TARTU UNIVERSITY
FACULTY OF MATHEMATICS
Institute of Computer Science
Chair of Theoretical Informatics

Meelis Roos

Integrating Time-Stamping and Notarization

Master's Thesis

Supervisor: Helger Lipmaa

Author: ..... " ... " May 1999

Supervisor: ..... " ... " May 1999

Head of Chair: ..... " ... " May 1999

Tartu 1999

Contents

Abstract

In this thesis, we examine the current state of the most widely used public key infrastructure model and secure time-stamping systems. We begin by giving an overview of the evolution of the time-stamping techniques to show the direction of research.

We identify some problems in PKI and time-stamping that need solutions in practice. The problems are not very bad today but are becoming worse as time goes on and people start using these services more and more. Both problems are essentially scalability problems.

We show a simple and known solution attempt to the problem with PKI. The solution doesn't work either but gives a hint for building a working solution. Using the hints we present a working solution for the PKI problem. The new protocol eliminates certificate revocation lists and reduces the number of time stamps required since no time stamps are needed any more for the PKI itself.

We also point the similarities between the new system and some existing time-stamping systems. We analyze the similarities and develop another new protocol that integrates the first new protocol with time-stamping and solves the scalability problem for time-stamping. The integration allows to build the temporal authentication of signed data tightly into the PKI.

Introduction

Time Stamps — What and Why?

The use of digital documents is increasing rapidly. Electronic mail was the pioneer in this area — it has been used already for decades. Other usage areas of digital documents have become popular too in recent years. Word processors are not used only for printing but also to produce electronic documents for sending to other people. Electronic commerce and banking via the Internet are becoming quite common. Electronic documents are the future.

There exist elementary security measures for digital documents — signing and encryption. These measures are quite common and usable nowadays. There also exist needed infrastructures and protocols of managing and distributing the keys, certificates etc.

This is good but this is not sufficient. Most of the mechanisms work only for short-term documents. As an example, the cryptographic signature on a e-mail is important only at the moment of receiving the mail. The widely used cryptographic primitives stop working after the keys have been compromised. This works for documents with short lifetime but not for documents with long lifetime. The documents with long lifetime need additional measures to be taken for increasing their validity period.

The idea is to write down the time where the key certificates become valid and become invalid and compare the times to the time of signing of the documents in question. If the time of the signature lies between the start and end times of the validity period of the corresponding certificate then the signature is valid. This can be verified anytime in the future if the times are saved with document signatures.

The times on signatures need to be secure or they would be useless. The main subject of this paper is the theory of secure time stamps on documents.

What Is Missing

Neither the theory nor the practice of time-stamping are ready yet. There are several areas in the theory and between the theory and the practice that need further research. The author has identified two problems that need to

be solved but lack a clear solution yet — scalability of the time-stamping services and integration with Public Key Infrastructures.

Scalability

The current time-stamping systems and proposed protocols work well on one server but have weak or no methods for using a network of many servers. The only time-stamping systems so far that scale well in the number of servers must be unconditionally trusted and are "trivial" from the cryptographic viewpoint. Most of the non-trivial systems have only one central time-stamping server and have no mechanism for scaling to multiple servers. But this kind of scalability is surely needed when the time-stamping becomes more widespread.

Time Stamps as a Part of Public Key Infrastructure

The current time-stamping systems are just what the name says — time-stamping systems. Time-stamping is considered a stand-alone service that is not integrated with real usage areas like digital signatures etc. In real life the time stamps will probably be used mostly as a part of Public Key Infrastructure (PKI). The main need for the temporal authentication will be between the signatures and the validity periods of corresponding certificates. There are other uses too that are not connected to the PKI — like time-stamping of pieces of art to show that the author had it earlier than some pirate. But current predictions show that these other uses of time stamps will be a lot less common than the use in PKI.

Work In progress

The paper is based on work done in Küberneerika AS where the author works. our research group has studied time-stamping in last two years. This work has resulted in several papers about time-stamping: the concept of time-stamping for national use, [BLLV98] about binary linking schemes, [BL98] about new and more efficient linking schemes, [BLS99] about even more efficient linking schemes and [Lip99] about authentication graphs — generalizations of binary linking schemes. We have also produced a specification for time-stamping server that uses our linking schemes and the author has

programmed a test version of the time-stamping server. The current work concentrates mainly on PKI and putting the time-stamping systems into use in PKI. Our specialists also take part as scientific advisors from preparing the digital signature laws of Estonia.

Outline of the Paper

This paper consists of four chapters. Chapter 1 gives the background information about the previous results in time-stamping and describes the basic ideas of existing time-stamping systems. Chapter 2 contains overview and some critics about certificate revocation lists in Public Key Infrastructure. The critics give the motivation for further research. A well-known simple notarization protocol is given that tries to solve the problem but fails. Chapter 3 describes a new protocol — accumulated notarization protocol. This protocol solves the given problem with PKI. Chapter 4 describes another new protocol — accumulated notarization integrated with time-stamping. This protocol serves as the scalable notarization protocol but also provides temporal authentication for signed documents.

1 Background

This chapter gives an overview of the results that are needed later. For a complete overview of the results in time-stamping as of fall 1998 see [Jus98b]. Subsequential work has been done mainly by the researchers of Küberneetika AS, current results have been summarized in [Lip99].

One of the central terms in this paper is *time stamp*.

Definition 1.1. Loosely, a *time stamp* of a bit string is a token that binds information about *time* with the bit string.

The main problem is obtaining the time stamps for digital documents in a secure manner. The *time-stamping protocols* are used for this.

1.1 Trusted Third Party

The simplest time-stamping protocol uses a *Trusted Third Party* (TTP) that knows the right time. The client sends the message digest $H(X)$ of its document X to the TTP. The TTP adds current time t and puts its signature on the pair $(H(X), t)$. It sends the time t and the signature $\mathrm{Sig}_{\mathrm{TSS}}\{(H(X), t)\}$ back to the client. The client adds received values to the document. So the time is connected to the document and the correctness of the time is guaranteed by the signature. This works well if we trust the *time-stamping server* (TSS) and its source of time. We also have to assume that the key of the TSS is never compromised because after the compromise of TSS's secret key all the time stamps are under question.

1.2 Linking of the Time Stamps

Definition 1.2. A *one-way function* is a function that is easy to compute but intractable to invert.

The next natural move is to link all the time stamps together using supposedly one-way functions. It is computationally infeasible to insert documents into this chain later. The TSS keeps a registry of all issued time stamps and gives the time stamps out for verification. This idea was first published in [HS90] and [HS91]. The protocol:

- The client C sends a bit string Y to the TSS
- TSS knows the sequence number of the time stamp, be it n. The TSS computes $H_n = h(n, ID_C, Y)$ where h is a hash function and ID_C is the identity of C.
- TSS computes $L_n = H(H_n, L_{n-1})$. This is the linking information. The hash function H may differ from h.
- TSS sends $\{n, ID_C, L_{n-1}, \mathrm{Sig}_{TSS}\{L_n\}\}$ to C. This is the time stamp.

The L_n's are the linking information that holds the proof. The one-way dependent chain of L_n's from one document to another proves that the first document was time-stamped earlier than the second. During the verification, the intermediate L_i's can be requested from the TSS.

1.3 Binary Linking Schemes

Linear linking schemes have two obvious drawbacks. The first one is the efficiency — to verify the chain between two documents, the verifier must do the same amount of work that the TSS did between the two time stamps. This may be years' worth of work and is obviously too much.

The second drawback is the amount of information that the TSS must store for later retrieval by the verifiers. All L_n's must be saved in the archive.

Identification of these faults lead researchers to the development of exponentially more efficient *binary linking schemes* [BLLV98]. The idea is to link the time stamps not only to the element directly preceding it but also to some other element further in the past. The other link can be used to traverse the chain more efficiently by taking longer jumps. This reduces the amount of time needed to verify the temporal order of documents.

Definition 1.3. A *linking scheme* is an algorithm that tells which existing elements in the chain should a new element be linked to.

Another idea is to group the requests into *rounds* and traverse only between the summaries of rounds. By doing this we can reduce also the amount needed by the TSS to keep the data items required for verification. Such linking-schemes are called *accumulated linking schemes*.

It has been shown in [BLLV98] that there exist linking schemes that provide logarithmic length verification paths in the chain, enable the use of rounds and guarantee possibility of verification the temporal order between any two time stamps issued in one round and the verifier does not need any additional information to perform the comparison. The linking scheme is shown on figure 1.

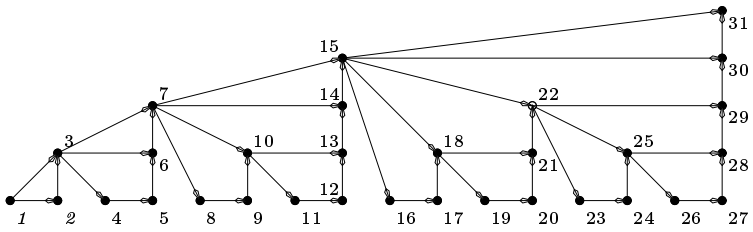

Figure 1: The linking scheme of BLLV98

The efficiency of intra-round linking schemes has been improved in [BLS99] (figure 2).

