DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS
29

SIZE-EFFICIENT INTERVAL TIME STAMPS

JAN VILLEMSON

TARTU 2002

DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS
29

DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS
29

SIZE-EFFICIENT INTERVAL TIME STAMPS

JAN VILLEMSON

TARTU UNIVERSITY
PRESS

Department of Mathematics, University of Tartu, Estonia

Dissertation is accepted for the commencement of the degree of Doctor of Philosophy (PhD) on April 12, 2002, by the Council of the Department of Mathematics, University of Tartu.

Opponents:

Pekka Orponen	Helsinki University of Technology Helsinki, Finland
Helger Lipmaa	Helsinki University of Technology Helsinki, Finland

Commencement will take place on June 17, 2002.

Publication of this dissertation is granted by the governmental financial support to PhD students.

Tartu Ülikooli Kirjastuse trükikoda
Tiigi 78, 50410 Tartu
Tellimus nr. 300

CONTENTS

LIST OF FIGURES

LIST OF ORIGINAL PUBLICATIONS

1. Ahto Buldas, Peeter Laud, Helger Lipmaa, Jan Villemson, Time-Stamping with Binary Linking Schemes, *Advances in Cryptology – CRYPTO '98*, Springer-Verlag 1998, pp 486-501.

2. Margus Freudenthal, Sven Heiberg, Jan Willemson, Personal Security Environment on Palm PDA, in proceedings of ACSAC 2000, December 2000, New Orleans, Louisiana, USA.

3. Arne Ansper, Ahto Buldas, Meelis Roos, Jan Willemson, Efficient long-term validation of digital signatures, *Advances in Cryptology – PKC 2001*, Springer-Verlag, LNCS 1992, pp 402-415, presented on PKC 2001, February 2001, Cheju Island, Korea.

4. Arne Ansper, Ahto Buldas, Märt Saarepera, Jan Willemson, Improving the availability of time-stamping services, *Information Security and Privacy – 6th Australasian Conference, ACISP 2001*, Springer-Verlag, LNCS 2119, pp 360-375, presented on ACISP 2001, July 11-13, 2001, Sydney, Australia.

5. Ahto Buldas, Meelis Roos, Jan Willemson, Undeniable replies for database queries, accepted to Fifth International Baltic Conference on DB and IS, June 2002, Tallinn, Estonia.

6. Kristo Heero, Uuno Puus, Jan Willemson, XML based document management in Estonian legislative system, accepted to Fifth International Baltic Conference on DB and IS, June 2002, Tallinn, Estonia.

ABSTRACT

In various applications of digital document management it is necessary to determine different parameters of documents – e.g. the format, the author or the time of creation. Determining the time can be unsuccessful since the bits of a digital document look the same regardless of their exact inception moment. Hence, in practical applications instead of fixing the digital document's *creation time* it is better to register the document at certain authority and consider the *registration time* instead. Such a process is called *time-stamping* and the corresponding authority is called *Time-Stamping Authority.*

There are a number of occasions where one-time registration of the data is enough. Registering a patent application is a good example of such a scenario. Determining the time of digital signature creation, on the other hand, differs substantially from the patent case. This is caused by the fact that digital signatures are given using private keys that should remain under the signers' sole control.

Hence, it is impossible to determine the exact moment of signing by any third party. Nevertheless, it is possible to fix two moments – one before and another after the signature creation. Using these moments we can later prove that the signature was given during some *time interval* and this form of time-stamping is called *interval time-stamping.*

The main idea of the current PhD thesis is to study interval time-stamping schemes that allow us to decrease the size of the time stamps as much as possible. While doing this, several restrictions must be taken into account, the most important one being the ability to compare creation times of the documents without help of any third party.

In the thesis, we state an explicit optimization goal, give an upper bound to the size of time stamps, find a lower estimate for this bound and construct a family of graphs approaching this (unachievable) estimate asymptotically.

The last chapter of the thesis is devoted to the questions of practical implementation of the proposed schemes. The original recurrent definition of the scheme family turns out to be unsuitable, so we will introduce an alternative description. This solution enables storing the server's internal state in a limited number of variables that can be efficiently back-upped. The description is further used to design an efficient step-by-step time-stamping algorithm.

1 INTRODUCTION

1.1 Paper-based documents

In many everyday applications there is a *document* involved to witness our behavior. Bus tickets, diplomas, promissory notes, passports, wills, student cards, driver's licenses, love-letters, stock shares, guarantee coupons, business contracts etc. are all very common and well-known examples.

There are several components that help a document to prove things and hence make it a document as we understand this notion today.

- **Contents** – this is the meaning of the document, stating that Mr. X has some rights or obligations or that he has just stated something (e.g. expressed love for Ms. Y). In other words, contents of the document shows why it was created and in all the examples above the contents were physically written on something that is called
- **Medium** – for usual documents, this is just paper or sometimes plastic; from the history we also remember people writing their messages on wood, on stone or even encoding them by making knots in ribbons.
- **Means of authentication** – a document can have no legal value if nobody is responsible for it.[1] Thus, in order to establish the person in charge, there must be something added to the document contents to enable an independent party (*judge*) to decide whether or not someone is bound to it. Mostly, a hand-written *signature* does the job, sometimes people also use fingerprints, water-marks or just three crosses attached.

It is important to note that in the case of conventional paper-based documents the contents are connected to the signature via the medium. Once a text has been typed on paper and a signature written under it, there is no way to erase either of them without leaving visible traces (at least it should be very hard).

