Audit-log integrity using redactable signatures with pseudonyms

Stuart Haber William G. Horne Tomas Sander Danfeng Yao*
{stuart.haber, bill.horne, tomas.sander}@hp.com, dyao@cs.brown.edu

Abstract

In this paper we describe a new approach for the integrity of audit records. We show how to simultaneously establish the integrity of an entire audit data set and of any derived subsets, adapting techniques that have been used before for redactable signatures. In addition, our algorithms allow for the pseudonymization of data fields, cryptographically enforcing the consistency of chosen pseudonyms. The resulting schemes do not add significant computational overhead to a practical system, and are shown to be secure under reasonable cryptographic assumptions. We believe these algorithms can be a helpful tool to meet audit and reporting needs, in order to comply with such regulations as the US Sarbanes-Oxley Act (SOX). The algorithms enable proofs of the integrity of audit data and derived reports, while simultaneously providing means to protect privacy-sensitive information against internal and external consumers of these reports.

1 Introduction

Audit and event logs are routinely collected in IT systems for a variety of applications such as intrusion detection, forensics, fraud detection, network monitoring and quality control. Recently, audit logs and IT auditing have become increasingly important as a means of assuring compliance with financial and legal regulations, such as the Sarbanes-Oxley Act (SOX) in the US. Variants of this type of governance law are in the process of being adopted worldwide. A major goal of this legislation is to minimize the risk of fraudulent or erroneous financial reporting. As most of the financial and accounting aspects for corporations are handled by enterprise software systems ensuring their correct functioning as well as keeping reliable audit trails of system events that might have an impact on financial reports is key for the implementation of governance practices. Furthermore internal and external auditors regularly have to review the correct working of the controls that have been put in place.

For any reasonable use of audit logs, the integrity of the data must be maintained. In this paper, "integrity" means that the data has not been corrupted since it entered the system, either accidentally or maliciously. In this paper we are not dealing with the case where the data may have been corrupted before entering the system, for example due to human data entry errors.

*Work performed during an internship at HP Labs, Princeton.

For certain applications, we want to have strong assurances of data integrity without relying on virtual and physical access control as the primary means of protection. Cryptographic techniques are particularly well-suited for these situations.

However, cryptographic techniques are not directly compatible with certain practical requirements, for the following reasons.

- In most practical settings, an entire set of audit data is not usable directly. Audit logs can be voluminous, potentially consuming terabytes of storage. It is typical to query the data or use data-mining techniques in order to create useful views of the data. In addition, audit records may contain confidential or privacy-sensitive information that must be filtered before being given to certain parties. (The removal of sensitive data from documents is called *redaction*, and in this paper we adapt several techniques that have been proposed for proving the integrity of redacted documents.)

- It is often the case that information is subject to data lifecycle and retention requirements. In some situations, organizations are required to keep certain types of data for a specified time period, after which it may be desirable to delete that data, e.g. as dictated by Section 802 of the Sarbanes-Oxley law in the United States[1], after it is no longer needed for the purpose for which it was collected. In addition, users in many jurisdictions have the right to request that their data be removed from an organization's system.

The problem is that most cryptographic techniques only establish the integrity of an entire set of data in its original form. These techniques do not apply to establishing the integrity of any data derived or transformed from its original bit-sequence representation. Although the derived data could be signed again, this method would not establish a correspondence between the integrity of the original and the derived data. In this paper we suggest a practical solution to this problem for audit logs.

Another practical requirement we will be addressing in our proposed scheme is the need for *pseudonymization*. Rather than exposing user names in the clear to auditors, it is often sufficient and generally has much better privacy properties if user names can be replaced by pseudonyms. In order to allow auditors to be able to correlate user actions reliably we want to enforce cryptographically that pseudonyms are *consistent*, i.e. that all occurrences of the same user name are replaced by the same pseudonym.

