and*
3. *if $p_i = \perp$, then either $q_i = m_i$ or $q_i = \perp$.*

In this case, we also write $q \succ p$.

The partial order is defined so that $p \prec q$ if and only if p is a permitted redaction of q. For example, suppose m contains four subdocuments m_1, ..., m_4. If m_1 and m_3 are redacted in p, and m_1 is redacted in q, then $p \prec q$. In particular, the original document is $\succ$ any redacted version of it. Our goal is that given a (possibly redacted) document p, along with its extended signature, anyone can obtain an appropriately redacted document $p' \prec p$ with a verifiable extended signature, but it is infeasible to forge a signature for any document $p'' \succ p$.

By our definition of the partial order $\prec$, this *unforgeability* requirement also implies a sort of *consistency* requirement: If a subdocument of a document m has been marked as nonredactable, then it is infeasible later to produce a valid extended signature for a redacted version of m in which this subdocument has been redacted.

In addition to the unforgeability requirement, the redaction operations also introduce a requirements for *confidentiality*: Given a redacted document, no adversary can infer anything about the original version of any of its subdocuments that have been redacted. We capture this property by requiring that no adversary can distinguish two redacted documents p and p' whose corresponding original documents m and m' only differ at a specific subdocument, as in the definition of chosen-ciphertext security for probabilistic encryption schemes.

Next, we give a formal definition of security, adapted from those of [21, 18, 12]. We use a game definition extending both the definitions of security for encryption schemes [6, 19] as well as for signature schemes [7], capturing all of our desired security properties in a single game. We allow an attacker to issue *commit queries*, queries for commitments for documents, *sign queries*, queries for signatures for documents, and *redact queries*, queries for redacted versions. These queries may be chosen adaptively. Also, we allow the adversary to choose the document on which she can ask be challenged.

DEFINITION 3. *A redactable-signature scheme is secure if no probabilistic polynomial-time adversary, issuing a polynomial number of queries in the game defined below, achieves a non-negligible advantage in the game.*

The game proceeds as follows.

Setup: The challenger takes a security parameter as input, and runs the **Setup** algorithm. It gives the adversary the resulting public parameters (PK, H, C, S), and keeps the private key SK to itself.

Phase 1: The adversary issues several queries, where a query is one of the following:

1. *commit* query (m): The challenger computes a GGM tree and then commitments for the subdocuments in m. The commitments and random values used are given to the adversary.
2. *sign* query (h): The challenger signs the hash value h, using its private key.
3. *redact* query (m, L, L'): The challenger runs the **Redact** algorithm, following the instructions in L and L' to redact or mark as nonredactable the appropriate subdocuments in m. The challenger's response is the resulting quantities $(m', L, L', S_G, S_M, \sigma)$.

These queries may be asked adaptively. Also, the documents queried may be distinct. Once the adversary decides that **Phase 1** is over, she may choose a challenge for attacking confidentiality. (There is no need to choose a challenge for attacking unforgeability.)

Confidentiality challenge: The adversary outputs two equal-length documents m_0, m_1 on which to be challenged, such that m_0 and m_1 are identical except except in a single subdocument (the i^*th, say), along with **Redact** instruction lists L, L', with $i^* \in L$. The challenger picks a random bit $b \in \{0,1\}$, uses the **Sign** algorithm to produce a signature for m_b and then uses the **Redact** algorithm to produce

$(L, L', m_b, S_G, S_M, \sigma)$, where we require that L mark subdocument i^* to be redacted.

Phase 2: The adversary issues more queries, and the challenger responds as in Phase 1, with the sole restriction that the adversary cannot make any *sign* or *redact* queries for any document $m' \succ m_b$.

Guess: Adversary A outputs one of two kinds of guesses: either a guess for attacking confidentiality, or one for attacking unforgeability.