But in digital world we see that our familiar intuition behind documents may break down. We no longer have any paper to take into our hands for

[1] Even more, the responsibility for document creation must be taken by a human. We can not say "the computer wrote it" as it is pointless to put the computer behind the bars if anything goes wrong. It is of course technically possible for a computer to create messages automatically, but there must still be a human person responsible for that.

reading, suddenly a piece of information may have many identical originals etc. It usually takes some time from people to get used to the new framework and sometimes unexpected things can happen. Let us conclude this section by a small real-world example of what happens if the conventional methods are applied to digital data management.

> *In Tartu University, Estonia, somewhere at the end of 1990s there was a regulation established concerning several aspects of bureaucracy. One part of it was talking about destroying the old and unnecessary documents. When the secretaries had collected a pile of old documents, they had to write short notes about the contents of all these documents, store those notes and then feed the documents to the shredder. But when they needed to delete some files from a computer hard disk, they first had to print those files out and feed the printouts to the shredder! And no word about actual deletion of files from hard disks!*

A lesson to be learned: changing from paper to computers really changes the notion *document*. One has to be cautious of what to say about digital documents if one has only seen the paper ones — they act differently in many important details!

1.2 Who is responsible for the document?

When dealing with paper documents, we are used to think that it is important to establish, *who wrote the document*. This information is important for contracts, wills, promissory notes etc. If a contract is signed and one of the parties breaks it, we must be able to determine *who is responsible* for the consequences. In many practical applications, finding out the creator of a document also gives us the responsible person.

On the other hand, having a digital document at hand (or in the computer), it is impossible to say who has created it, because the bits cannot be distinguished by handwriting. Hence, the best we can do is to be sure about who takes the responsibility for the document.

Technically, responsibility means potential repressions against a person, possibility to apply penalties (financial or even death penalty) if something goes wrong. Consequently, we must be able to determine the right person by the document and possibly using some additional data.

The process of binding a person to a (digital) document is called *giving a digital signature* and it is implemented via a *signature scheme*.

1. In order to be able to give digital signatures, the user A needs a *key pair* consisting of a private signature key S_A and a public verification key V_A. The private part is kept secret, whereas the public part is made available to everybody.

2. The user A can apply signature creation procedure to a document X and the private key S_A to obtain the signature $\mathrm{Sig}_A\{X\}$.

3. When the verifier B has a document X, a signature $\mathrm{Sig}_A\{X\}$ and a public key V_A, he can apply the verification procedure to them and get "Yes" or "No" as the outcome, indicating the correctness of the signature.

4. Knowing only the public key V_A, it is computationally infeasible to find its secret counterpart S_A, or even produce a valid signature of A to a new document X.

There are many signature schemes proposed, out of which RSA [RSA78] and Digital Signature Standard (DSS) [NIS00] are two of the most popular ones. You may also look at Birgit Pfitzmann's excellent PhD thesis [Pfi96] for more information on digital signature schemes.

1.3 When was the document signed?

It is not always enough to know *who* created/signed the document, but also *when* it was done. For an example, if Alice signs a promissory note and makes her private key public right after that, she can later claim that anybody could have given the signature instead of her. One possible solution to this problem is to make Alice responsible for all signatures given with her private key *until* she explicitly claims her private key compromised (*revokes* her key). Hence, before accepting Alice's promissory note, the bank should verify whether Alice has revoked her key and give her money only if the key is still valid. For instance, if *digital certificates* (the framework of stating the validity of signature keys proposed by Kohnfelder [Koh78]) are used, the bank may consult an OCSP [MAM+99] server or a Notary server [ABRW01].

Nevertheless, in order to be able to prove later (e.g. in court) that Alice's key was not revoked at the moment of signing, the bank has to present some more convincing arguments than just the claim "We checked that the signature key was valid". These extra arguments should at least say what happened *before* — was it the act of signing or the act of revocation.

One of the most widely used ideas to overcome this problem is to introduce a new party to the game. This party is responsible for attaching