The first contribution of this work is the application of redactable signature schemes to the problem of establishing the integrity of audit logs and derived subsets of audit data, as opposed to documents. (See §3.1 below for references to previous work on redactable signature schemes.) The second contribution is that we introduce new cryptographic schemes to maintain the integrity of data, even when parts of the data have been pseudonymized. We formalize a three-party trust model consisting of a data owner, a redactor, and a user (auditor), in which we provide a security analysis of the algorithms. We believe that this model addresses practically relevant concerns for the usage of audit logs in compliance. In addition, this use case raises additional research questions for which other natural audit reports data integrity proofs can be constructed.

The structure of the paper is as follows. In §2, we describe our model and the basic cryptographic building blocks that we need. In §3, we review previous work on redactable

signatures and describe the particular variant that we will use, and then go on to describe two variants of our pseudonymization and redaction algorithms for documents. In §4, we give a formal definition of our security requirements and a theorem stating the properties of our protocols. In §5, we adapt these algorithms to a dynamic audit log scenario and show how to address other practically relevant aspects of an integrity-enhanced auditing system. Finally, in §6 we list a number of open problems for future work.

2 Preliminaries

2.1 Model

There are three players in our model: a *data owner*, a *redactor*, and a *user*. The data owner prepares and authenticates the data once by producing a signature and some auxiliary information. The data, signature, and auxiliary information are given to the redactor. When a user submits a query for the data, the redactor redacts or pseudonymizes portions of the data according to some policy. The redacted values might be simply deleted all together or replaced by a special symbil indicated the value has been removed. If the redactor pseudonymizes the data, the data will be replaced with proper pseudonyms, which may be chosen by the data owner or the redactor. The data owner and the redactor may be the same entity. Intuitively, the "integrity" of pseudonymized data captures the property that the data should only be modified by the redactor according to the specifications of the data owner.

We describe our cryptographic algorithms in terms of how they apply to simple documents, which can just be viewed as strings of characters. In Section 5, we describe how these ideas extend to audit logs. Let m denote a document to be signed, segmented into a sequence of subdocuments $m_1, m_2, \ldots, m_n$. These might correspond to words, sentences, or paragraphs, depending on the level of granularity desired. Let $\perp$ be a standard symbol denoting a redacted subdocument, agreed upon by convention by users of the system.

2.2 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 [14], 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]. Let $C(\cdot, \cdot)$ be a secure *commitment* scheme, as can be constructed based on the existence of collision-free hash functions [11]. (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.)

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. It works by computing the list of values pseudorandomly from a single random seed s by building a binary tree from the root to the leaves. Specifically, suppose that s is k bits long. The data owner uses the pseudorandom generator to expand s to a $2k$-bit string, and let the first and second k bits form, respectively, the left and the right children of s. (In practice, this could be implemented by computing $H(0, s)$

for the left child and $H(1, s)$ for the right child.) Continuing in this manner, we obtain n leaves.

All of the algorithms discussed in this paper can be stated in terms of any sort of 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 [10, 4, 3]). Therefore we state all our security results in terms of digital signatures.

3 Redactable and pseudonymizable signatures

Several governments in the world have analogues to the US Freedom of Information Act (FOIA). Under this act, formerly confidential documents are released to the public. Typically, before a document is released, certain words containing sensitive information—such as names of individuals or national secrets—are *redacted*, i.e. blacked out.

For electronic documents, redaction is problematic when it is also important to protect the authenticity and integrity of the document. While conventional digital signatures and time-stamping schemes can be used to prove the integrity of the original data, these schemes do not work if the document has been changed in any way. So the problem is to devise a scheme that can be used to attest to the integrity of correctly redacted versions of the original document (and no other versions). This problem has been independently considered by several groups of authors, under different names: as "content extraction signatures" by [18], as "digitally signed document sanitizing schemes" by [15, 16], and as "redactable signatures" by [12].

In addition to redacting information, of particular interest in this paper is the ability to *pseudonymize* information, i.e. to replace some information with an alternative name so as to hide the actual value. An important property of pseudonyms is that they be used consistently. That is, if a subdocument is replaced with a pseudonym in one part of the document, any other occurances of the same same subdocument should be replaced by the same pseudonym.