- **Confidentiality guess:** The adversary outputs a guess $b' \in \{0, 1\}$. The adversary wins the game if $b = b'$. We define its advantage in attacking the scheme to be $|\Pr[b = b'] - \frac{1}{2}|$.
- **Unforgeability guess:** The adversary outputs

  $$(m, L, L', S_G, S_M, \sigma).$$

  We define its advantage in attacking the scheme as the probability that no *sign* or *redact* query has been issued for any document m' satisfying $m' \succ m$ or $m' \succ m_b$ in Phase 1 or 2, and that **Verify**$(m, L, L', S_G, S_M, \sigma)$ is true.

THEOREM 1. *Assume that H is a collision-free hash function, S is a digital signature scheme that is secure against existential forgery, C is a secure commitment scheme, and G is a secure pseudorandom generator. Then the algorithm presented in §3 is an efficient secure redactable-signature scheme.*

We sketch the proof of this theorem in §5.2 below. The algorithm's efficiency is described in §3 and §4.

5.2 Security analysis

The security is proved based on the security of collision-free hash functions, commitment schemes with hiding and binding properties, and signature schemes secure against existential forgery. Next we give informal definitions of security for the cryptographic primitives that we use, omitting their formal definitions.

Security of collision-free hash function: a polynomial-time adversary has negligible probability of finding two different messages $m \neq m'$ with identical hash values.

Security of commitment scheme with binding and hiding properties: a polynomial-time adversary has negligible probability of breaking the hiding property by identifying from a commitment its corresponding message, randomly chosen as one of the two messages of her choice, and has negligible probability of breaking the binding property by finding a commitment that can be opened to two different messages [9].

Security of signature scheme with existential unforgeability: a polynomial-time adversary has negligible probability of forging a valid signature S of a signer on a message m such that the signer has never signed m [7].

Proof of Theorem 1: Let A be an adversary that has a non-negligible advantage against our redactable-signature scheme. We will construct an adversary B that uses A to gain advantage against the collision-free hash function, the secure commitment scheme, the secure signature scheme, or the secure pseudorandom generator. The adversary B acts as the challenger for A and uses A's outputs as her own outputs. B proceeds as follows.

Setup: B's challenger chooses hash function H, commitment scheme C, signature scheme S, and pseudorandom generator G for B to break. B's challenger gives B a public key PK of the signature scheme S. B then gives the adversary A the resulting public parameters (PK, H, C, S, G). Note that B does not know the private key SK of signature scheme S.

Phase 1: B answers A's queries as follows. The queries may be asked adaptively. Also, the queried document at each query may be distinct.

1. *commit* query (m): B runs the first several operations in the **Sign** algorithm on input m, including buliding a GGM tree and computing commitments, building a Merkle hash tree over commitments, and gathering auxiliary information that would be included in the extended signature. All of these values are given to the adversary A.

2. *sign* query (h): B cannot sign the root hash h of any Merkle tree that she calculates, because she does not have the private key. Therefore, B submits a signing query on the root hash to her challenger (of the signature scheme to break), and obtains a signature σ. (The game definition for security of signature schemes is not given here; please see [7]). Signature σ is given to the adversary A.

   B can similarly request a signature for any hash value h of her choice.

3. *redact* query (m, L, L'): B runs a commit query and a sign query on m to obtain the signature σ, along with the seed needed for the GGM tree that would be part of the extended signature for m. Then, B runs the **Redact** algorithm, following the instructions in L and L' to redact or mark as nonredactable the appropriate subdocuments in m, and computes the appropriate extended signature.

Once A decides that **Phase 1** is over, if she chooses she can issue a challenge for attacking confidentiality.

Confidentiality challenge: A outputs two equal-length documents m_0, m_1 on which to be challenged, such that m_0 and m_1 are identical except in the i^*th subdocument (along with **Redact** instruction lists L, L', with $i^* \in L$). B will attempt to use A's advantage in its confidentiality guess to break the hiding property of the commitment scheme. B needs to embed his commitment challenge in the challenge of A. The i^*th subdocuments in m_0 and m_1 are B's two messages of choice for breaking the hiding property of commitment scheme C. B's challenger generates a challenge for B as follows. B's challenger picks a random bit $b \in \{0, 1\}$, and computes a commitment of the i^*th subdocument in m_b. Denote this challenge as C_b^*. B uses C_b^* as the commitment of the i^*th subdocument in m_b. B then computes the commitments of the other subdocuments in m_b (using either m_0 or m_1) and obtains the root hash of the commitments.