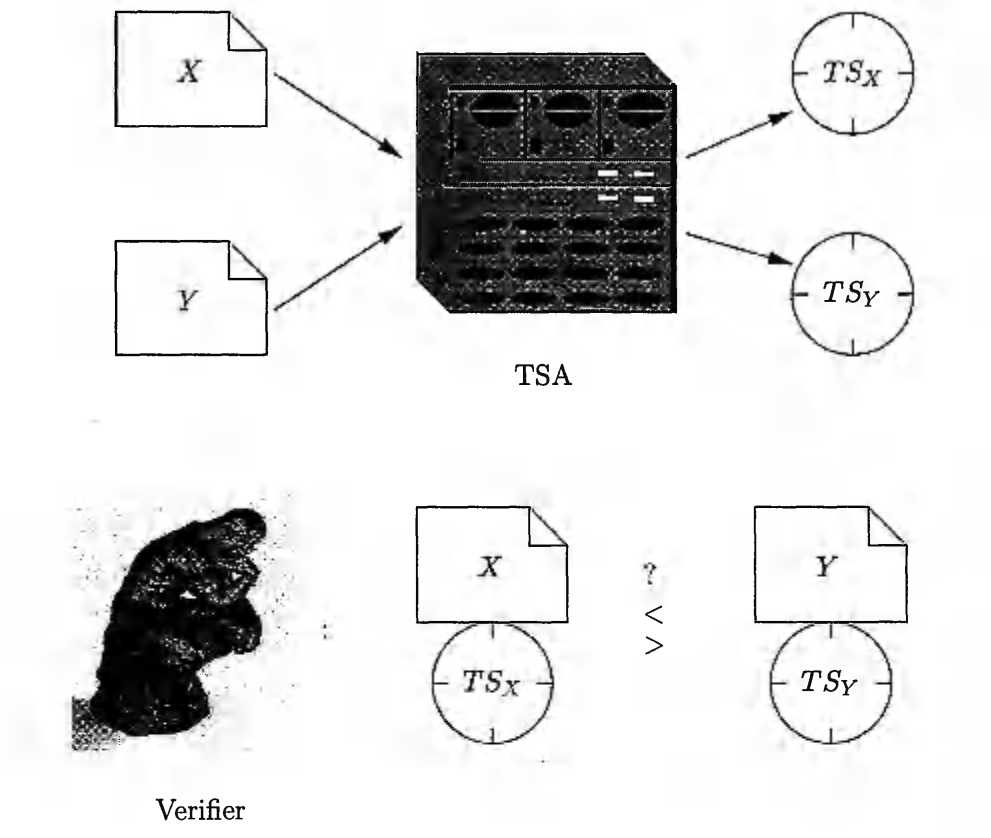

Figure 1.1: The general model of time-stamping.

time information (*time stamps*) to the documents (or to any other kind of digital data, e.g. auction bids, patent claims etc.). Because of its role, he is generally known as *Time-Stamping Authority* (TSA).

The general work model of TSA is explained in Figure 1.1. If a document X is to be time-stamped, some communication is initiated with the authority. During this communication the document is transmitted to the TSA who computes the time stamp TS_X and returns it. Of course, the whole protocol can be much more complicated than just having one pass, e.g. we might add client and/or server authentication, compute the time stamp in several parts etc.

It is not enough only to issue time stamps. We must also have means of comparing them and establishing which one was issued earlier (Figure 1.1,

below). Of course, in order to make the time-stamping system practical there are several requirements to meet.

- The protocol should not take too long to run.
- The time stamps should be as small as possible to avoid unnecessary overhead in communication and storage.
- It should not be necessary to invoke any parties (even the TSA) to the verification procedure, i.e. time stamps should be comparable off-line.

The last two conditions are in a way controversial. The feature of off-line comparison implies that time stamps must contain enough information about all the other time stamps. As we want our time-stamping system to allow potentially any number of time stamps, it is for instance very difficult to make the time stamps having constant size. Hence, a tradeoff has to be found and the question how small can off-line comparable time stamps be needs an answer. Answering this question in case of time stamp for digital signatures is the core of this thesis, but before reaching the core we still need to discuss some details.

1.4 Absolute and linkage-based time stamps

The standard approach when building a time-stamping framework is to give the TSA a (generally trusted) clock and let him sign the request together with the clock's state at the moment the request is received. For example, the IETF PKIX time-stamping standard [ACPZ01] is based on this idea. Still, such an *absolute time-stamping* approach suffers from several drawbacks.

1. The TSA is completely trusted. Among other things, this means that the TSA can attach any time (not necessarily the correct one) to the requests.
2. As the TSA signs its statements, the compromise of its private key also becomes a problematic issue.
3. In a way, attaching absolute time to the documents gives too much information. We are mostly not interested in exact time moments, but rather in establishing the temporal relationship (earlier/later) of several events. This was the case with Alice's signature and key revocation. It also is when we need to compare patent registrations, or to make sure that a job-application arrived before the deadline etc.

The last consideration leads to the question, whether it is possible to establish temporal relationships by some other (less-demanding) means, and if so, perhaps it could also be possible to reduce the trust requirements of the TSA.

Such a framework can be built by using one-way functions to create undeniable causal relationships between digital events rather than assigning physical time to them. As a result, all the time-stamped items form something like a chain of links where it is impossible to insert some new elements or to delete any old ones. Because of this analogy the described approach is called *linkage-based time-stamping*. It was first proposed in 1990 in [HS91] and later improved in [BLLV98].

1.5 Two scenarios of time-stamping

Before starting to create a new time-stamping scheme we must analyze the requirements the scheme has to meet. First, we will discuss the possible settings where it makes sense to apply time-stamping. On a very general level we can distinguish the following two basic scenarios.

First scenario: Who gets it first? In this scenario, there are several participants who are interested in the same resource and they need to be (among) the first ones to get it. There are many common examples of this scenario.

- When several scientists make the same invention, only the first one to reach the patent bureau is the one who can claim the rights for the invention.
- If flights are booked internationally, the airline companies tend to double-book some seats as their experience shows that many bookings are often canceled. Still, from time to time, it happens that some flights are over-booked and in this case only the first bookers should get the seats.
- Temporal ordering of the requests can be applied for several kinds of auctions, see [PSST01, RG95] for discussion.

In order to prove later to the patent-interested scientists that some of their competitors were not favored unfairly, time-stamping is a natural tool to use. It is also quite easy to organize time-stamping in cases like the ones above as all the clients are obliged to express their interest directly anyway, by pressing a button or running to the travel agent. Hence, it is enough

for the TSA to record the moment of the interest expression and to issue a time stamp for that moment.

Second scenario: When did the cat die? Recall the Schrödinger's famous mental cat experiment [Sch35] (the English translation used here is due to John D. Trimmer [Tri80]):

> *A cat is penned up in a steel chamber, along with the following device (which must be secured against direct interference by the cat): in a Geiger counter there is a tiny bit of radioactive substance, so small, that perhaps in the course of the hour one of the atoms decays, but also, with equal probability, perhaps none; if it happens, the counter tube discharges and through a relay releases a hammer which shatters a small flask of hydrocyanic acid. If one has left this entire system to itself for an hour, one would say that the cat still lives if meanwhile no atom has decayed.*

For us, the important question arising from this experiment is: if we open the chamber and see a dead cat then how can we tell the exact moment of death? The sad truth stated by Schrödinger is – we cannot.