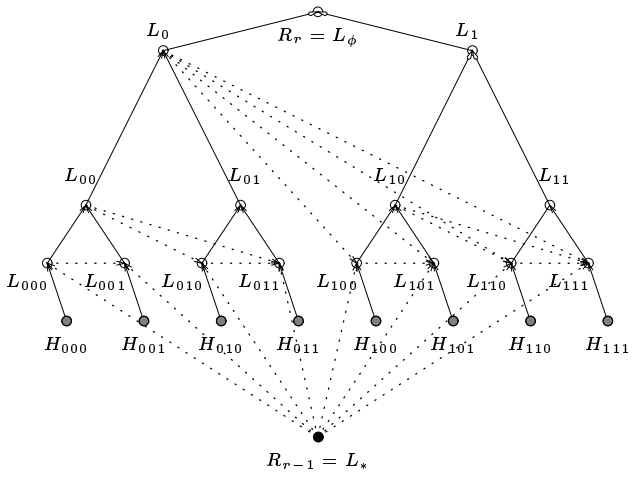

Figure 2: The efficient intra-round linking scheme of BLS99

Furthermore, it appears that the inter-round and intra-round linking schemes do not have to be the same ([Lip99]).

1.4 Relative Temporal Authentication

When talking about temporal authentication of data, we can distinguish between *absolute* and *relative* temporal authentication. In the case of *absolute temporal authentication* the time stamps contain information that is some representation of the time value as known in non-digital world. When comparing two absolute timestamps we compare the time values in them.

In the case of *relative temporal authentication* the time stamps contain information that can only be used to check whether one time stamps was made earlier than another. There's no analogue for the real absolute time.

While both schemes can be used to compare the time stamps there are deep differences between these two methods. It has been shown in [Jus98a] that absolute time can not be used without trusted third parties serving the absolute time. This means that we must trust both the trusted third party itself and its time source. For relative temporal authentication no trusted third party is required because the time stamps can be linked together with secure linking schemes that prevent forgery, as shown in [BLLV98].

However there are problems with relative temporal authentication too. While all digital documents can be made verifiable, the connection between the digital documents and the real absolute time is weak. With absolute time stamps we can associate the time-stamped documents with events in the real world. With relative time stamps this is not possible without additional work.

The most natural way to do it is to periodically time stamp a nonce (a bit string not known earlier — can be generated randomly). If I hourly time stamp a nonce and keep track of these nonces then I can use these time stamps to determine the time of some other digital time stamps with the precision of 1 hour. And that's all — anybody else has no reason to trust my hourly time stamps. If they would trust my hourly marks then it's like I would be the trusted third party for them.

So every party that wants to know the approximate real-world time for digital time-stamped documents must use its own periodical time stamps. To solve these problems in court, the court system must use its own periodical time marks.

1.5 Public Key Infrastructure

In 1976 W. Diffie and M. Hellman published their work about public key cryptosystems [DH76]. With the help of the RSA cryptosystem [RSA78] this ideology has became dominant in cryptography. Users have their public and private keys. The keep the private key secret and publish the public key with help of *public key certificates*. The certificates connect the persons and their public keys. A *public key infrastructure* is the infrastructure that deals with public key certificates and all problems associated with these certificates (like certification, certificate expiration, certificate revocation, distributing the certificate validity info etc). There are several paradigms of how the infrastructure should operate and which cryptographic systems and protocols to use. The X.509 standard ([HFPD99]) is the *de facto* standard for PKI in today's Internet. It has been developed originally for off-line operations and then extended to operate on-line too. The current version of X.509 is 3. Despite being already in the third incarnation it still has problems with scalability and delays in the distribution of certificate information.

2 Simple Notarization

2.1 Certificate Revocation Lists

The X.509 family of standards uses a hierarchy of *Certificate Authorities* (CA's) to issue and revoke the certificates. When a certificate is issued, it is expected to be in use for its entire validity period. However, various circumstances may cause a certificate to become invalid prior to the expiration of the validity period. Such circumstances include change of name, change of association between the subject and the CA (e.g., an employee terminates employment with an organization), and compromise or suspected compromise of the corresponding private key. Under such circumstances, the CA needs to revoke the certificate.

The current PKI standards [HFPD99] use *Certificate Revocation Lists* (CRLs) to distribute the information about revoked certificates. Each CA periodically issues a signed list of revoked certificates that have not yet expired. The list is valid for some fixed amount of time lasting at least to the next periodic issue of CRLs. The CRL consists of time-stamped certificate numbers. In the current implementations the CRL contents use absolute time stamps and do not use a time-stamping services for obtaining these stamps. When we want to get the reliable temporal authentication for the revocation info, we must use cryptographically secure time-stamping services.

When a certification-using system wants to verify a signed and time-stamped document, it must verify all these conditions:

- The signature is correct with respect to the certificate
- The certificate is not in a recent-enough CRL. If the certificate is in a CRL, the time stamps of the signed message and the corresponding CRL record are verified. The time stamp of the message must be earlier than the time stamp of the revocation record.
- The signature and the certificate of the CRL are correct and the certificate of the CA has not expired with respect to a higher level CA. The same method is used recursively.

The CRLs are issued periodically and each verifying operation must use a recent-enough CRL. The security policy specifies how recent the CRLs must

be. To get a good-enough proof of the certificate not being revoked, the entire current CRL must be transferred to the client. The client must then do a linear search in the CRL. This takes $O(n)$ time where n grows very large. In addition a delay is introduced because of the periodic nature of CRLs — we cannot use the latest information about the revocations but must use the latest published list.

Closer analysis of the transmission costs of the CRLs ([BCF+94] and [Mic96]) shows that the CRLs are very costly. For instance, if for US Federal PKI there are

- 3 000 000 users,
- each CA serves 30000 users,
- 10% of the certificates are revoked,
- CRLs are sent out biweekly,
- the verifiers of the signatures request certificate information for 5 signatures per day, and
- the communication costs are 2 cents per kilobyte,

then the total PKI yearly costs are $732 Millions, of which $563 Millions are due to CRL transmissions. The share of CRLs would be even greater when more certificates are verified per day ([Vil99]).

This doesn't count the resources needed to time stamp every CRL record if we want exact and secure temporal authentication. In this situation the main use of time stamps is to verify the temporal order of signing and certificate revocations. We want to eliminate the need of CRLs because of their inefficiency and to reduce the additional costs required for time-stamping.

2.2 The Idea Behind Notarization

A *notary* is an authority that certifies clients' signatures. Traditional notaries verify and sign the signatures of clients. Each notary has its own certificate from a CA. The main idea of simple notarization is simple: both the duties of the CA and the traditional notary are given to one principal. We call this principal *notary* here and hereinafter. When the client (say, A) wants to sign

a message M, it signs M and submits the signature $\mathrm{Sig}_A\{M\}$ to the notary. The notary signs the signature only if it finds the certificate of A to be valid. Since the notary and the CA are the same, the notary knows whether the certificate is valid at the moment of the signing. The verifier only needs to check whether the signature matches the document and whether the notary has signed the signature. The signature of the notary is sufficient to prove that to the best of the notary's (CA's) knowledge the certificate was valid at the time of the signing.

If the notary signs some signature made with a non-valid certificate then nothing bad happens. The signature of the notary makes the notary responsible for any damages when the user has revoked the certificate. The user may use several notaries simultaneously to protect itself from bad notaries that refuse to revoke the certificates.

2.3 Principals and Notation

The principals of the protocols in this section and hereinafter:

- C — the client of the notary (the signer)
- V — the verifier (the one who want to verify the signature of C)
- N — the notary (the notary+CA of C)
- N^* — the parent notary of N where applicable

By $\mathrm{Sig}_A\{B\}$ we denote the signature that the principal A has given on the bit string B. The signature contains the message digest of B encrypted with the private key of A.

By PK_A we denote the public key of the principal A.

By Auth_A we denote the credentials of the principal A. The credentials are meant to show the legal uses of the key (like signing contracts with level of responsibilities not above some fixed value).

By ID_A we denote the personal data of principal A that is needed to identify the principal outside the PKI.

By Cert_A we denote the certificate of principal A. It consists of PK_A, Auth_A and possibly also ID_A.

In the protocols, the notation $A \xrightarrow{n} B : X$ denotes protocol step number n, during which the principal A sends the data X to the principal B.

The asterisks on some of the variables mean that these variables come form the parent notary.

2.4 Certification Protocol

The simple protocols are taken from [Bul99]. The client C generates a pair of a private key and a public key. It sends the certificate request to N. The certificate request consist of its public key PK_C, identity ID_C and credentials Auth_C.