We begin by reviewing a redactable signature algorithm due to [12]. Then we present two new algorithms for the verification of pseudonymized documents. In our first algorithm, the pseudonyms are determined pseudorandomly. The communication overhead from the redactor to the user is logarithmic in the number of pseudonymized subdocuments. In order to allow the data owner to have greater control, for example to preserve some intended meaning for the sub-documents, in our second algorithm, the data owner can choose the pseudonyms arbitrarily. The communication overhead of this algorithm is linear in the number of subdocuments. In both cases, our verification algorithm enforces the consistency of the pseudonyms.

As with conventional digital-signature schemes, it should be computationally infeasible to forge signatures. In addition, all information about redacted subdocuments should be hidden, other than their existence. These requirements are formalized in §4 below.

3.1 A redactable signature algorithm

Because our pseudonymizing algorithms are all based on it, we describe in some detail a particular redactable signature algorithm, which is a slight variant of the one due to [12]. In this algorithm, the additional data accompanying the original document is of constant size, and only grows logarithmically with the number of redacted subdocuments.

The algorithm is as follows:

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

Sign: The data owner chooses a random seed s. Let r be a vector of length n denoting the set of leaves of a GGM tree computed from s, using G. For each subdocument m_i, the data owner computes $x_i = C(m_i, r_i)$. Next, the data owner 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 data owner 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: The user requests the document m. Based on an appropriate policy, the redactor releases a redacted version of m. Let L be the set of indices of sub-documents to be redacted in document m. The redactor constructs the GGM tree from random seed s, and obtains n pseudorandom numbers $r_1, \ldots, r_n$. For $k \in [1, n]$, let

$$\begin{aligned} M &= \{m_k \mid k \notin L\}, \\ R &= \{r_k \mid k \notin L\}, \\ C &= \{C(m_k, r_k) \mid k \in L\}. \end{aligned}$$

Let G be the minimum set of subroots of the GGM tree that covers R. Let D be the minimum set of subroots of the Merkle tree that covers C.

The redactor then sends the following information to the user over a secure channel: (L, M, G, D, σ).

The communication overhead of this algorithm can be compared to a baseline case where the redactor simply redacts subdocuments and *re-signs* the document. In such a case, the redactor would have to send to the user the following information: (L, m, σ). Thus, the communication overhead from the redactor to the user of this algorithm is due to G and D, which is logarithmic in the number of redacted subdocuments.

Verify: The user expands each of the subroots in G to reconstruct R. From these values, the user can compute the commitments

$$C = \{C(m_k, r_k) \mid k \notin L\}.$$

The user combines these commitments with the subroots in D to find the root of the Merkle tree, and verifies the correctness of σ as a signature on that root with the respect to the public key PK of the data owner.

3.2 Algorithm 1: Commitments as pseudonyms

First, we consider the case where all of the subdocuments are distinct. The algorithm is straightforward. We use the redactable signature algorithm described in §3.1. However, we use the commitments $C(m_k, r_k)$ as the pseudonyms. The rendering algorithm that displays the document would have to decide how to display these pseudonyms. For example, suppose the commitments were realized as 160-bit hash values. In the simplest case, the rendering algorithm could display them as 27 base-64 encoded characters. A slightly more complicated rendering algorithm might display them as 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 commitment.

Now, suppose a subdocument can occur multiple times in the document. The above approach doesn't work because the pseudonyms will not be consistent since the commitments on which they are based each depend on a different random number.

Suppose there are l *unique* subdocuments. Then we can use a GGM tree to generate l random numbers — one for each unique subdocument. The data owner then builds a lookup table that defines for each $k \in [1, n]$ the index of the random number corresponding to subdocument m_k. The above algorithm can then be applied by using the commitment for m_k based on the random number r_j, where j is value of the lookup table at entry k. Similarly, the redactor can compute the same lookup table when computing pseudonyms.

As with the redaction algorithm in §3.1, the communication overhead of this algorithm is due to G and D, and so is logarithmic in the number pseudonymized subdocuments.

3.3 Algorithm 2: Data owner chooses pseudonyms

Our second algorithm relies on the data owner to specify pseudonyms. Details of the algorithm are as follows.

Setup: The setup is the same as in Section 3.1.

Sign: The data owner chooses a vector p of n pseudonyms. It is the responsibility of the data owner to choose pseudonyms that are consistent, i.e. if $m_i = m_j$ then the data owner should choose $p_i = p_j$.