A similar situation can be observed in the computer world if we need to determine the time of some private digital action. One very important example – signing – was already presented above. Note that the situation of signing is substantially different from registering patents. As we saw before, a scientist interested in the honor of invention can (and has to) show his interest explicitly and publicly. Signing, on the other hand, involves application of a private signature key that is known only to the signer and to no-one else. Hence, no-one except for the signer knows the exact moment when the signature was created.

So what can we do if we still need to determine the time somehow? The answer is simple – if you cannot do it exactly, try to be as precise as possible and prove that the event took place during some *time interval.* For the Schrödinger's cat, this means saying that the cat died during the hour when the chamber was closed; for digital signatures the very same approach applies – if we can prove that a signature was created during some (relatively short) time interval, one can be reasonably satisfied with the result. The time stamps used for the proof are called *interval time stamps* from now on.

Whereas the first scenario is pretty well studied [HS91, BLLV98, BL98, BLS00, Lip99], the second one has arisen only recently. Still, being applicable for time-stamping digital signatures, it is by no means less important than the patent scenario. There are also other possible applications of interval time stamps. In principle, any computation that is carried out outside of the direct sight of the TSA can be a subject to it. For example, computation of message authentication codes (MACs, see [MvOV97], Section 9.5) also involves usage of secret keys and hence interval time stamps are to be used. Another interesting application of this approach is time-stamping other TSA's time stamps. This way, it is possible to create dependencies between the "histories" "written" by different TSAs. These dependencies can be used to

- synchronize the actions of TSAs and make items in different "histories" comparable with each other; and
- increase the reliability and availability of the TSAs: when one TSA is temporarily down, the other one still retains the continuity of the first one's work (see [ABSW01] for a more detailed discussion on availability issues).

1.6 Objectives and outline of the thesis

The basic motivation of this thesis was already stated in Section 1.3. Adding the results of the discussions from Sections 1.4 and 1.5, we formulate the following central problem of the thesis.

> Find a linkage-based time stamping scheme that provides as small interval time stamps as possible and enables off-line comparison.

When solving this problem we will mostly concentrate on the mathematical side and postpone the discussion about practical implementations to the end. Still, this discussion is by no means less substantial than the rest of the thesis as the work of the TSA must also be efficient and reliable. Hence, when proposing some new schemes, one must also ask how good algorithms can be designed based on mathematical descriptions of the schemes.

The rest of the thesis is devoted to solving these two problems and is organized as shown below. Several results presented in the thesis have not yet been published on any conference nor in any journal, but rather in a

series of manuscripts and technical reports. The references can be found below as well.

- Chapter 2 gives technical background and a brief historical overview necessary to understand the rest of the thesis.
- Chapter 3 presents a general framework of linking schemes together with detailed technical descriptions. Size-efficiency of the previous best-known time-stamping system is considered for interval time stamps and improved by 25%. This result was first obtained by Buldas and Willemson and described in manuscript [BW01a].
- Chapter 4 presents a new family of linking schemes and proves its asymptotical optimality with respect to an upper bound for the size of time stamps. The new scheme family was defined first in [BW01a] and further analyzed in [BW01b] by Buldas and Willemson. The optimal schemes were first found by the author of the thesis in [Wil01b].
- Chapter 5 discusses the restrictions that are set on the TSA's server that uses new linking schemes. We give an efficient and reliable algorithm for generating the schemes on the fly. The algorithm was originally described by the author in [Wil01a].
- Chapter 6 ends the thesis and draws some philosophical conclusions.

2 STATE OF THE ART

2.1 Hash functions

In order to build a linkage-based time-stamping scheme we use a *collision resistant hash function* (see [MvOV97], chapter 9)[1], i.e. a function h such that it

- inputs bit-strings of arbitrary length and outputs bit-strings of fixed length k, i.e. $h : \{0,1\}^* \to \{0,1\}^k$;
- works efficiently (i.e. for any x, $h(x)$ is easy to compute);
- is *collision resistant*: it is computationally infeasible to find inputs $x \neq x'$ such that $h(x) = h(x')$.

It can be easily seen (e.g. [Sti95], chapter 7) that (under some natural extra conditions) collision resistant hash functions also have the next desirable properties:

- *preimage resistance*: for essentially all outputs $y \in \{0,1\}^k$ it is infeasible to find $x \in \{0,1\}^*$ such that $h(x) = y$;
- *2nd preimage resistance*: for given $x \in \{0,1\}^*$, it is infeasible to find $x' \neq x$ such that $h(x) = h(x')$.

Whereas the question of existence of such functions is still open, several candidates have been tailored and they are believed to be good enough for practical use. SHA-1 [NIS95] together with its improvements SHA-256, SHA-384 and SHA-512 [NIS01] are the most popular ones at the time of this writing.

Later on we will extensively use the notation $h(x_1, x_2, \ldots, x_l)$ and by that we mean the value of the function h on some predefined data structure from where all the bit-strings $x_1, x_2, \ldots, x_l$ can be restored. One might e.g. use concatenation of the strings or some container format.

[1] As the idea of the current thesis is not to make a deep contribution into the theory of hash functions, our definition used here is rather informal and intuitive. For more detailed discussion we refer the reader to [Pre93].