The notary generates the certificate $\mathrm{Cert}_C = \{\mathrm{PK}_C, \mathrm{ID}_C, \mathrm{Auth}_C\}$ and adds the certificate to its database of valid certificates. The notary doesn't need to send the signed certificate back to the client because there's no need for such certificate — each signature contains a notary-signed certificate anyway. So the notary just acknowledges the certification request.

The notary and the client should also agree on a *revocation password* of the given certificate (one-time password is a suitable example). The password may be necessary to revoke the certificate later.

We don't specify anything about the database of currently valid certificates that the notary maintains. The CRLs were linear lists; we can avoid the linear search in this database if it is appropriately organized. The database may use some faster methods to check whether a given certificate is valid or not (trees for $O(\log n)$ or hashes for $O(1)$ for example).

Protocol 2.1. Certification protocol 1

1. $C \xrightarrow{1} N$: $\mathrm{PK}_C, \mathrm{ID}_C, \mathrm{Auth}_C$
2. $C \xleftarrow{2} N$: ACK

2.5 Notarization Protocol

The client C signs a document X and sends the signature $\mathrm{Sig}_C\{X\}$ and the certificate Cert_C to the notary. The notary checks whether the certificate is

in its database of valid certificates. If it is then the notary signs the signature of C and the certificate with its own private key and sends it back to C.

Protocol 2.2. Notarization protocol 1

1. $C \xrightarrow{1} N \quad : \quad \mathrm{Sig}_C\{X\}$

2. $C \xleftarrow{2} N \quad : \quad \mathrm{Sig}_N\{\mathrm{Sig}_C\{X\}, \mathrm{Cert}_C\}$

The certificate of C is needed in the response since it is the only thing that binds the signature to the client C. The answer from the notary tells that the client C has produced a bit string that we call $\mathrm{Sig}_C\{X\}$ while the certificate of C was valid.

2.6 Verification Protocol

C sends the document X along with its own signature, its own certificate and the notary-signed signature to V. V verifies that the signature of C matches the document and the certificate and that the signature and the certificate are signed by N. V also checks whether Auth_C permits this kind of document to be signed with this certificate. V can also learn the identity of C from the certificate. V doesn't need to check any other sources of information, all necessary values must have been received with the document.

Protocol 2.3. Signature verification protocol 1

1. $C \xrightarrow{1} V \quad : \quad X, \mathrm{Sig}_C\{X\}, \mathrm{Cert}_C, \mathrm{Sig}_N\{\mathrm{Sig}_C\{X\}, \mathrm{Cert}_C\}$

2.7 Certificate Revocation Protocol

The client C signs and sends the revocation request $\mathrm{Sig}_C\{\text{REVOKE } \mathrm{Cert}_C\}$ to the notary. Or the client C calls the notary N and tells its certificate revocation password. The notary N removes the certificate of C from the database of valid certificates and no longer signs signatures that use this certificate. In fact the notary may even completely forget about the certificate if no other rules prohibit it — the certificate is not needed any more by the protocol.

Protocol 2.4. Certificate revocation protocol 1

1. $C \xrightarrow{1} N \quad : \quad \mathrm{Sig}_C\{\mathrm{REVOKE\ Cert}_C\}$

2.8 Disadvantages of Simple Notarization

The protocol assumes that the public key of N has been distributed to all the participants and that the key is valid through the whole period of the use of the protocols. This does not hold in real life. Another problem is scalability — one notary can serve only a limited number of requests in a time period. This may not be sufficient in real world.

We could build a simple hierarchy of the notaries — we require the signature of a higher level notary on the signature of N. The signature of document X would become
$X, \mathrm{Sig}_C\{X\}, \mathrm{Sig}_N\{\mathrm{Sig}_C\{X\}, \mathrm{Cert}_C\}$,
$\mathrm{Sig}_{N^*}\{\mathrm{Sig}_N\{\mathrm{Sig}_C\{X\}, \mathrm{Cert}_C\}, \mathrm{Cert}_N\}$.

We can extend this method to create a tree of notaries. This method would solve the problem of distributing most notaries' keys automatically but it doesn't scale either. The upper level notaries must do the same amount of work as all their clients together. When each notary in the hierarchy has about the same number of clients (the other notaries inside the hierarchy and the real clients as the leaves of the tree), the load of notaries rises exponentially from bottom to top. Something must be done to reduce the load on higher level notaries.

3 Accumulated Notarization

We now know that we probably want a hierarchical structure of the notaries. We constructed a simple tree in section 2 but the tree was not scalable enough. In this section we want to modify the simple notarization protocol to a scalable protocol. The idea of this protocol comes with slight optimizations from [Bul99]. The optimizations involve freeing the notary from the duty of verifying the actual signatures, in our system the notary just certifies which public key certificate was used. The actual verification is done by the verifier.

3.1 The Idea of Accumulated Notarization

The main idea of accumulated notarization is to make every notary do roughly the same amount of work. To reduce the amount of work needed by its parent notary, a client notary doesn't send each of its own signatures to the parent notary to sign. Instead, it groups the signatures into rounds. The length of the round is limited either by time, by the number of signatures in the round or by any other reasonable measure. At the end of each non-empty round it sends a summary of the round to the parent notary to sign. When it receives the answer it sends the summary of the round and the answer from parent notary to each of the clients. In addition, each client is sent a proof that its data was used to create the summary.

When every notary behaves this way and the tree of notaries is almost in balance then all the notaries do roughly the same amount of work. Suppose they have a round length of 1 second. The notaries receive requests, sign them and add them to the summary in some way. At the end of the round they submit the summary to the parent notary. When they receive the answer, they send the answer with other data back to the client. They may work on the next round while waiting for the answer of the previous round so the work does not stop.

The top of the hierarchy has no parent so it must behave a little differently. No rounds are necessary there since all the data is internal to the notary. So the topmost notary may just answer each request as soon as it is received.

The delay between the request and the answer from the client viewpoint raises from top to down. Each level in the hierarchy adds an additional delay from 0 to the round length of this notary. If there is a round length of 1 second for every notary, the average delay at each level is 0.5 s and the total

delay is $\frac{k-1}{2}$ seconds where k is the height of the hierarchy. This is not too bad since $k = O(\log n)$ where n is the number of notaries in the hierarchy.

3.2 Group Hashes

Definition 3.1. A *hash function* is a computationally efficient function mapping binary strings of arbitrary length to binary strings of some fixed length, called the *hash values*.

The group hashes are a generalization of hash functions of one argument. Normal hash functions take one arbitrary-length bit string as an argument and produce a fixed-length bit string as a result. H is the family of hash functions that generate n-bit output:

$$H = \{h : \{0,1\}^* \longrightarrow \{0,1\}^n\}$$

We use a subset of all possible hash functions — the cryptographic hash functions. The most important property of cryptographic hash functions is that it's computationally infeasible to find collisions, i.e. to find an A for an X such that $h(A) = X$ or to find A and B such that $h(A) = h(B)$. From the moment somebody invents a way to find collisions, the hash functions is not secure any more and is considered broken. Until the hash function has not been broken, the output value is the evidence that the input value was used to create the output value.

Group hash extends the hash functions to multiple arguments (a group of arguments — hence the name). The family of group hash functions may be defined as

$$G = \{g : (\{0,1\}^*)^* \longrightarrow \{0,1\}^n\}$$

(the functions take some arbitrary length bit strings as input and produce a n-bit result). Since we use the group hash values for evidence, we also require the group hash functions to be collision-free.

But this alone is not sufficient. The other property of normal hash functions — the output is the evidence that an input was used some way to create the output — is not automatic. The group hash function must be built so that for each input we can compute a proof that shows that the input was used to create the output.

Definition 3.2. A *group hash function* is a hash function that takes multiple arguments and provides a proof for each argument that the argument was used to compute the result.

The classical example of group hash functions is the algorithm of Benaloh and de Mare [BdM94]. It relies on the fact that RSA is hard to break. The group hash function g is defined as follows:

$$g(y_1, \ldots, y_n) = x_0^{y_1 \cdot \ldots \cdot y_n} \mod m$$

and the proof that y_i was used is

$$p(i, y_1, \ldots, y_n) = x_0^{y_1 \cdot \ldots \cdot y_{i-1} \cdot y_{i+1} \cdot \ldots \cdot y_n} \mod m$$

where x_0 is a constant and $m = pq$ is an RSA modulus factoring of which is unknown to any party. The value $p(i, Y)$ is given to the verifier that wants to verify whether y_i was used to calculate $g(Y)$. The verification succeeds iff

$$p(i, Y)^{y_i} \mod m = g(Y).$$

This scheme uses linear storage size for the proof. Better schemes like logarithmic and constant size also exist [Jus98b]. In practice there may be other needs for the group hashes — like the need of being able to distinguish the order of the inputs when the inputs come in sequentially.