As with Algorithm 1, a GGM tree with root s is used to generate n random values r. For each subdocument, m_i, the data owner computes $h_i = H(C(m_i, r_i), p_i)$. These hashes are then hashed together to form a single hash value $h' = H(h_1, \ldots, h_n)$. The data owner signs h' with the private key SK, to get a signature σ. The data owner sends the following information to the redactor over a secure channel: (m, p, s, σ).

Pseudonymize: The pseudonymization step is similar to the **Redact** step described in Section 3.1. Specifically, L is the set of subdocuments that are chosen for pseudonymization, and the GGM tree is built the same way and M, R, C, and G are computed identically. However, in this algorithm, the redactor then sends the following information to the user over a secure channel: (L, M, G, C, p, σ).

Verify: The user expands each of the roots in G to find the leaves r_k corresponding to those subdocuments that have not been pseudonimized. The user computes $x_k = C(m_i, r_i)$ from m_i and r_i for those subdocuments that have not be pseudonymized. For those subdocuments that have been pseudonymized, the user can obtain x_k directly from C. From, these values the user computes $h_k = H(x_k, p_k)$ and hashes them together to find h', and

verifies the correctness of σ as a signature on h' with the respect to the public key PK of the data owner.

In this algorithm the communication overhead is due to G and p. While G is logarithmic in the number of pseudonymized subdocuments, p is linear in the total number of subdocuments.

3.4 Variations

There are several variations on the above algorithms that are worth noting. First, in real documents only sensitive values are likely to be pseudonymized. It is straightforward to modify Algorithm 2 to give the data owner the flexibility to only specify pseudonyms for specific subdocuments. More importantly, it means that the redactor need only to communicate a smaller list of pseudonyms to the user. Now the communication overhead becomes linear in the length of this list.

The algorithms discussed so far have the property that the receiver of a partially redacted document may redact them further. The authors of [16] posed the question how to prohibit the redaction of certain subdocuments which may be desirable in some cases to preserve some intended meaning of the document. Earlier on, the authors of [18] gave a solution for a related problem of giving the signer more control about which subsets can be redacted and which not.

[16] gives a redactable-signature algorithm with the additional feature that any subdocument can be specified by the original signer or a subsequent redactor to be nonredactable. Their algorithmic ideas can be applied to our schemes as well and provide a similar functionality. A caveat of the original solution in [16] is that the size of the cryptographic companion data grows linearly $O(n)$, with n the total number of subdocuments and the constant being approximately the bit length of the used hash values. It is shown in [8] that the space complexity of these algorithms can be improved so that the size of the data accompanying the original document is of constant size and only grows logarithmically with the number of redacted subdocuments.

Using related technical ideas to the ones in [8] also the space complexity of our Algorithm 2 in which the data owner chooses the pseudonyms can be improved significantly.

4 Security

In this section we formally define our security requirements, and state a theorem describing the security achieved by the algorithms presented in §3 above.

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, in the case of redactable and pseudonymizable signatures we need a precise characterization of the class of modifications to the original document that we consider to be legitimate. Following [12], we define a partial order on redacted pseudonymized documents, as follows.

Definition 1 *Let document M consist of n subdocuments $\{m_1, \ldots, m_n\}$. Let P_1 and P_2 be two redacted or pseudonymized versions of M. $P_1 \prec P_2$ holds if and only if the set of*

subdocuments that are redacted or pseudonymized in P_2 is a subset of the set of subdocuments that are redacted or pseudonymized in P_1.

For example, suppose M contains four subdocuments $m_1, \ldots, m_4$. If m_1 and m_3 are pseudonymized in P_1, and m_1 is pseudonymized in P_2, then $P_1 \prec P_2$. In particular, the original document is $\succ$ any redacted orpseudonymized version of it. Our goal is that given a document P, anyone can obtain an appropriately redacted or pseudonymized document $P' \prec P$.

In addition to a suitable *unforgeability* requirement, the operations of redaction and pseudonymization introduce requirements for *confidentiality* and *consistency*.