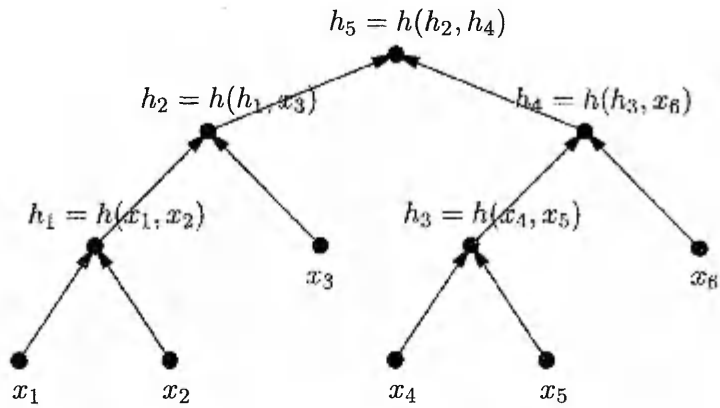

Figure 2.1: Example of Merkle's authentication tree.

2.2 Proving dependencies between the data items

In what follows, we will consider a process where outputs of some hash computations are used as inputs to others. This way we can make one output value dependent on many input values and prove this dependence by exposing some of the inputs or intermediate hash values. For instance, if we are given x_1, x_2 such that

$$h(x_1, h(x_2, x_3)) = y \tag{2.1}$$

then we say that y *depends* on the inputs x_1, x_2, x_3 in the sense that there is no other way to obtain y as a result of some hash computations than computing it by formula (2.1) (because otherwise we should be able to find second preimages to the function h). Hence, in order to prove that y depends on x_2, it is sufficient to show the additional values x_1 and x_3 so that anyone can compute $h_1 = h(x_2, x_3)$ and verify that $y = h(x_1, h_1)$.

Such a reasoning can be generalized to quite complicated data structures, for instance to binary trees as done by Merkle [Mer80]. Figure 2.1 presents an example of resulting *Merkle's authentication tree.*

Similar to the above, for all the leaves labeled $x_1, \ldots, x_6$ it is possible to prove that h_5 depends on them. For instance, if the dependence of h_5 on x_3 is to be proven, one may add the vertices h_1 and h_4 and the verifier may compute $h_2 = h(h_1, x_3)$ and $h_5 = h(h_2, h_4)$.

Hence, if h_5 is published in authentic and undeniable way, the presence of the leaves $x_1, \ldots, x_6$ at the time of forming the Merkle's authentication tree can not later be denied (even by the party who formed the tree). If we consider $x_1, \ldots, x_6$ to be records in some database $\mathcal{D}$, then h_5 is the *digest* of

$\mathcal{D}$ and the proof methodology described above can be used to prove whether for any particular x_i the condition $x_i \in \mathcal{D}$ or the condition $x_i \notin \mathcal{D}$ holds. We refer the reader to [BLL00, BRW02] for more details.

2.3 Linkage-based time-stamping

As noted in Section 1.3, there are two security drawbacks in absolute-time based time-stamping: the need to trust the TSA and TSA's potential key compromise. Hence, in order to avoid these problems, the TSA should have tools for time-stamping such that

1) he is not able to recompute his statements afterwards; and

2) his statements do not depend on any secret information.

It turns out that the cryptographic hash functions described in Section 2.1 can be successfully applied in order to meet the requirements above.

This idea was first proposed by Haber and Stornetta in [HS91] who introduced *linking schemes*. They compare a linking scheme with a lab notebook the entries of which are filled one after another and the sewn-in pages of which make the record hard to tamper with.

The approach of Haber and Stornetta is (being a bit simplified) the following. Let us have a collision resistant hash function h and let x_n be the next time-stamping request (later also called an *item*). The time stamp for x_n will be

$$(x_n, L_n),$$

where L_n is the *linking information* defined as

$$L_n = (x_{n-1}, h(L_{n-1})).$$

Thus one-way dependencies are created between the linking information strings L_n and through them also between the items x_n, allowing us to say that x_n was time-stamped *later* than x_{n-1}. As no-one knows how to compute second preimages for the hash function, even the TSA can not alter the time stamps after they are issued. Of course he can try to delay some time-stamps, but if he delays too much he will be caught on cheating. Because the linking information items form a linear chain, such scheme is called *linear*; an example is depicted in Figure 2.2.

Though reliable in the sense of security, the linear linking scheme of [HS91] is very impractical for two reasons.

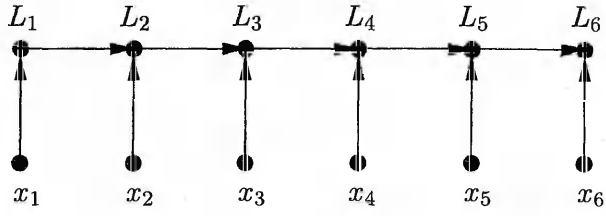

Figure 2.2: Example of linear linking scheme.

- Every time it is necessary to establish the temporal relationship between two items, the verifier must recompute the whole chain between them and this requires a lots of time if the items are far apart.
- For the verification procedure it is necessary to have all the intermediate items available as well. They can be kept in a central server, at the verifier or anywhere else, but the required storage space increases linearly in time anyway. Besides, if for some reason the storing server becomes unavailable, time stamp dependencies can not be verified anymore. Once again, we refer the reader more interested in the availability issues to [ABSW01].

