PRACTICAL AUDITABILITY IN TRUSTED MESSAGING SYSTEMS

Miguel Reis and Artur Romão
Novis Telecom, SA
Estrada da Outurela, 118-A, Carnaxide, Portugal

A. Eduardo Dias
Universidade de Évora
Rua Romão Ramalho, 59, Évora, Portugal

Keywords: Auditability, Trusted Repository, Secure Messaging, Secure Electronic Commerce.

Abstract: The success of a dispute resolution over an electronic transaction depends on the possibility of trustworthily recreating it. It is crucial to maintain a trusted, thus fully auditable, repository to which a judge could request a transaction recreation. This article presents a practical scheme providing strong guarantees about the auditability of a trusted repository. We use the messaging paradigm to present the mechanism, but it can be applied to any other scenario that needs to maintain fully auditable long term information.

1 INTRODUCTION

Common messaging systems, in particular electronic mail, do not possess enough security guarantees to satisfy most of the assumptions required on security demanding areas, such as military or business to business electronic commerce. Even secure electronic mail is not enough, since it only satisfies some requirements, like authentication, integrity, confidentiality and non-repudiation of origin.

Stronger security requirements, like non-repudiation of submission and non-repudiation of receipt (Kremer et al., 2002; Zhou, 2001), together with trusted auditability (Haber and Stornetta, 1997; Peha, 1999) from the message transportation and delivery systems, are not guaranteed at all. Furthermore, it is fundamental to assure the effective message delivery, or some warning about the impossibility of delivery, as well as reliable and secure (e.g., confidential) message archiving, needed for legal effects and long term availability.

This article presents an approach to maintain long term auditability of this type of electronic messaging systems, and it is organized as follows. In section 2 we present a series of required assumptions. In sections 3 and 4 we propose a new scheme and in section 5 we analyze its security. In section 6 we analyze the efficiency of the proposed scheme. In section 7 we present implementation guidelines using widely available tools. In section 8 we conclude the article.

2 SECURITY REQUIREMENTS

Messaging systems auditability is based on the possibility of recreating a transaction or a transaction set. This requirement demands the trusted storage of the set of messages belonging to a transaction. Every message, as well as additional attributes, is mapped to a specific record. A record is the basic unit of a trusted repository. We can define trusted storage as a series of assumptions made over a record:

- **Content integrity**: It is impossible to corrupt the content of a record without detection.
- **Temporal ordering**: Every record must be in chronological order, and this ordering cannot be corrupted without detection.
- **Record elimination**: It is impossible to delete a record without detection.
- **Record insertion**: It is impossible to add a non-authorized record without detection. We define authorized entity as someone possessing or having access to a secret needed to create records.

From this point on we will use the word ”validity” to refer to a situation where all the assumptions are achieved.

Reis M., Romão A. and Eduardo Dias A. (2004).
PRACTICAL AUDITABILITY IN TRUSTED MESSAGING SYSTEMS.
In *Proceedings of the First International Conference on E-Business and Telecommunication Networks*, pages 169-174
DOI: 10.5220/0001392001690174

3 A NEW AUDITABILITY SCHEME

In this section we describe a new scheme providing strong guarantees of meeting all the assumptions previously identified. Let us assume that all the records are kept inside a trusted repository. Let us also assume that message transportation from the messaging system to the trusted repository is done without corruption.

3.1 Notation

We use the following notation to represent data elements and functions in this article:

M : message belonging to a specific transaction

E : record element

E_1,E_2 : concatenation of two elements E_1 and E_2

$R = \{E_1,...,E_n\}$: record containing the concatenation of elements E_1 to E_n

f_m, f_e, f_h: flags indicating the purpose of a record

L : label linking a message with a specific transaction (transaction identifier)

$H_k(E)$: keyed message digest applied to element E using key k

$H(E)$: message digest applied to element E

$s_k(E)$: signature applied to element E using private key k

V_A e S_A : verification and signature key of principal A

$E_{\langle n \rangle}$: element placed in position n in an ordered list

$T(E)$: timestamp (Adams et al., 2001) applied to element E

3.2 The protocol

Whenever a message M is sent to the trusted repository a new record R_m is created in the following way:

$$R_m = \{f_m, L, M, Mac\}$$

$$Mac = H_k(f_m, L, M)$$

The Mac element is generated using elements present in the record. If any of these elements changes, the Mac element must change too, in order to keep the record integrity. This way we ensure that only who owns the secret k is able to change or add records to the repository. With this mechanism we satisfy the integrity assumption.