Our confidentiality requirement is that, given a redacted or pseudonymized document, no adversary can infer anything about the original version of any of its subdocuments that have been redacted or pseudonymized. We capture this property by requiring that no adversary can distinguish two pseudonymized document 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 [6].

By consistency, we mean the requirement that all occurrences of a particular subdocument of a pseudonymized document should be given the same pseudonym if it is pseudonymized at all; and that distinct pseudonymized subdocuments should be given distinct pseudonyms.

Next, we give a formal definition of security in the random oracle model, adapted from those of [18, 16, 12]. We use a game definition extending both the definitions of security for encryption schemes [6] 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*, i.e. queries for commitments for documents, *sign queries*, i.e. queries for signatures for documents, *redact queries*, i.e. queries for redacted versions, and *pseudonymize queries*, i.e. queries for pseudonymized versions. These queries may be chosen adaptively. Also, we allow the adversary to choose the document on which she wants to be challenged.

Definition 2 *A pseudonymization protocol 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 k, and runs the **Setup** algorithm. It gives the adversary the resulting public parameters *param*, 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 commitments of subdocuments in M. The commitments and random values used are given to the adversary.

2. Sign query (h_r): The challenger signs the root hash h_r with its private key.

3. Redact query (M, L): The challenger runs the **Redact** algorithm to redact the subdocuments in M whose indices are in list L, the resulting redacted document M', its proof *Prf*, and the signature *Sig* of M are sent to the adversary.

4. Pseudonymize query (M, L): The challenger runs the **Pseudonymize** algorithm to pseudonymize the subdocuments in M whose indices are in list L, the resulting pseudonymized document P, its proof *Prf*, and the signature *Sig* of M are sent to the adversary.

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

Confidentiality challenge: The adversary outputs two equal length documents $M_0, M_1 \in \mathcal{M}$ on which to be challenged, such that M_0 and M_1 are identical except in the i^*-th subdocument. The challenger picks a random bit $b \in \{0, 1\}$, and sets $(P^*, Sig, Prf) = \textbf{Pseudonymize}(M_b, i^*, Sig, Info)$. It sends (P^*, Sig, Prf) as a challenge to the adversary. The adversary needs to guess whether M_0 or M_1 is used to produce the pseudonymized version P^*.

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

Guess: Adversary A outputs one or more of three guesses: one for attacking confidentiality, one for attacking unforgeability, and one for attacking consistency.

- **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 $(\tilde{P}, \tilde{Sig}, \tilde{Prf})$ where $\tilde{Sig}$ is the signature of $\tilde{P}$ and $\tilde{Prf}$ is the integrity proof. The constraint for choosing $\tilde{P}$ is that no sign query has been issued for any document $M' \succ \tilde{P}$ in Phase 1 or 2. We define its advantage in attacking the scheme as the probability that $\textbf{Verify}(\tilde{P}, \tilde{Sig}, \tilde{Prf})$ is true.

- **Consistency guess:** The adversary outputs $(\tilde{P}, \tilde{Sig}, \tilde{Prf})$ where $\tilde{Sig}$ is the signature of $\tilde{P}$ and $\tilde{Prf}$ is the integrity proof, such that (1) at least two identical subdocuments are given different pseudonym in $\tilde{P}$, or (2) at least two different subdocuments are given identical pseudonyms in $\tilde{P}$. We define its advantage in attacking the scheme as the probability that $\textbf{Verify}(\tilde{P}, \tilde{Sig}, \tilde{Prf})$ 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 algorithms presented in §3 are secure pseudonymization protocols.*

We omit the proof of this theorem, due to lack of space. The theorem can also be stated in terms of concrete security, depending on the security of its cryptographic building blocks.

5 Application to audit logs

It turns out that guaranteeing the integrity of audit data is similar to the problem of secure document redaction. Computing an integrity certificate (for example a digital signature)

for an audit report 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. In this section we apply the redactable, pseudonymizable signature system described above to arrays of data such as the database tables in which audit-log data is typically stored.

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 any of the algorithms 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 companion data required when entire rows of A are redacted.