2.4 Off-line comparability

By *off-line comparability* we mean the property of the time-stamping scheme to provide such time stamps that can be compared by the verifier without connecting to any other parties, i.e. based on the time stamps only.

Is it possible to achieve this property? The answer is affirmative, as it can be seen from the following naïve time-stamping scheme:

- time stamp for the request x_i is the set $T_i = \{x_1, x_2, \ldots, x_{i-1}\}$,[2]
- if it is necessary to compare the time stamps T_i and T_j of the items x_i and x_j, respectively, find out whether $x_i \in T_j$ or $x_j \in T_i$.

Another scheme providing off-line comparable time stamps, but also linear time stamp size was proposed by Pinto and Freitas in [PF96].

There have been several attempts to decrease the size estimate for time stamps based on Merkle's authentication trees, e.g. Benaloh and de Mare

[2]The time stamp T_i may also be signed by the TSA in order to achieve authentication and non-repudiation. In this case we also need off-line signature verification as done e.g. in [ABRW01].

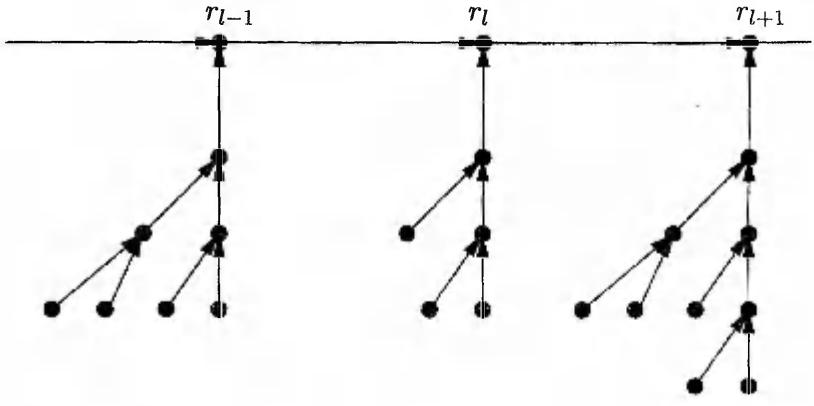

Figure 2.3: Example of linking scheme of [BHS93].

[BdM91] and Haber, Stornetta et. al. [BHS93, HS97]. We consider here briefly the scheme of Haber and Stornetta that lies on the foundation of Surety Digital Notary and Timestamping Service [Sur].

Haber and Stornetta divide the work process of a time-stamping server into *rounds*. All the items $x_1^l, \ldots, x_{k_l}^l$ obtained during the round l are used as leaves for a Merkle's authentication tree. As explained in Section 2.2, the tree's root value r_l depends in undeniable way on all the items x_i^l and this dependence can later be proven by exposing some additional items. The number of extra items needed is logarithmic in k_l. In order to create dependencies between the root nodes of different rounds, linear linking is used. An example of the Haber-Stornetta scheme is depicted in Figure 2.3.

Note that the items inside one round are actually not ordered in the Haber-Stornetta scheme. In practice, we may accept incomparability of two time-stamps, if they are close enough in time. This implies the need to make the duration of one round short enough in the Haber-Stornetta scheme but doing so we loose in logarithmic efficiency provided by the Merkle's authentication trees used inside the rounds.

The first time-stamping scheme providing both logarithmic time stamp sizes and undeniable linear ordering of the items was proposed by Buldas, Laud, Lipmaa, and Willemson in [BLLV98]. Their basic idea was to link a new item to two older ones: the previous item and a specifically selected item from (possibly very distant) past. Because of this property these schemes are called *binary linking schemes*. An example of [BLLV98] scheme is depicted in Figure 2.4

The research on size-optimal linking schemes was continued by Bul-

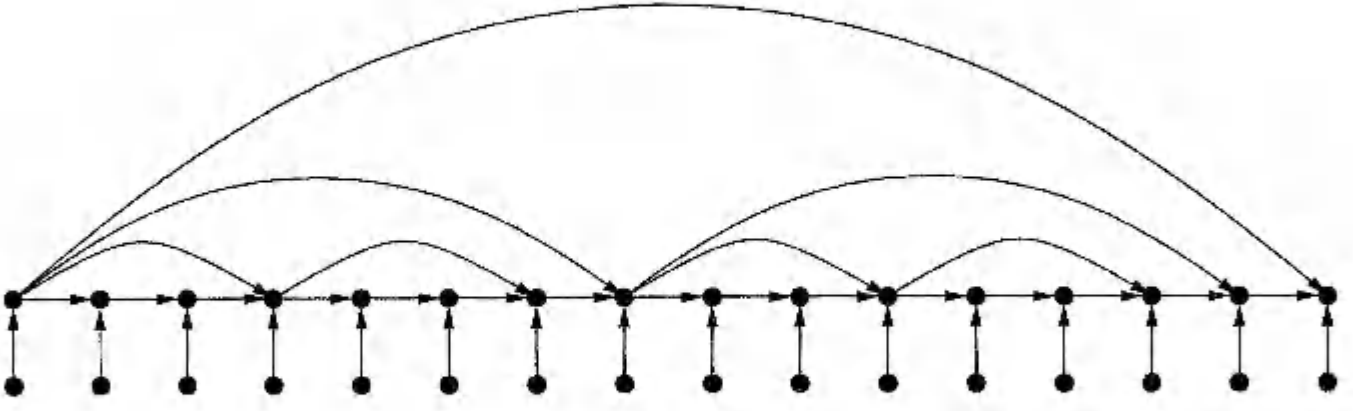

Figure 2.4: Example of linking scheme of [BLLV98].

das, Laud, Lipmaa and Schoenmakers [BL98, BLS00]. The Buldas-Lipmaa-Schoenmakers (BLS-) scheme [BLS00] was proven to give size-optimal time stamps for the patent scenario, but in this thesis we show that for interval time-stamps more efficient solutions can be given. We present a new family of schemes based on unbalanced trees which reduces the size of time stamps about 28% compared to the BLS-scheme. We will also prove that with respect to the best currently known estimates for the time-stamp sizes, this family of schemes is asymptotically optimal.