Now B has embedded his commitment challenge at the i^*th position of A's challenge. For completeness, B asks her challenger to sign the root hash (as in a *sign* query), and obtains signature σ^*. And for auxiliary information (S_G, S_M), B chooses at random according to the protocol specifications.

Finally, B runs $\mathbf{Redact}(m_b, L, L', S_G, S_M, \sigma^*)$ in order to redact the i^*th subdocument in m_b, which can be computed without knowing the actual content of the subdocument. B gives adversary A the outputs of the **Redact** operation, to be used as the confidentiality challenge to adversary A. Readers can verify that the verification should be successful, even though B does not know b.

Phase 2: The adversary issues more queries, and B responds as in Phase 1.

Guess: Adversary A outputs either a confidentiality guess or an unforgeability guess. B uses A's outputs to attack one of the cryptographic primitives used: the hash function, the digital signature scheme, the commitment scheme, or the pseudorandom generator.

Confidentiality guess: If adversary A outputs a guess $b' \in \{0, 1\}$, then B outputs b' as his guess for breaking the commitment scheme. Because of the way in which B constructed his own commitment-scheme challenge, B's guess will be correct exactly when A's guess is correct.

Unforgeability guess: Adversary A outputs

$$(m, L, L', S_G, S_M, \sigma),$$

where we assume without loss of generality that no *sign* query has been issued for (the root hash corresponding to) any document $\bar{m}$ satisfying $\bar{m} \succ m$ or $\bar{m} \succ m_b$ in Phase 1 or 2. A's advantage in attacking the scheme is the probability that $\mathbf{Verify}(L, M, S_G, S_M, \sigma)$ is true.

Suppose first that σ is not equal to any of the signatures $\bar{\sigma}$ that were returned to A in response to a previous query in **Phase 1** or **Phase 2**. To convert A's output into a signature forgery, B constructs the Merkle hash tree for the redacted document m, and obtains the root hash h_r. Let h_1 and h_2 be the hash values at the two child nodes of the root node. In this case, σ is a correct signature for the message $h_1|h_2$ with respect to the public key PK, B can successfully attack the digital signature scheme.

We are left with the case that signature σ is equal to a signature $\bar{\sigma}$ that has been given to A in a previous query in **Phase 1** or **Phase 2**, corresponding to extended signature $(\bar{m}, \bar{L}, \bar{L}', \bar{S}_G, \bar{S}_M, \bar{\sigma})$.

B compares the Merkle hash tree constructed for m with the Merkle hash tree constructed for $\bar{m}$. If the two trees differ in their root nodes, then B can use this to produce a forgery for PK. If the two trees differ anywhere below the root, then B has found a hash collision.

Next, B compares the two GGM trees constructed respectively for m and for $\bar{m}$. If B finds any index i with subdocuments $m_i \neq \bar{m}_i$, then B has found a pair of messages that he can use to break the binding property of the commitment scheme.

Because of our stipulation that $m \not\prec \bar{m}$, the only remaining case is that for some index i, one of the following is true:

- $i \in \bar{L}$ but $i \notin L$ (so that A has computed a subdocument of m that was already redacted from $\bar{m}$); B can use this case either to break the hiding property of the commitment scheme, or to break the pseudorandom generator.
- $i \in \bar{L}'$ but $i \in L$ (so that A has redacted a subdocument of m already marked as nonredactable in $\bar{m}$); B can use this case either to break the one-way property of the hash function, or to break the pseudorandom generator.

6. REFERENCES

[1] G. Ateniese, D. H. Chou, B. de Medeiros, and G. Tsudik. Sanitizable signatures. In *European Symposium on Research in Computer Security (ESORICS) 2005*, volume 3679 of *Lecture Notes in Computer Science*. Springer-Verlag, 2005.

[2] G. Ateniese and B. de Medeiros. On the key-exposure problem in chameleon hashes. In *SCN '04*, volume 3352 of *Lecture Notes in Computer Science*. Springer-Verlag, 2004.