The accumulated notarization protocol just uses the facts that the group hashes have smaller size than the total size of inputs and there exists a proof for every input that says the input was used to create the output.

3.3 Certification Protocol

The client C generates a pair of a private key and a public key. It contacts the notary and gives the certificate request to N. The notary must have some way to identify the client and check that the identity of the user is correct — else anybody could fake its identity and the certification would be no good. The certificate request consists of client's public key PK_C, identity ID_C and credentials Auth_C. The notary adds the certificate to its database of valid certificates and acknowledges the certification.

Protocol 3.1. Certification protocol 2

1. $C \xrightarrow{1} N$: $\mathrm{PK}_C, \mathrm{ID}_C, \mathrm{Auth}_C$
2. $C \xleftarrow{2} N$: ACK

We assume that all the notaries (N) in the tree have registered at their parent notary before any notarization involving the notary N begin.

3.4 Notarization Protocol

The same protocol is used between the final client of the notary service and between the notaries in the hierarchy. We choose a notary from the tree and call it N. C is a client of its and N^* it its parent notary.

The protocol begins as in section 2.5. The client C signs a document X and sends the signature $Y = \mathrm{Sig}_C\{X\}$ to the notary N. The notary checks whether the certificate of C is in its database of valid certificates.

When no problems are found, N accumulates $\mathrm{Sig}_N\{Y, \mathrm{Cert}_C\}$ to the current group hash. When the current round ends, N calculates the group hash L and sends the pair $\{L, \mathrm{Cert}_N\}$ to the parent notary N^* as a normal notarization request. It gets $\{\mathrm{Sig}_{N^*}\{L, \mathrm{Cert}_N\}, L^*, T^*, R^*\}$ in response. L^* is the value of the group hash for N^* (like L is for N). T^* is the proof that L^* is one-way dependent from L. R^* is the rest — the proof information from upper layers.

Protocol 3.2. Notarization protocol 2

1. C computes $Y = \mathrm{Sig}_C\{X\}$
2. $C \xrightarrow{1} N \quad : \quad Y, \mathrm{Cert}_C$
3. N computes the group hash value $L = h(\ldots, Y, \ldots)$
4. $N \xrightarrow{1} N^* : \quad L, \mathrm{Cert}_N$
5. $N \xleftarrow{2} N^* : \quad \mathrm{Sig}_{N^*}\{L, \mathrm{Cert}_N\}, L^*, T^*, R^*$
6. N compiles $R = \{\mathrm{Sig}_{N^*}\{L, \mathrm{Cert}_N\}, \mathrm{Cert}_N, L^*, T^*, R^*\}$ and the proof T.
7. $C \xleftarrow{2} N \quad : \quad \mathrm{Sig}_N\{Y, \mathrm{Cert}_C\}, L, T, R$
8. C compiles its own $R' = \{\mathrm{Sig}_N\{Y, \mathrm{Cert}_C\}, \mathrm{Cert}_C, L, T, R\}$

The root notary just responds with empty R since it has no parent notaries and thus the rest of the chain is empty. The root notary may still want to use rounds so it doesn't differ from the others too much.

We carry the certificate information along at all the levels. This is because some information is needed about the public key that was used to create the

corresponding signature. It should be possible to reduce the amount of this information, like with using message digests of the certificate and storing the whole certificate somewhere where it is accessible to the verifiers.

The size of R (and thus the size of the whole notarized signature) is linear to the height of the notarization tree. As the height $k = O(\log n)$ where n is the number of notaries, it's still only logarithmic to the number of notaries. So it's not too bad but it could be smaller.

3.5 Verification Protocol

Here C denotes the final client — the leaf of the tree.

C sends the document X to V. It also sends $\mathrm{Sig}_C\{X\}$ and R'. R' is essentially a chain of signatures from all the notaries from the notary of C to the top of the hierarchy.

Protocol 3.3. Signature verification protocol 2

1. $C \xrightarrow{1} V \quad : \quad X, \mathrm{Sig}_C\{X\}, R'$

V checks the following criteria:

1. $\mathrm{Sig}_C\{X\}(= Y)$ matches X and Cert_C
2. Auth_C (contained in Cert_C) permits this kind of signing
3. $\mathrm{Sig}_N\{Y\}$ matches Y and Cert_N
4. Auth_N (contained in Cert_N) permits this kind of signing
5. T shows that $\mathrm{Sig}_N\{Y\}$ was used to create L
6. ... (repeat last three lines for every level up to the root)
7. The certificate of the root notary matches the published and well-known one

If all these conditions are met then the signature is considered valid.

In fact the verifier checks that on each level, the higher level notary has properly notarized the signature and that the higher level notary had permission to notarize it at this moment. The latter is achieved by verifying that the notary signed the data given to it and submitted the data to a higher level notary for approval. All this is essentially the same as in the simple protocol. The difference is in the technique of submitting the notarized bit string for approval. Here the group hash helps to track that it was really approved.

3.6 Certificate Revocation Protocol

C sends N a signed revocation request or just calls N and tells its revocation password. N deletes the certificate of C from its database of valid certificates and doesn't respond to requests with certificate Cert_C any more.

Protocol 3.4. Certificate revocation protocol 2

1. $C \xrightarrow{1} N$: REVOKE $\text{Cert}_C, \text{Sig}_C\{\text{REVOKE Cert}_C\}$

The certificate in REVOKE request is needed to determine which certificate should be revoked. The signature is needed to avoid forgery of the revocation request (otherwise anybody could revoke my certificate if he knew my certificate from earlier communication). When the private key of a user has been compromised then anyone knowing the private key can revoke the certificate. This is only good since the certificate really needs revocation in this case.

3.7 Addition and Deletion of Notaries

The addition and deletion of the notaries is extremely simple with the current model of the protocol. The addition process has been already covered with the certification protocol — the same protocol applies to both notaries and real clients.

The deletion is also simple — the notary that wants to quit doesn't respond to any requests any more and it revokes its certificate at the parent notary. That's all. All issued signatures continue to hold since a higher level notary has signed them. No more signatures can be issued since the certificate has been revoked.

The protocol doesn't require any archive of the leaving notary to be kept so there's also no archive to transfer to any other notary. Note that while the protocol doesn't require any archives to be saved, some legal acts may still require them.

3.8 The Advantages of Accumulated Notarization

- It's scalable — all the notaries do roughly the same amount of work. So the upper level notaries don't have to do more work then the lower level notaries.
- No additional information is needed to check the signature if the notarization chain is given with the signed document. No CRLs, no time stamps.
- No protocol needs negative proofs (like "the certificate of A is not in any kind of blacklist")

4 Notarization with Temporal Authentication

It's interesting to note that we can use the linking schemes from time-stamping systems with rounds from section 1.3 as the group hashes. The reason we do this is because the notarization chain in accumulated notarization protocol has the same structure as one half of the time-stamping chain. We look at the notarization chain in terms of time-stamping and add another chain for the other direction of one-way dependence to integrate the properties of time-stamping into the notarization hierarchy.

Let's enumerate the client requests. At first the order of the requests is not significant — we can use just any order. Then we apply the linking scheme to the sequence of inputs. Since we have no connection to the earlier rounds of the notary at the moment we can use just any value for the first (initial) value as long as it is fixed. We obtain the round value and a head and a tail (as defined in [BLLV98]) for each of the inputs. For any input, the combination of the initial value, head, tail and the hash value is sufficient to prove that the input was used to calculate the hash value.

This kind of group hash doesn't guarantee anything more than we already have. But this approach gives us the hints where to look further if we also want the temporal authentication of the inputs.

4.1 The Construction

4.1.1 Step 1

Let's concentrate on a single notary. It receives requests from its clients, processes them by rounds and requests notarization of the group hash from its parent notary. The client requests are hashed together with a group hash. So far the only requirement for the group hash was that the clients could verify the fact that their request was used to create the hash value. Now we set up more strict requirements — the temporal order of the requests must be verifiable in the future.

We showed that we can use a linking scheme from time-stamping systems as the group hash but since we didn't connect the beginning of the chain to anything we didn't use a half of the power of the linking schemes. We could verify the order of inputs inside a round if we enumerated them in the

natural (incoming) order. But this was all, we couldn't get further into the future or into the past.

The obvious next step is to connect the subsequent rounds together (figure 3). We use the hash value of $(n-1)$-th round to get the first value of the nth round.

Figure 3: A chain with linked rounds

The way how the round summaries are connected is not important here. We can use any linking scheme that is efficient for us. The following protocols just require that the documents in the chain must be comparable. The choice of the linking schemes is out of the scope of this paper, both for intra-round and inter-round linking.

The linking schemes may require some intermediate round summaries to be present for inter-round dependency verification. We further assume that the information is kept available.

4.1.2 Step 2

Now let's have a look at the connections between the notaries. We continue to use the same hierarchy that connected the notaries in section 3. The submission of the group hash value to the parent notary can be seen as a link between the linking chains of these two notaries. We name these links the upward links — the direction of information goes from a lower level notary to its parent. This means that the element in the parent chain where the upward link ends is one-way dependent from the summary of the current round of the client notary.