We now introduce the concept of an *epoch*. An epoch is defined as a set of R_m records, ordered in time, and completed with an R_e record. This type of record is defined in the following way:

$$R_e = \{f_e, k, V_A, Sig_e, T(Sig_e)\}$$

$$Sig_e = s_{S_A}(f_e, k, H(Mac_{<1>}, ..., Mac_{<n>}))$$

Sig_e is a digital signature made over a sequence of elements belonging to the R_e record together with a message digest element. The message digest is built from an ordered sequence of elements belonging to all R_m records which form this epoch.

As explained above using R_m records, R_e records can only be changed or added to the trusted repository by who owns a secret, which is, in this particular case, the signature key S_A. By using a message digest built over elements orderly collected from all the records R_m included in this epoch, the signature element Sig_e gives us the guarantee of content integrity, temporal ordering, non-elimination and non-authorized insertion of records without detection.

The message digest function referred above is created using Mac elements. This way we not only guarantee the integrity of these particular elements within each R_m record, but also the integrity of the set of R_m records belonging to this epoch.

So far we only guarantee the assumptions defined in section 2 within each particular epoch (as long as it is closed). But we must also guarantee that epochs are ordered in time, as well as the impossibility to completely remove one or more epochs without detection, thus certifying that the assumptions defined in section 2 are verified in all the trusted repository. To achieve this goal we need to modify the definition of the Sig_e element in the following way:

$$Sig_{e_{<n>}} = s_{S_A}(f_e, k, H(Mac_{<1>}, ..., Mac_{<n>}), H(Sig_{e_{<n-1>}}))$$

This way all of the R_e records are directly connected and ordered in time. Every R_e record has among its elements a reference to the immediately previous R_e record (as depicted in figure 1). For someone to be able to compromise one or more epochs without detection from the validation system, all of the epochs generated after those ones would also have to be changed. Assuming the signature key used in the last R_e record of the trusted repository is never compromised, we can state that with this scheme all of the assumptions defined in section 2 are fully satisfied.

R_e records act as checkpoints of validation throughout the repository. The concept of creating checkpoints and link them together follows the concept of a Merkle tree (Merkle, 1980).

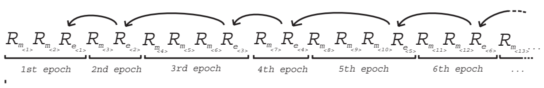

Figure 1: Example with a non-fixed number of R_m records in each epoch.

3.3 Record integrity verification

To validate the content of an R_m record it is necessary to go through the following steps:

1. Check which epoch the R_m record belongs to, thus identifying that epoch's R_e record.
2. Obtain $H(Mac_{<1>}, ..., Mac_{<n>})$. Mac elements are obtained from every R_m record belonging to the current epoch.
3. Obtain the previous epoch R_e record, using it to obtain $H(Sig_{e_{<n-1>}})$.
4. Check if the content from record R_e is not corrupted, using verification key V_A, the elements gathered in the previous steps and elements f_e and k to validate the digital signature present in element Sig_e.
5. Validate the content of record R_m by checking if the content from Mac element matches $H_k(f_m, L, M)$. We need to use the secret element k present in this epoch's R_e record.
6. Repeat steps 2 to 4 to all of the epochs created after this one, thus validating the chain of R_e records until the last one currently present in the trusted repository is correctly verified, or an error occurs.