3 LINKING SCHEMES

3.1 Basic requirements

3.1.1 General framework

In Chapter 2, we saw how several graphs[1] (e.g. chains and trees) can be used to create dependencies between different data items. In this chapter, we consider the general case and assume a rooted directed acyclic graph (with arcs heading towards the root) as a basis of our linkage-based time-stamping schemes.

The items to be time-stamped are represented as nodes with in-valency 0 (by an analogy with trees they are also called *leaves* in this thesis) and arcs refer to hash computations performed using a predefined hash function h. In order to make statements about temporal relationships between different items, we also assume that the leaves of the graph are linearly ordered. There are $n!$ possible orders for a graph with n leaves and not necessarily all of them give rise to an equally good linking scheme. Hence, specifying the order of leaves plays an important role in scheme construction.

Figure 3.1 shows a simple linking scheme with time-stamped items x_1, x_2, x_3, x_4 and with hash values $h_1 = h(x_2, x_3), h_2 = h(x_1, h_1), h_3 = h(h_1, x_3), \ldots$.

Based on such a scheme the time-stamping server works as follows. The server's work is divided into a sequence of steps. At each step i

- a new item x_i is obtained;
- several hash computations are performed on x_i and previously stored values;
- for the next steps some old and some newly computed values are stored.

An example of computations carried out on the graph of Figure 3.1 can be seen in Table 3.1. Note that the set of values to compute and to store is not necessarily uniquely determined by the graph. For example, on step 3 we could also compute the values h_3, h_4, h_5 and store the value h_5 only.

[1] The current thesis relies on graph theory quite heavily. It was the author's choice not to include an introductory chapter about graphs into the thesis as there are many good resources available in literature. A reader in need for more background should probably start from some classical books like [Chr75] or [Har69].

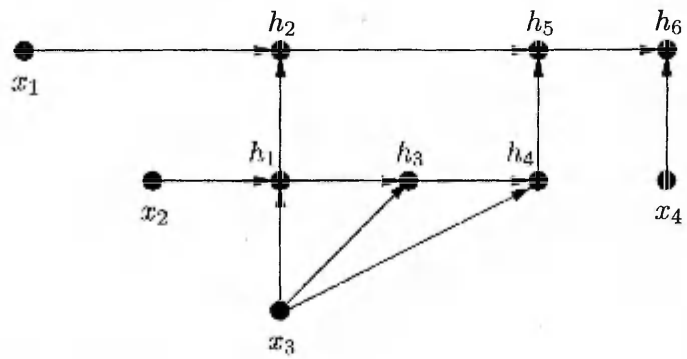

Figure 3.1: A simple example of time-stamping scheme.

Step	Input	Compute	Store
1	x_1		x_1
2	x_2		x_1, x_2
3	x_3	$h_1 = h(x_2, x_3)$, $h_2 = h(x_1, h_1)$	x_3, h_1, h_2
4	x_4	$h_3 = h(h_1, x_3)$, $h_4 = h(h_3, x_3)$, $h_5 = h(h_2, h_4)$, $h_6 = h(h_5, x_4)$	h_6

Table 3.1: A simple example of time-stamping computations.

In what follows, we will use the labels x_i and h_j in two different meanings. First, they denote the actual values attached to the vertices and hence we can perform hash computations using the values x_i and h_j as arguments. Second, we will usually speak about the vertices x_i and h_j using the labels as vertex references just like it is a general custom in graph theory.

Like in Chapter 2, we are able to prove one-way dependencies between several items, e.g. we can say that h_6 depends on x_2. As a proof of such a statement, it is enough to give some extra items required to repeat the hash computations that lead from x_2 to h_6. For example, we can compute that

$$\begin{aligned} h(x_2, x_3) &= h_1, \\ h(h_1, x_3) &= h_3, \\ h(h_3, x_3) &= h_4, \\ h(h_2, h_4) &= h_5, \\ h(h_5, x_4) &= h_6, \end{aligned}$$

and hence we may present the set of items $\{x_3, h_2, x_4\}$ (sometimes called *time certificate*) as a proof. Note that these computations are performed following the directed path

$$x_2 \to h_1 \to h_3 \to h_4 \to h_5 \to h_6. \tag{3.1}$$

Such a path will be called an *authentication path*. Later we will also need the arcs that are not parts of the authentication path but participate in the computation process. E.g., for the path (3.1) in Figure 3.1 the necessary additional arcs are

$$(x_3, h_1), (x_3, h_3), (x_3, h_4), (h_2, h_5), (x_4, h_6).$$

These arcs are called *authentication path support arcs*.

At the same time it is also possible to verify the necessary dependence by computing

$$\begin{aligned} h(x_2, x_3) &= h_1, \\ h(x_1, h_1) &= h_2, \\ h(h_2, h_4) &= h_5, \\ h(h_5, x_4) &= h_6, \end{aligned}$$

where the time certificate $\{x_3, x_1, h_4, x_4\}$ is required for proof.