[3] A. Buldas and M. Saarepera. On provably secure time-stamping schemes. In *Advances in Cryptology — ASIACRYPT 2004*, volume 3329 of *Lecture Notes in Computer Science*, pages 500–514, October 2004.

[4] A. Buldas and M. Saarepera. Do broken hash functions affect the security of time-stamping schemes? In *4th Internatioanl Conf. on Applied Cryptography and Network Security – ACNS '06*, volume 3989 of *Lecture Notes in Computer Science*, pages 50–65, 2006.

[5] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, 1986.

[6] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, April 1984.

[7] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptively chosen message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.

[8] S. Haber and W. Stornetta. Secure names for bit-strings. In *Proceedings of the 4th ACM Conference on Computer and Communication Security*, pages 28–35. ACM Press, April 1997.

[9] S. Halevi and S. Micali. Practical and provably-secure commitment schemes from collision-free hashing. In N. Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 201–215. Springer-Verlag, 1996.

[10] T. Izu, N. Kanaya, M. Takenaka, and T. Yoshioka. PIATS: A partially sanitizable signature scheme. In *ICICS 2005*, volume 3783 of *Lecture Notes in Computer Science*, pages 72–83. Springer-Verlag, 2005.

[11] T. Izu, M. Sano, N. Kunihiro, K. Ohta, and M. Takenaka. Sanitizable signature schemes based on aggregate signature. In *Proceedings of the Symposium on Cryptography and Information Security (SCIS2007)*, 2007. (In Japanese).

[12] R. Johnson, D. Molnar, D. Song, and D. Wagner. Homomorphic signature schemes. In *Proceedings of the RSA Security Conference Cryptographers Track*, volume 2271 of *Lecture Notes in Computer Science*. Springer-Verlag, February 2002. Available at `http://www.ece.cmu.edu/~dawnsong/papers/hom-rsa02.pdf`.

[13] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[14] K. Miyazaki, G. Hanaoka, and H. Imai. Digitally signed document sanitizing scheme from bilinear maps. In *2005 Symposium on Cryptography and Information Security (SCIS2005)*, pages 1471–1476, January 2005.

[15] K. Miyazaki, G. Hanaoka, and H. Imai. Bit-by-bit sequence sanitizable digitally signed document sanitizing scheme. In *Proceedings of the Symposium on Cryptography and Information Security (SCIS2006)*, 2006. (In Japanese).

[16] K. Miyazaki, G. Hanaoka, and H. Imai. Digitally signed document sanitizing scheme based on bilinear maps. In *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, 2006.

[17] K. Miyazaki, M. Iwamura, T. Matsumoto, R. Sasaki, and H. Yoshimura. Digital documents sanitizing problem. Technical Report ISEC2003-20, 2003. IEICE Technical Report.

[18] K. Miyazaki, M. Iwamura, T. Matsumoto, R. Sasaki, H. Yoshimura, S. Tezuka, and H. Imai. Digitally signed document sanitizing scheme with disclosure condition control. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E88-A(1):239–246, January 2005. Available at `http://ietfec.oupjournals.org/cgi/reprint/E88-A/1/239`.

[19] C. Rackoff and D. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In J. Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91: Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 433–444. Springer-Verlag, 1992.

[20] M. Sano, T. Izu, N. Kunihiro, K. Ohta, and M. Takenaka. On sanitizable and deletable signature schemes. In *Proceedings of the Symposium on Cryptography and Information Security (SCIS2007)*, 2007. (In Japanese).

[21] R. Steinfeld, L. Bull, and Y. Zheng. Content extraction signatures. In *4th International Conference on Information Security and Cryptology — ICISC 2001*, volume 2288 of *Lecture Notes in Computer Science*, pages 285–304. Springer-Verlag, December 2001.

[22] M. Suzuki, T. Ishiki, and K. Tanaka. Sanitizable signature with secret information. In *Proceedings of the Symposium on Cryptography and Information Security (SCIS2006)*, 2006.

[23] L. Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):557–570, 2002.

[24] L. Sweeney. Achieving k-anonymity privacy protection using generalization and suppression. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):571–588, 2002.