4 HIERARCHICAL PARTITIONING OF RECORDS

The procedure explained above becomes impractical as the number of epochs increases. This is due to the fact that the number of record verifications that are necessary is directly proportional to the number of records in the trusted repository.

4.1 Definitions

To solve this problem we introduce a new hierarchical partitioning scheme. Instead of directly connect all the R_e records in the trusted repository, we now only directly connect R_e subsets. To explain this scheme we present a new notation:

$R_{[y]}$: record belonging to hierarchical level y

$R_{<l>}$: last record of a subset

The last record of an epoch subset, which is always an R_e record, is no longer directly connected to the first R_e record of the next epoch. Instead, it is now only directly connected to a hierarchically superior record. This new type of record will be defined as R_h. R_h records are also grouped in subsets, and directly connected one to another, just like explained to R_e records. In a generic way, every time a record subset is terminated with an $R_{h_{[y]_{<l>}}}$ record, a new hierarchically superior record level beginning with an $R_{h_{[y+1]_{<1>}}}$ record is created. The first record belonging to hierarchical level $y+1$ is always directly connected to the last record belonging to hierarchical level y (as depicted in figure 2). We now formally introduce this new type of record:

$R_{h_{[y]}} = \{f_h, V_A, Sig_{h_{[y]}}, T(Sig_{h_{[y]}})\}$, where

$Sig_{h_{[1]_{<n>}}} = s_{S_A}(f_h, H(Sig_{e_{<l>}}))$

$Sig_{h_{[y]_{<1>}}} = s_{S_A}(f_h, H(Sig_{h_{[y-1]_{<l>}}}))$ if $y \neq 1$

$Sig_{h_{[y]_{<n>}}} = s_{S_A}(f_h, H(Sig_{h_{[y]_{<n-1>}}}))$ on all other situations

4.2 Integrity Verification Procedure

With this approach, to validate the content of an R_m record we begin by going through all the steps defined in section 3.3 with some minor changes. We re-define the last step and extend the procedure:

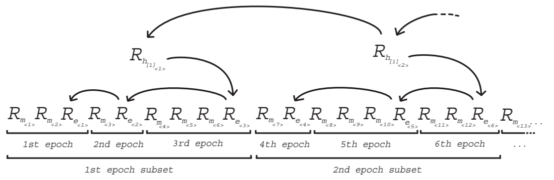

Figure 2: Hierarchical record scheme example.

6. Repeat steps 2 to 4 to all of the epochs created after this one and belonging to the current subset, thus validating a chain of R_e records.
7. Find which record $R_{h_{[1]}}$ is directly connected to the last record of the current subset, $R_{e_{<l>}}$, and check its content integrity by validating the signature in element $Sig_{h_{[1]}}$. Check also the integrity of all the other $R_{h_{[1]}}$ records belonging to the same subset.
8. Find which record $R_{h_{[y+1]}}$ is directly connected to the last record of the current subset, $R_{h_{[y]_{<l>}}}$, and check its content integrity by validating the signature in element $Sig_{h_{[y+1]}}$. Check also the integrity of all the other $R_{h_{[y+1]}}$ records belonging to the same subset.
9. Repeat the previous step until the hierarchically highest level as been successfully verified.

With the procedure explained above, verifying the integrity of some R_m record is no longer directly proportional to the number of existing records, as explained in section 6.

5 SECURITY ANALYSIS

The integrity of some record R_m is based on the security of the underlying message digest algorithm, as well as on the privacy of the secret k used to calculate Mac elements. Due to this fact, we should use a well known message digest algorithm, whose inviolability is perfectly demonstrated. We should also choose a secret in line with the computational capabilities available during the underlying epoch, minimizing the risk of well succeeded attacks over Mac elements. To prevent an attacker from manipulating R_m records, it is vital to keep the privacy of secret k assured as long as the current epoch is not completed with the generation of an R_e record.

The secret k used to build Mac elements is present in the respective R_e record. This is not a security weakness, since the epoch is closed and its security now lies on the integrity of the Sig element.