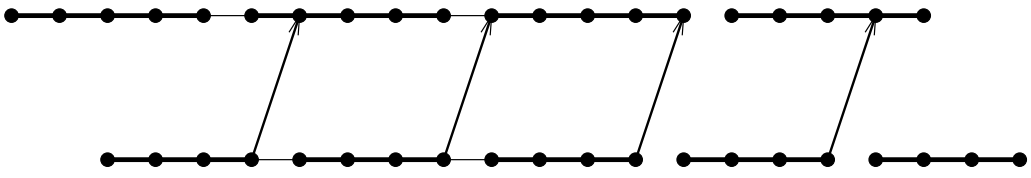

Figure 4: Two upwards linked chains

Figure 4 illustrates this. The upper chain denotes the parent notary and the lower chain denotes a client of its. The parent notary has the round length of 5 and the client notary has the round length of 4. The round lengths at each notary are not related at all — each notary may use even its own linking scheme inside and between its rounds. To be usable by the clients, the schemes must be published and know to the clients otherwise they couldn't check the correctness of the temporal dependency and signatures.

The time goes from left to right on this figure and all the other figures where time is important. At the end of each round the client computes the group hash and sends it to the parent to sign. The arrows don't go directly up but to the right since the transmission of the data takes some time. It gets the first empty place in the parent's current round.

4.1.3 Step 3

Now we want to add the downward links — the links that go from the parent notary to the client notary and carry linking information. These links are needed for achieving temporal dependence from the documents in the past — the dependency information comes through these links. There is also some other information transferred from the parent to the client — the signed answer to upward links — but this is not important for the linking and so the other information is not considered here.

The downward arrows create a one-way dependency between the group hash of the parent's last finished round and the client's freshly starting round. The client requests this information from the parent in some way that is again not important to the linking but is easily doable. This one-way dependency can be used to show that an element in the client chain has been notarized later than some previous elements in the parent chain.

Since the round lengths of the parent and the client don't have to be the same, it may happen that some round at the parent's chain is giving the downward linking information to several consequent rounds at the client chain when the client round length is smaller. Similarly, when the client's round length is larger that the parent's round length then it may happen that some round at the parent chain gives no linking information to the client. This good since the client gets the latest information always and so the probability of being able to compare two documents becomes larger.

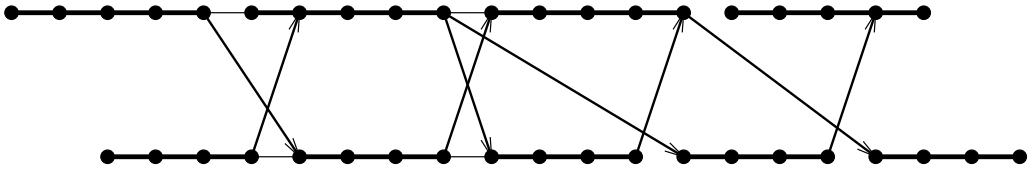

Figure 5: Two fully linked chains — like figure 4 but with downward links

4.1.4 Step 4

Now we have described the links between notaries and so we are able to put together a big tree of notaries all linked together. Each two notaries are connected the way we just described iff they were connected in section 3. There are no other connections in the tree.

The top-level notary has no upward links since it is the authority itself and doesn't need a signature from something else. The same goes for the time-stamping information — the top-level notary manages the "master" chain of linking information.

Similarly there must be the lowest level in the hierarchy with no downward links.

This is illustrated by figure 6. The figure shows that the resulting graph is similar to a sheet of paper folded into two in the top and broken into three at point C and just folded a little at point B. All the arrows are on the sheets of paper but not between the sheets.

The figures 7 and 8 illustrate the dependency paths in the graph. Figure 7 shows the maximum predecessors of a top-level linking element. There exists a one-way dependency from all the predecessors in the figure and all the earlier elements on their levels to the top-level element. There exist no such proof for any later element at any level. Similarly, figure 8 represents all minimal successors of a top-level element. There exists a one-way dependency from the top-level element to any of the successors in the figure and any element right of them and there doesn't exist such a proof for any earlier element.

The figures together give a good view about how the dependency propagates in the graph: from the earlier element up to a common notary, then some time along the chain of the notary and then down to the later element. It's