The basic approach to verifying the integrity of queries against audit logs is straightforward. Essentially, audit messages are stored in a table in a database, viewed as an array, as just described. Any response to a database query that consists of a subset of the entries of the database, with certain ones of these pseudonymized, can be accompanied by a signature that proves its integrity. However, there are several practical issues that must be addressed that impose additional requirements on how the data is handled.

First, audit data arrives continuously. Therefore we cannot view the audit database as a static table but rather as a continuously growing one.

Second, raw audit data is typically not structured in a standard way. The audit data must be converted from its raw form into a structured form suitable for a database.

Third, audit systems in practice must be capable of handling large volumes of data without loss. As a result, it is a requirement that incoming records be first stored in *bulk storage* before any significant processing of the records can occur.

In the remainder of this section we discuss how these steps impact the integrity of the data and what steps can be taken to preserve integrity throughout this lifecycle.

5.1 Epochs

The audit database is constantly being appended with new records. Therefore, we cannot treat it as a single static table. To deal with this problem we partition the audit database into a set of disjoint *epochs*. Each epoch may correspond to a specific number of rows of the table, or to a time period during which records are appended to the database.

We apply any of the algorithms in §3 to the subset of rows corresponding to an epoch. The final hash value for the epoch's signature, i.e. the root of the relevant Merkle treem is linked together in a hash chain, which can then be signed with a digital signature algorithm, or time-stamped.

When users submit subset queries against the audit database, the result set is accompanied by the companion data for the redactable and pseudonymizable signatures in each epoch of data, and any additional hash values necessary to verify the hash chain over the relevant range of epochs. The signature or time-stamp on the last value in the chain is sufficient to verify the integrity of the entire set.

5.2 Bulk storage and "shredding"

We anticipate that in large systems, audit data will be arriving at a server in sufficient quantities that it may be infeasible to perform significant processing on the data in real time. Moreover, it may be desirable to preserve the data in its rawest form to be certain that processing operations did not modify or delete any relevant information.

Although processing may be limited, it may be possible to apply a signature or time-stamping procedure directly on the raw records to preserve the integrity of bulk storage, even if this does not allow the kind of subset queries that redactable signatures allow.

To get the records into a form that can be more efficiently queried, the audit records are "shredded"; that is, each audit record is parsed into a list of fields[1]. These fields form a database record that is appended to the *audit database*, consisting of a single designated table. In practice, fields resulting from any particular audit record may end up in several different tables in the database, but for simplicity we limit this discussion to just a single table.

How can the integrity of the audit database be tied to the integrity of the raw data in bulk storage? Here is one method, using the *content integrity service* (CIS). This service is a procedure that can be used to demonstrate that information in a long-term digital archive is authentic and has not been unintentionally or maliciously altered, even after its bit representation in the archive has undergone one or more transformations [9].

The essence of the CIS is to use a secure digital time-stamping system, first to time-stamp every document at ingestion into the archive, storing the resulting time-stamp certificate in the archive with the document; and second to produce an auditable record of every transformation to a document in the archive, in such a way as to verifiably link the time-stamp certificate for the transformed version of the document to its original form. This can be regarded as a generalization of the procedure for "renewing" digital time-stamp certificates or digital signatures that was introduced by the authors of [2].

We treat the audit data epoch by epoch. Consider first the case of a single epoch. As audit data enters bulk storage, each raw audit record is hashed, and the list of hash values from this epoch is used to build a Merkle hash tree, whose root R is signed.

Later, as a batch process, the records from this epoch are shredded, and the shredded data is entered into a redactable array. Let R' denote the root of the Merkle hash tree that would be signed in one of the algorithms of §3. Instead, the shredding process that transforms the raw data into the redactable parsed data that is to be stored in the audit database can be regarded as a "transformation", and CIS can be invoked for this transformation. The resulting CIS certificate is a signature authenticating the raw data encapsulated by the hash value R, the parsed data encapsulated by the hash value R', as well as the connection between the two.

For any single raw audit record a and its corresponding parsed version a', the CIS certificate for this epoch can be extended to give a CIS certificate for the single record, by adding to the certificate the list of sibling hash values on the Merkle-tree path from a's hash value $H(a)$ to R and the similar list linking $H(a')$ to R'. Presumably the end-to-end verification of this extended certificate would only be performed by a party that is permitted to see the entire record.