The content integrity of record types R_e and R_h is based on the security of the signature algorithm, as well as on the secrecy of the signature key. This leads us to conclude that epoch validity is dependent on the Sig element integrity. If the signature key becomes compromised the current epoch becomes also compromised, due to the possibility of re-signing it without detection.

In order for this compromise procedure to become fully undetectable it is also necessary to compromise the chain of directly connected records. This implies compromising all the R_e records belonging to the current subset and created later in time, as well as all the directly connected and hierarchically superior R_h record subsets (as depicted in figure 3). We can conclude this from the fact that every record belonging to the chain has among its elements a reference to the Sig element of some previously created record.

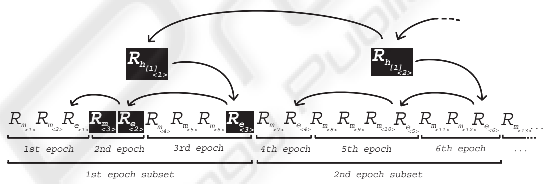

Figure 3: Undetected corruption of record $R_{m_{<3>}}$ implies compromising the chain of directly connected records.

This fact, together with the possibility of renewal of signature keys from one epoch to another makes undetected corruption extremely hard to achieve. Since signature key sizes can (and should) be adapted to the current computational power, the greatest security concern comes from the ability to maintain the secrecy of the signature key itself. We should eliminate the signature key after the end of an epoch, minimizing the risk of key disclosure. Since we are using asymmetric cryptography, this key is no longer necessary to verify a record's validity.

Nevertheless, it is possible that signature keys used a long time ago will become compromised (i.e., discovered), mostly due to technology breakthroughs. Even in this case, the record integrity may remain fully verifiable. All we need to do is to check if the T element remains valid. This element holds a timestamp of the Sig_e or Sig_h element, thus placing an upper boundary on the date the signature has been made. If this timestamp is later than the date the signature key has been or known to be compromised, the record is marked as valid. The T element is created by applying a digital signature to the target data. We assume the private key used to generate this signature is not compromised.

6 EFFICIENCY

Adapting secret k or signature key sizes to the current computational power is a basic operation, since both can be replaced after every epoch.

Creating R_e and R_h records takes much more time than creating R_m records, mostly due to the characteristics of asymmetric versus symmetric cryptography (Schneier, 1995). To minimize this constraint it is possible to increase the number of R_m records per epoch, keeping always in mind the fact that one must never allow the computational power available during the current epoch to be able to compromise secret k used to build Mac elements.

Secret k is revealed in the end of each epoch, thus making a validation over an R_m record a very easy and fast operation to conclude, even after long time periods. Verifying the validity of an R_m record implies the validation of $O(\log_x n)$ records, where

n = total number of records in the trusted repository

x = average number of records within each subset

7 IMPLEMENTATION GUIDELINES

In this section we present guidelines to an implementation of the scheme defined in the previous sections, through the application of technology widely used and proved to be secure and efficient nowadays.

The message digest algorithm used to create Mac elements must be widely deployed and proved to be secure, and at same time efficient, since it will be used very often. A keyed-hashing algorithm like $HMAC$ (Krawczyk et al., 1997) satisfies all of the above requirements.

In a similar way, it is important to use a widely deployed one-way hash function like $SHA-1$ (NIST, 1994) to generate elements which will be needed to create Sig_e or Sig_h elements.

Sig_e and Sig_h elements will be built using asymmetric cryptography. The private key is used to generate those elements and the public key, which will be bound to a $X.509$ (ITU-T, 2000) digital certificate, is used to verify the integrity of the elements, interacting with a certification authority (CA). Despite the fact that the trusted repository must fulfill certain security requirements, critical operations like certificate life cycle management are much more suitable to be done by a CA.

It is crucial to the preservation of the protocol security to establish a security scheme for the certificate requests (RSA, 2000) to be done. There are several good approaches: one is to use a mutually-authenticated TLS (Dierks and Allen, 1999) connection. Other may be by carrying some shared secret, which should be evolving from one request to the next, among the certificate request extensions.