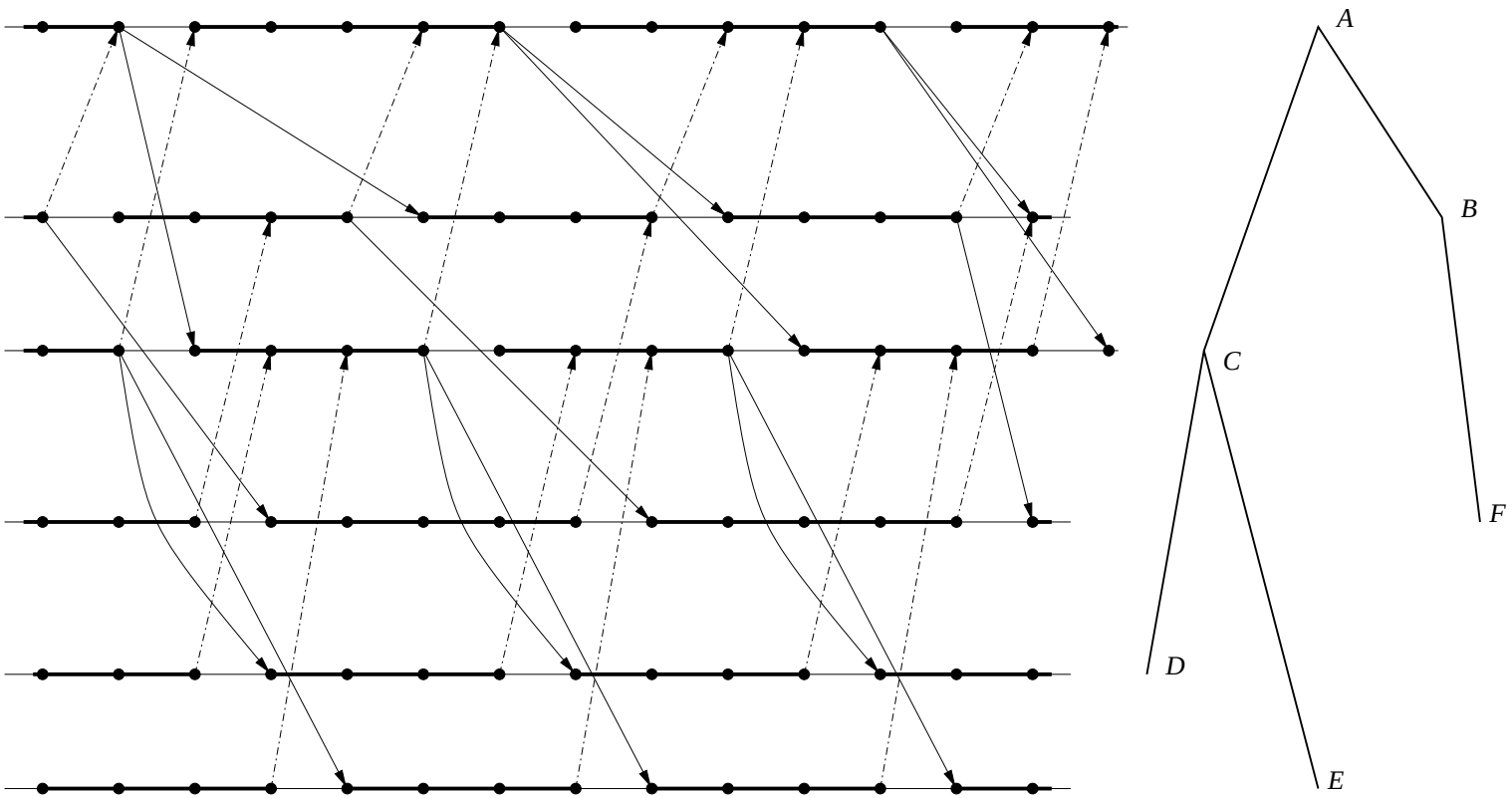

Figure 6: The general schema with 6 notaries and all upward and downward links

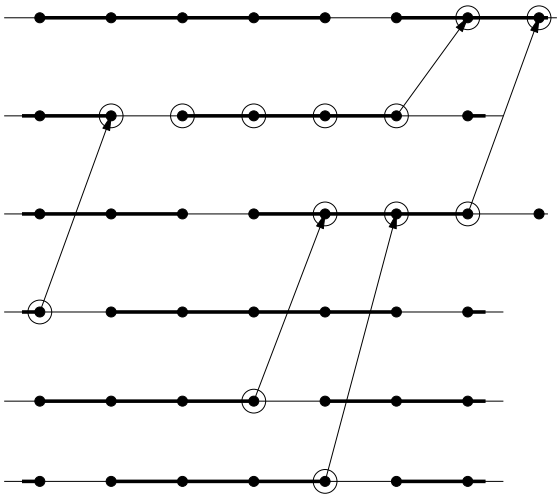

Figure 7: The predecessors of an element

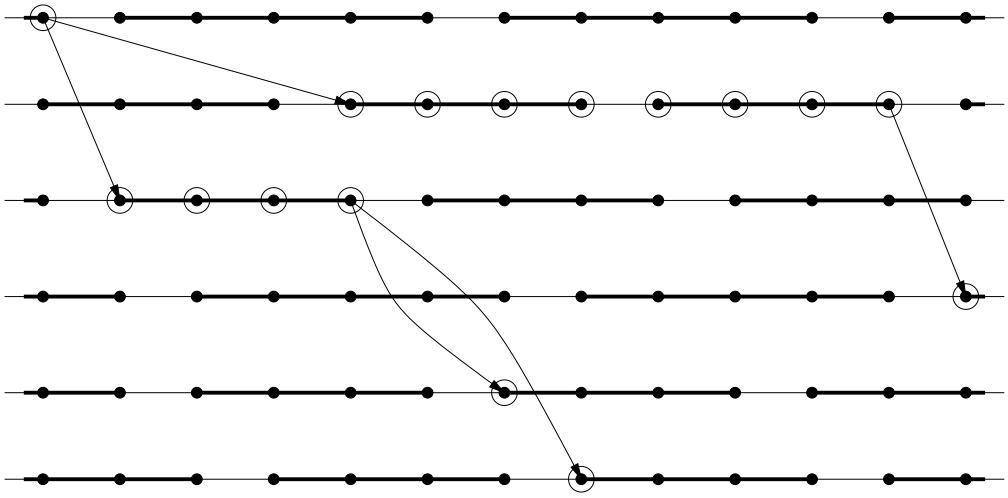

Figure 8: The successors of an element

sufficient to go up only until there is a common notary, i.e. a notary that is in the path from the earlier element to the top and in the path from the top to the later element. In the worst case the top-level notary is the only common notary. In the best case the elements are clients of the same lowest-level notary and no up-down links between the notaries are necessary at all.

The upward path of the document is the linking information for comparing against future documents and the downward path is for comparing with past documents. The client connects a downward path and an upward path together with the document signature. It first asks the downward linking information from the notary, adds it to the document information (the document itself or a message digest) and notarizes the result. This creates the signature on the document and also adds the upward path. The triple consisting of downward path, document and upward path (also contains the signature) is the whole unit of temporal authentication.

The conventional time-stamping system that was orthogonal to PKI had an annoying property: it required two time stamps on a signed document: one before the signing (for showing that the signature was made later than some documents) and one after the signing (for proving that the signature was made before some other document). Our integrated system removes this need since we really need only one half of each time stamp.

4.2 Certification Protocol

The certification protocol is the same as the certification protocol in section 3.3. The client C generates a pair of a private key and a public key. It contacts the notary and gives the certificate request to N. The notary must have some way to identify the client and check that the identity of the user is correct — else anybody could fake its identity and the certification would be no good. The certificate request consists of client's public key PK_C, identity ID_C and credentials Auth_C. The notary adds the certificate to its database of valid certificates and acknowledges the certification.

Protocol 4.1. Certification Protocol 3

1. $C \xrightarrow{1} N$: $\mathrm{PK}_C, \mathrm{ID}_C, \mathrm{Auth}_C$
2. $C \xleftarrow{2} N$: ACK

We again assume that all the notaries (N) in the tree have registered at their parent notary before any notarization involving N begin.

4.3 Downlink Protocol

The downlink protocol is for making the downward links in the graph. At the beginning of a new round each notary (except the top-level notary) asks the group hash value of the last finished round from its parent notary. The parent notary gives the value as the answer (G), the group hash value from its parent's last finished round (G^*), the proof (H) that G^* was used to compute the value of G and the rest (S). The rest is the downlink information that came with G^*.

The downlink information G is used as the initial value of the new round of N. It is also used by the final clients to associate it with the documents that are to be signed.

Protocol 4.2. Downlink protocol 3

1. $N \xrightarrow{1} N^*$: REQUEST
2. $N \xleftarrow{2} N^*$: G, G^*, H, S
3. N computes its own $S' = \{G, G^*, H, S\}$

When a client asks the downlink information from the notary N, N answers with G, G^* and H from its last finished round and uses S' for the rest. So the rest gathers into S on the way down.

The top-level notary must have G in the answer since G is used as a seed on lower levels. It may not have G^* and H since it may give out just its latest element in the linking chain as the seed for lower levels. But the top-level notary may use rounds and G^* and H as well if it chooses so.

The one thing that is certainly different about the top-level notary is that it doesn't have anything to add in S. Instead it should sign the G that it puts in the answer and put the signature into S. This is necessary to for any lower level clients for deciding whether to trust the downlink chain that it gets from the parent notary. Otherwise the notaries in the path from the top-level notary to some other fixed notary may choose to provide false information

that can not be used by the client later because it has no proof value. Adding the signature of the top-level notary helps against false linking information but doesn't help against too old linking information. So the clients should regularly check whether their documents are reliably comparable with respect to temporal order. Documents notarized at different notaries must be used for this kind of check else the linking information is not used in the checks.

The signature of the top-level notary is sufficient — no signatures from lower level notaries are required. These signatures would not add any useful proofs and they would require putting the certificates of the intermediate notaries into S.

The downlink information is independent from the uplink information. There are two points where the two directions meet: the client associates both downward and upward linking information with the document and the verifier finds a connection between the upward links of the older document and the downward links of the newer document.

4.4 Notarization Protocol

The same protocol is used between the final client of the notary service and between the notaries in the hierarchy. However, the final client has to prepare for the protocol slightly differently. While normal notaries must have used the downlink protocol to get the initial element for the round, the client must use the downlink protocol too but it uses the information in creation of the initial signature of the document.

We choose a notary from the tree and call it N. C is a client of its and N^* it its parent notary of N.

The client C has a document X and the current downlink information S. It signs the pair $\{X, S\}$ and sends the signature $Y = \mathrm{Sig}_C\{X, S\}$ to the notary N. The notary checks whether the certificate of C is in its database of valid certificates.

When no problems are found, N accumulates $\mathrm{Sig}_N\{Y, \mathrm{Cert}_C\}$ to the current group hash. When the current round ends, N calculates the group hash L. The calculation also yields Z that is the actual bit string that represents Y in the group hash. Y can't be used directly because the value in the hash must depend on Y and the previous elements in the hash.

N sends the pair $\{L, \mathrm{Cert}_N\}$ to the parent notary N^* as a normal notarization request. It gets $\{\mathrm{Sig}_{N^*}\{L, \mathrm{Cert}_N\}, P^*, Z^*, L^*, T^*, R^*\}$ in response.

Z^* is the representation of L in the group hash of the parent as described above. L^* is the value of the group hash for N^* (like L is for N). T^* is the proof that L^* is one-way dependent from Z^*. R^* is the rest — the upward linking information from the upper layers. P^* is the previous element in the chain of the parent notary. It is needed for checking the correctness of Z^*. To compute Z^*, the parent notary used the input from client and some earlier information from its chain — like the directly preceding element in the chain. The client needs the earlier linking information to verify the correctness of Z^*. The previous element exists always even in the same round — for the very first element of the round we use the initial values of the round as the previous element (G).

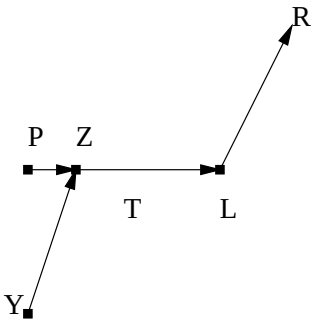

Figure 9: The important variables of one round

The protocol:

Protocol 4.3. Notarization protocol 3

1. C computes $Y = \mathrm{Sig}_C\{X, S\}$
2. $C \xrightarrow{1} N$: Y, Cert_C
3. N computes the group hash value $L = h(\ldots, P, Y, \ldots)$, this also yields Z and the new proof T
4. $N \xrightarrow{1} N^*$: L, Cert_N
5. $N \xleftarrow{2} N^*$: $\mathrm{Sig}_{N^*}\{L, \mathrm{Cert}_N\}, P^*, Z^*, L^*, T^*, R^*$

6. N compiles $R = \{\mathrm{Sig}_{N^*}\{L, \mathrm{Cert}_N\}, \mathrm{Cert}_N, P^*, Z^*, L^*, T^*, R^*\}$

7. $C \xleftarrow{2} N \quad : \quad \mathrm{Sig}_N\{Y, \mathrm{Cert}_C\}, P, Z, L, T, R$

8. C compiles its own $R' = \{\mathrm{Sig}_N\{Y, \mathrm{Cert}_C\}, \mathrm{Cert}_C, P, Z, L, T, R\}$

4.5 Temporal Verification Protocol

The input of temporal verification protocol is a pair of notarized documents, the notarized signatures of both documents and the downlink information the documents. The goal is to determine if the supposedly later document is really later than the supposedly earlier document.