[1]The usage of "shred" is common in discussion of XML parsing.

Naturally, this use of CIS can be applied, epoch by epoch, to the entire audit database, if desired.

6 Further work

Financial and other audit applications motivate a number of additional research questions.

In this paper we described how to apply redactable signatures to the problem of verifying the integrity of subsets of data. However in many applications, it is necessary that audit reports contain *aggregate computations*, such as counts or sums of data that meet a certain condition. The research question is how to provide efficient correctness proofs for reports containing aggregation.

Another practical issue is that audit data may be collected across multiple systems, each of which locally stores its data. An interesting problem is how to establish the correspondence between data across these systems. Timeline entanglement is a cryptographic method to map a time step in the history of one service onto the timeline of another [13]. An interesting topic of research would be how to integrate redaction-enabled signatures with timeline entanglement. In this way we could provide more flexible integrity assurance for time-based audit reports, e.g. asking for a report on all the users that logged into the system within a specified time window.

Pseudonymization for audit data raises at least two further questions. First, pseudonyms as described in this work allow for linkability between any two reports in which the same pseudonym might occur. This is not always desirable, e.g. if one wishes to prevent different auditors from learning information by putting together two previously unrelated reports. Thus the question is how pseudonyms can be updated to prevent unwanted linkability, in such a way that the updated pseudonyms can still be used for integrity and correctness proofs. A second question that pseudonymization (and redaction) motivate is *generalization* of pseudonyms. For example, for a date like "Aug 10, 1973", one may wish to allow a redactor fine-grained control over *how much* information is redacted. Different natural generalizations of this date might include "Aug 1973" or "1973" or "1970-1979". How can we give efficiently integrity assurances for these generalized pseudonyms?

We assume for audit data that append-only databases are a primary requirement. Another interesting question is whether the kinds of techniques discussed in this paper could be applied to databases where frequent updates to data are allowed.

In terms of building practical systems to implement the algorithms described in this paper, much more work needs to be done to make this approach practical. Specifically, there are issues around key management, user interface design, and administrative tools that are crucial to turning this technical idea into a practical product. A complete privacy-aware solution may also need to filter the queries that are passed to the database. For example, consider a database that handles HR records. Simple redaction of a sensitive field such as 'salary' does not prevent breaches of privacy if users are allowed to ask for employee records with salary over $100,000. A similar problem was addressed in the work on privacy-enhanced access control by [17], and their methods might be applicable here as well.

References

[1] Sarbanes-Oxley Act. From 2002, 107 Pub. L. No. 204, 116 Stat. 745.

[2] D. Bayer, S. Haber, and W.S. Stornetta. Improving the efficiency and reliability of digital time-stamping. In R.M. Capocelli, A. De Santis, and U. Vaccaro, editors, *Sequences II: Methods in Communication, Security, and Computer Science*, pages 329–334. Springer-Verlag, 1993. (Proceedings of the *Sequences* Workshop, Positano, Italy, 1991.).

[3] 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.

[4] 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.

[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.L. Rivest. A digital signature scheme secure against adaptive chosen message attacks. *SIAM J Computing*, 17(2):281–308., April 1988.

[8] S. Haber, Y. Hatano, Y. Honda, W. Horne, K. Miyazaki, T. Sander, and S. Tezuka. Efficient redactable signatures with disclosure control, 2006. In preparation.

[9] S. Haber and P. Kamat. A content integrity service for long-term digital archives. In *Proceedings of Archiving 2006*. Society for Imaging Science and Technology, 2006. To appear.

[10] S. Haber and W.S. 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.

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

[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] P. Maniatis and M. Baker. Secure history preservation through timeline entanglement. In Dan Boneh, editor, *Proceedings of the 11th USENIX Security Symposium*, pages 297–312. USENIX Press, 2002.

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

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

[16] 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`.

[17] M. Casassa Mont, R. Thyne, and P. Bramhall. Privacy enforcement with HP Select Access for regulatory compliance. Technical Report HPL-2005-10, HP Labs, 2005.

[18] 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.