Another point where we may increase security is by settling an agreement with the CA in which we can define a set of $X.509$ extensions to be included in all the certificates issued to the service. By issuing specific extension values for each digital certificate we may, for instance, lower the risk of certificate replacement frauds.

Validating an R_m record must also always require validating the digital certificate state by using an CRL (ITU-T, 2000) or an $OCSP$ (Myers et al., 1999) service. Whenever a certificate is found to be invalid (revoked, expired, etc.) it is necessary to validate the timestamp present in the T element, as explained previously, to decide if the record remains valid.

One final note concerning secret and private key protection. As pointed out before, it is extremely important to keep the items private. To achieve this goal we should use cryptographic hardware which allows us to generate secret and private keys inside an hardware token. The keys also have the possibility to never leave the token, thereby creating a very secure environment.

8 CONCLUSION

In this article we proposed a new scheme providing strong guarantees about the auditability of a trusted repository. The trusted repository maintains three types of records:

- R_m records keep the actual messages belonging to a specific and well defined transaction.
- R_e records aggregate sets of R_m records together, establishing epochs. This type of records are also aggregated in sets. In every set an R_e record keeps a reference to the R_e record created immediately before.

- R_h records are aggregated in directly connected sets and bound to a hierarchical level. Generically, every first R_h record of a subset belonging to hierarchical level y holds a reference to the last R_h (or R_e if y represents the first hierarchical level) record of a subset belonging to hierarchical level $y-1$.

We state that maintaining the secret k used to build R_m records private as long as the current epoch is not terminated, and adapting the size of this secret to the computational power available during the present time makes undetected corruption of R_m extremely records hard to achieve. Besides that, if we maintain strong guarantees that the private keys used in creating the last record of every hierarchical level are not compromised, undetected corruption of the complete chain of R_e and R_h records alse becomes extremely hard to achieve.

We have provided guidelines that prove informally that it is possible to make a practical implementation of this scheme through the application of technologies available in the present time. Although we use the trusted messaging system concept as a way of presenting the scheme, it can be applied to any system that needs to maintain fully auditable long term information.

REFERENCES

Adams, C., Cain, P., Pinkas, D., and Zuccherato, R. (2001). Time-stamp protocol (tsp). RFC 3161, Internet Engineering Task Force.

Dierks, T. and Allen, C. (1999). The tls protocol version 1.0. RFC 2246, Internet Engineering Task Force.

Haber, S. and Stornetta, W. S. (1997). Secure names for bit-strings. In *ACM Conference on Computer and Communications Security*, pages 28–35.

ITU-T (2000). Itu-t recommendation x.509. Technical report, ITU-T.

Krawczyk, H., Bellare, M., and Canetti, R. (1997). Hmac: Keyed-hashing for message authentication. RFC 2104, Internet Engineering Task Force.

Kremer, S., Markowitch, O., and Zhou, J. (2002). An intensive survey of fair non-repudiation protocols. *Computer Communications*, 25(17):1606–1621.

Merkle, R. C. (1980). Protocols for public key cryptosystems. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 122–134. IEEE Computer Society Press.

Myers, M., Ankney, R., Malpani, A., Galperin, S., and Adams, C. (1999). X.509 internet public key infrastructure online certificate status protocol - ocsp. RFC 2560, Internet Engineering Task Force.

NIST (1994). *NIST Federal Information Processing Standard Publication 180-1: Secure Hash Standard*.

Peha, J. M. (1999). Electronic commerce with verifiable audit trails. In *Proceedings of ISOC*.

RSA (2000). Pkcs #10 v1.7: Certification request syntax standard. Technical report, RSA Laboratories.

Schneier, B. (1995). *Applied cryptography: protocols, algorithms, and source code in C*. John Wiley and Sons, Inc., second edition.

Zhou, J. (2001). *Non-Repudiation in Electronic Commerce*. Artech House, first edition.