The client C sends two documents X_1 and X_2 to the verifier V. It also sends R'_1, R'_2, S_1 and S_2. R'_1 and R'_2 are the upward links for X_1 and S_1 and S_2 are the downward links for X_1 and X_2.

Protocol 4.4. Temporal verification protocol 3

1. $C \xrightarrow{1} V \quad : \quad X_1, X_2, S_1, S_2, \mathrm{Sig}_{C_1}\{X_1, S_1\}, \mathrm{Sig}_{C_2}\{X_2, S_2\}, R'_1, R'_2$

Here C_1 and C_2 are the signers of X_1 and X_2. They don't have to be equal to neither each other nor C. The verification algorithm uses the following scheme:

1. The verifier checks that the documents X_1 and X_2 have correct signatures from their respective owners. The validity of the signatures is not checked — just the fact that the signatures match the documents. S_1 and S_2 are needed for this since they were used for signing and so must be used for signature verification. If any of the signatures doesn't match then we can't continue the verification and the answer is negative.

2. The verifier finds the lowest common notary of X_1 and X_2 by following the chains R'_1 and S_2 from bottom to top. We call this common notary N. If there's no common notary then the temporal order can't be determined and the answer is negative. This is the case only when the documents have been notarized in different notarization hierarchies or when some linking information is forged.

3. The verifier checks the one-way dependency from $\mathrm{Sig}\{X_1, S_1\}$ to the data item in R'_1 that corresponds to N. We name the already verified data D. At the beginning of the chain $D = \{\mathrm{Sig}_C\{X_1, S_1\}, \mathrm{Cert}_C\}$. The verifier uses iterative method. Each step uses the following scheme until it has checked the Z given by N.

   - check that $Z = h(P, D)$ where h is the hash function used to compute Z
   - check that T proves that Z was used to compute L
   - $D := \{L, \mathrm{Cert}_M\}$ where M is the current notary

   If any of these checks fail then the answer is negative. If the checks succeed then the verifier has found that the value of Z in N's chain is dependent from the document X_1.

4. The verifier checks the one-way dependency from the round summary of N (the summary is contained in S_2) to $\mathrm{Sig}\{X_2, S_2\}$. We name the already verified data D. At the beginning of the check $D = G$ where G is the downlink information that came from N. If for R' we checked from the beginning of the chain to the inside of the structure then here we take the initial value from deep inside the chain structures and iterate until we reach the outmost structure level. At each level we have the current values of G, G^*, H and S. The verifier must use the following scheme at each level:

   - check that H proves that G^* was used to compute G

   If any of these checks fail then the answer is negative. If the checks succeed then the verifier has found that the last G in the chain is one-way dependent from the G from N. That means S_2 is one-way dependent from the G in N's chain.

5. The verifier checks the notarization of $\mathrm{Sig}_{C_2}\{X_2, S_2\}$ using the information in R'_2. The scheme is exactly the same as for checking R'_1 but now we want to check until we reach the top. If there are any problems reaching the top-level notary, the answer is negative. If the notarization from top-level notary cannot be verified with the publicly known certificate of the top-level notary then the answer is negative.

   This step is needed to check that S_2 and X_2 are bound together. The notarized signature on the pair them binds the pair together.

6. The last thing to be checked is the temporal dependency in the time-stamping chain of N. We have one upward link ending in the chain of N and one downward link starting there. We want to know if the end of the upward link is there before the beginning of the downward link. For this we must use the inter-round linking scheme of N. This is the only step in the verification that may need network access because we might need to retrieve some intermediate round summaries from the notary N or from an archive-keeper. If the check fails then the answer is negative.

7. If nothing failed then the answer is positive — X_2 is later than X_1.

4.6 Signature Verification Protocol

The input of the signature verification protocol is a notarized document. The goal is to determine if the signature is valid. This is similar to the verification protocol in section 3.5.

Here C denotes again the final client — the leaf of the tree.

C sends the document X to V. It also sends $\mathrm{Sig}_C\{X\}$ and R' as defined in section 4.4. R' is essentially a chain of signatures from all the notaries in the path from the notary of C to the top of the hierarchy. S' is not important for signature verification since the proof of signature validity comes only from upward links.

Protocol 4.5. Signature verification protocol 3

1. $C \xrightarrow{1} V \quad : \quad X, \mathrm{Sig}_C\{X\}, R'$

V checks the following criteria:

1. $\mathrm{Sig}_C\{X\}(= Y)$ matches X and Cert_C
2. Auth_C (contained in Cert_C) permits this kind of signing
3. $\mathrm{Sig}_N\{Y\}$ matches Y and Cert_N
4. Auth_N (contained in Cert_N) permits this kind of signing
5. T shows that $\mathrm{Sig}_N\{Y\}$ was used to create L

6. ... (repeat last three lines for every level up to the root)
7. The certificate of the root notary matches the published and well-known one

If all these conditions are met then the signature is considered valid.

4.7 Certificate Revocation Protocol

The certificate revocation protocol is the same as in section 3.6. C sends N a signed revocation request or just calls N and tells its revocation password. N deletes the certificate of C from its database of valid certificates and doesn't respond to requests with certificate Cert_C any more.

Protocol 4.6. Certificate revocation protocol 3

1. $C \xrightarrow{1} N$: REVOKE $\mathrm{Cert}_C, \mathrm{Sig}_C\{\text{REVOKE } \mathrm{Cert}_C\}$

4.8 Addition and Deletion of Notaries

Addition of notaries is still simple as in the case of accumulated notarization — the protocol was described in section 4.2. To keep the tree efficient, new notaries should be added in a way that keeps the tree mostly in balance. A central advisor may help in choosing the parent notary for the new node.

The simple accumulated notary system allowed deleting the notaries in a clean way. Here the situation is not so brilliant — the time-stamping chains keep us from just deleting the notary and its data. The summaries of the rounds must be saved for later retrieval because the intermediate values might be needed when comparing two distant elements in the chain.

So in addition to revoking its certificate the leaving notary must give all its round summaries to some other authority that keeps the summaries available. This may be another notary but since the archival is not a key part of the proposed notary service, some other party might be the *archive-keeper*. Even more, the notaries might give all their summaries to the archive-keeper at once after the creation. In this case the archive-keepers are the only parties which serve the summaries and in such case the notaries may leave easily again.

The archive-keepers (archiving authorities) must be trusted to keep all the information available. There's no need to trust them more because they can't forge the data they serve since the linking information binds the data items together with a secure linking scheme.

4.9 Differences Between the Notaries

All the notaries may use different group hash functions and linking schemes for their internal time-stamping chains. For the clients to be able to verify the computations and time-stamping links, the notaries must put some identification of the algorithms used into the data that is computed using the algorithm. It means that the item Z in notarization protocol must carry the hash function identifier that was used to link the bits from the document into the chain; the item T must include the group hash function identifier that is used inside the rounds; L must include the identifier about the linking scheme that connects the rounds together; the item H must include the group hash identifier that it is proof for. We do not need to include the linking scheme identifier in G since this is the same as for L as far as the notary doesn't change the inter-round linking scheme. This is a reasonable assumption because changing the inter-round linking scheme would be very difficult and it's easier to just retire, ask a new certificate and use the new linking scheme in the future.

If different linking schemes and group hashes are used then the verification procedure must check whether it understands the schemes and whether it trusts the schemes. The users must understand that a verifier with more restrictive security policy may not accept proofs that some notaries produce.

4.10 The Top of the Hierarchy

The proposed system doesn't solve the problems with the top of the hierarchy. All the participants must unconditionally trust the top of the hierarchy; it is assumed that the public key of the top-level notary has been distributed to every participant.

Secure multi-party computations and threshold schemes [Rab98] can be used at the top-level notary to reduce the security risks coming from the high trust level. With these methods we can achieve that no one knows the whole secret key of the top-level notary and for the security of the root notary to

be compromised many key-piece holders must collaborate. If the number of parties in the multi-party computation is high enough and the threshold is not unreasonably low, pretty high level of security can be achieved.

The key distribution may use many broadcast channels (Internet, newspapers etc) to distribute the key to everybody. The people can get the key from several sources and trust it only when the majority of the sources agree on the key.

4.11 Cross-notarization

The proposed system does not have cross-notarization methods built into it. This doesn't mean that cross-notarization between several hierarchies is impossible.

First, the classical "brute-force" method applies — the client notarizes its document at several notaries that belong to different hierarchies (like a banking hierarchy and some national hierarchies according to the possible usage jurisdictions of the document). The internals of the protocol are not reached with this method. This method is not practical enough since the user must know before the signing which hierarchies may be needed in the future.

A slightly better cross-notarization method can be used. If we build our own notary that automatically requests notarization from several notary hierarchies then we get essentially the same that we have with the previous method — it's just slightly easier for the final client to use it. But this method gives a hint of building the hierarchy below this notary. The hierarchy should allow several heads and several tails at each level. Now if we modify the protocols to use more than one head and tail then we can make the crossing points practically everywhere. This means that with some little modifications the protocol gives us also a mechanism for some cross-notarization.

4.12 Disadvantages of Integrated Notarization

In summary, some disadvantages remain and some new disadvantages are introduced with the proposed notarization protocol. Here are some disadvantages that may be problematic in some situations:

- The protocol assumes on-line operations. This is inevitable if we want

temporal authentication of data.

- The protocols still use large amounts of data. This is different from the CRL transport problem — most of the network bandwidth is used during the signing process, the verification usually takes very little bandwidth (if any). But the protocols that are provided here may be optimized further. For example, hash values of some data items may be used in some places. I have not made these optimizations to make the scheme more understandable.

- The suitable round lengths for time-stamping and notarization may be too different and it may be hard to find a optimal round length for the notaries. Notarization needs short rounds because long rounds make the response time very long. On the contrary, time-stamping needs long rounds because the short rounds make the summaries useless in time-stamping chains since near-linear amounts of data should be searched for verification in some cases. Good inter-round linking schemes may help here ([BLS99], [Lip99]).

References

[BdM94] Josh Benaloh, Michael de Mare, *One-Way Accumulators: A Decentralized Alternative to Digital Signatures*, In Advances in Cryptology — Eurocrypt'93, LNCS 765, pp. 274-285, Springer-Verlag, Berlin, 1994.

[BCF+94] Shimshon Berkovits, Santosh Chokhani, Judith A. Furlong, Jisoo A. Geiter, Jonathan C. Guild, *Public Key Infrastructure Study: Final Report*, Produced by the MITRE Corporation for NIST, April 1994, `http://csrc.nist.gov/pki/documents/mitre.ps`

[Bul99] Ahto Buldas, *Certificate revocation, revisited*, manuscript, May 5, 1999

[BL98] Ahto Buldas, Peeter Laud, *New linking schemes for digital time-stamping*, in The 1st International Conference on Information Security and Cryptology, pages 3–14, December 1998

[BLLV98] Ahto Buldas, Peeter Laud, Helger Lipmaa, Jan Villemson, *Time-Stamping with Binary Linking Schemes*, In Advances in Cryptology — CRYPTO'98, LNCS 1462, pp. 486-501, Springer-Verlag, 1998.

[BLS99] Ahto Buldas, Helger Lipmaa, Berry Schoenmakers, *Optimally Efficient Accountable Time-Stamping*, Submitted, May 1999

[DH76] Whitfield Diffie, Martin E. Hellman, *New directions in cryptography*, in IEEE Trans. Inform. Theory, IT-22, pp. 644–654, November 1976

[HS90] Stuart Haber, W. Scott Stornetta, *How to time-stamp a digital document*, In Advances in Cryptology—CRYPTO '90, pp. 437–455, Springer-Verlag, 1991

[HS91] Stuart Haber, Wakefield Scott Stornetta, *How to Time-Stamp a Digital Document*, Journal of Cryptology, vol. 3 (2), pp 99–111, Springer-Verlag, 1991

[HFPD99] R. Housley, W. Ford, W. Polk, D. Solo, *Internet X.509 Public Key Infrastructure: Certificate and CRL Profile*, Internet RFC 2459, January 1999, `ftp://ftp.isi.edu/in-notes/rfc2459.txt`

[Jus98a] Michael K. Just, *Some Timestamping Protocol Failures*, in Proceedings of the Internet Society Symposium on Network and Distributed Security (NDSS '98).

[Jus98b] Michael K. Just, *On the Temporal Authentication of Digital Data*, Ph.D. Thesis, School of Computer Science, Carleton University, December 1998

[Lip99] Helger Lipmaa, *Secure and Efficient Time-Stamping Systems*, Ph.D. Thesis, Tartu 1999

[Mic96] Silvio Micali, *Efficient Certificate Revocation*, Laboratory of Computer Science, Massachusetts Institute of Technology, 1996, available from `ftp://ftp-pubs.lcs.mit.edu/pub/lcs-pubs/tm.outbox/MIT-LCS-TM-542b.ps.gz`

[Rab98] Tal Rabin, *A Simplified Approach to Threshold and Proactive RSA*, Advances in Cryptology: CRYPTO'98, LNCS-1294, 440-454

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the ACM, 21(2), pp. 120–126, 1978.

[Vil99] Jan Villemson, *Certificate Revocation Paradigms*, manuscript, April 15, 1999

Ajatembelduse ja notariseerimise ühendamisest

Meelis Roos

Kokkuvõte

Käesolev töö uurib viimase aja probleeme ajatembelduse ning avaliku võtme infrastruktuuride alal. Alustuseks antakse lühike ülevaade ajatembelduse tehnoloogiate arengust, et näidata uurimuste üldist suunda. Selleks tutvustatakse krüptograafiliselt seostamata ajatempleid, lineaarselt lingitud ajatempleid ning jõutakse binaarsete ja üldistatud linkimisskeemideni. Viimaste puhul on tegemist käesoleva aasta jooksvate tulemustega. Autor identifitseerib ajatemplisüsteemide juures skaleeruvusprobleemi — teadaolevad krüptograafiliselt turvalised süsteemid ei skaleeru ühest serverist ülespoole.

Avaliku võtme infrastruktuuridest (PKI) tegeldakse *de facto* standardiks oleva X.509-ga ja antakse ülevaade selle protokollistiku ühest problemaatilisest tahust — sertifikaatide tühistusnimekirjadest. Siin komistame X.509 algse variandi *off-line* ülesehitusest päritud probleemide otsa: sertifikaatide tühistusinformatsiooni antakse välja pikkade nimekirjadena perioodiliselt mingi aja tagant. Kuna nimekirjad on pikad, siis tekivad probleemid võrguliikluse mahuga jooksva informatsiooni saamiseks (iga kord tuleb värskeima info saamiseks terve nimekiri omale ära kopeerida ja teha selles nimekirjas lineaarne läbivaatus). Tühistusnimekirjade perioodilise iseloomu tõttu ei ole aegumisinfo perioodi pikkusest täpsem ajaline autentimine üldse võimalik.

PKI probleemi lahenduseks pakutakse töös (järjekordselt) radikaalset abinõu — loobuda üldse tühistusnimekirjadest ja ehitada kogu süsteem teisiti üles, arvestades seejuures tänapäevaseid nõudeid ja võimalusi. Töös pakutakse lahenduseks notariseerimist — iga juriidilist jõudu omav allkiri tuleb lasta notaril kontrollida ja kinnitada ning edaspidi piisab allkirja kontrolliks notari kinnitusest.

Esimese lahendusvariandina tuuakse ära primitiivne notariseerimisprotokoll, mis ei lahenda probleemi, kuid annab ideid edasiseks. Selle protokolli edasiarendusena pakutakse välja uus notariseerimise protokoll (*akumuleeritud notariseerimisprotokoll*), mis lubab notaritest skaleeruva ja usaldatava hierarhia moodustada. Usaldatavus garanteeritakse krüptograafiliste meetodi-

tega, nii et ka notarid ise ei saa midagi võltsida ning nende töö on täielikult kontrollitav.

Pakutud süsteem lahendab probleemi PKI-ga ning tulemus sarnaneb oma ülesehituselt ajatemplite linkimiseks kasutatavate skeemidega. Selgub, et notariseermise hierarhia realiseerib automaatselt poole ajatemplisüsteemist. Autor ehitab pakutud süsteemile juurde ka teise poole sellest süsteemist. Tulemusena saadud protokoll (*ajatembeldusega notariseerimisprotokoll*) tagab lisaks allkirjade õigsuse näitamisele ka allkirjastatavate dokumentide omavahelise ajalise autentimise. Kuna süsteem on hierarhiline ja koosneb paljudest väiksematest ajatemplisüsteemidest, siis on sellega leitud üks võimalik lahendus ka ajatemplisüsteemide skaleeruvuse probleemile.

Töö on jätkuks ajatemplialastele uuringutele Küberneetika ASis, kus ajatembeldust on uuritud paar viimast aastat. Uuringute käigus on valminud ajatempli kontseptsioon, rida artikleid ([BLLV98], [BL98], [BLS99] ja [Lip99]) ja ajatempliserveri spetsifikatsioon. Käesoleva töö autor on kirjutanud selle järgi ka ajatemplite pilootserveri. Praegune uuringute suund on ajatemplite rakendamine PKI-s. Aidatakse kaasa ka Eesti digitaaldokumentide seaduse loomisel.