Later we will see that time certificate forms an important part of a time-stamp. As the main objective of the thesis stated in Section 1.6 is

to decrease the size time stamps, we are also interested in reducing the size of time certificates. The minimal set (in the sense of cardinality) of extra nodes needed to prove the dependence of node y on node x is called *authenticator* of node x (relative to node y) and is denoted by $\mathsf{Auth}(x, y)$.

In order to be precise we also give a formal definition of this notion. First we define the operation of set hash.

Definition 1 *Let G be a time-stamping graph and h be the hash function in use. Let $K \subseteq V(G)$. Then $\mathcal{SH}_h$ is a* set hash operation *that works as follows:*

$$\mathcal{SH}_h : 2^{V(G)} \to 2^{V(G)} : K \mapsto K \cup \{x = h(x_1, \ldots, x_k) : x_1, \ldots, x_k \in K;\ (x_1, x), \ldots, (x_k, x) \in E(G)\}.$$

It is natural to denote $\mathcal{SH}_h^1(K) := \mathcal{SH}_h(K)$, $\mathcal{SH}_h^2(K) := \mathcal{SH}_h(\mathcal{SH}_h^1(K))$, etc. As G is a finite graph, for some natural number i it must happen that $\mathcal{SH}_h^i(K) = \mathcal{SH}_h^{i+1}(K)$. This set will be denoted as $\mathcal{SH}_h^*(K)$.

Now we are ready to define what it means to be able to prove dependencies.

Definition 2 *A subset K of $V(G)$ is called a* proof set *(proving the dependence of y on x) if*

1) $y \notin \mathcal{SH}_h^*(K)$;

2) $y \in \mathcal{SH}_h^*(K \cup \{x\})$.

One of the proof sets (chosen in some way) having the minimal cardinality is called authenticator *and is denoted as* $\mathsf{Auth}(x, y)$.

Note that the set $\mathsf{Auth}(x, y)$ is not necessarily unique as there may be several proof sets of minimal cardinality. For example, we may have both $\mathsf{Auth}(h_1, h_5) = \{x_3, h_2\}$ and $\mathsf{Auth}(h_1, h_5) = \{x_1, h_4\}$ for the graph in Figure 3.1. Later on, we should be careful not to cause misinterpretations based on this non-uniqueness. As we will mostly be interested in the cardinality $|\mathsf{Auth}(x, y)|$ only, this is not going to be too difficult.

As the notion of authentication path was important in the first informal description of authenticators, it is interesting to ask, how this notion relates to Definition 2. In order to answer this question, we first prove the following lemma.

Lemma 1 *If K is a proof set proving the dependence of y on x, then either $x = y$ or there exists a predecessor y' of y such that K is also a proof set proving the dependence of y' on x.*

Proof. If $x = y$ then we are done. If x is a predecessor of y we are also done by Definition 2.

Otherwise, consider all the predecessors $y_1, y_2, \ldots, y_k$ of y. As $y \in \mathcal{SH}_h^*(K \cup \{x\})$ then by Definition 1, for every index i it holds that $y_i \in \mathcal{SH}_h^*(K \cup \{x\})$. On the other hand, if for every index i it would hold that $y_i \in \mathcal{SH}_h^*(K)$, it would also mean that $y \in \mathcal{SH}_h^*(K)$ which is not possible. Hence, for some predecessor y_{i_0} the condition $y_i \notin \mathcal{SH}_h^*(K)$ is satisfied. Choosing $y' = y_{i_0}$ concludes the proof. □

Now we can carry on this process for several times: find a predecessor y' of y, then a predecessor y'' of y', etc. As the graph G is finite and acyclic, eventually we must arrive at the vertex x, obtaining hence the path from x to y.

It is generally not the case that for any two vertices there exists a dependence one way or another. The root node r ($r = h_6$ in Figure 3.1) is an exception: it depends on any other node and this way the whole "history" of the scheme is captured into the root. This way we may say that the root is younger than all the other items in the scheme, but we would also like to compare the items inside the scheme as well.

For that purpose, we need to keep track of the "history" throughout the formation of the scheme: after a new item x_j is added for time-stamping, we perform some hash computations and output a set of items capturing one-way information about all the items time-stamped this far. We give the following definition.

Definition 3 *Set $H_j \subseteq V(G)$ is called a* history set *(for the item x_j) if*

1) $\forall i \leq j$ $\exists y \in H_j$ such that y depends on (or is equal to) x_i;

2) every $y \in H_j$ can be computed from the elements $x_1, x_2, \ldots, x_j$.

Note that the choice of the set H_j is generally not unique. For example, in Figure 3.1 we may take $H_3 = \{x_3, h_1, h_2\}$ or $H_3 = \{h_2, h_4\}$ or even $H_3 = \{h_5\}$.

Next to the authenticators, the sets H_j form another important part of the time-stamps, hence we are interested in minimizing the number of their elements as well.

Definition 4 *The set history set H_j having the smallest possible cardinality is called* freshness token *and is denoted as* FT_j.

Just as it was the case with the authenticators, we must be careful when operating with the sets FT_j as they are not uniquely determined. Being mostly interested in the cardinalities only, this will not be a big problem.

Note that the definition of authenticators does not depend on the order of leaves, but the definition of freshness tokens does.

3.1.2 Interval time-stamping

Recall now our original task stated in Chapter 1. We need to prove that some action C (e.g. creation of a digital signature) took place between two events (which are not necessarily time moments, but can also be linking events) t_1 and t_2. Such a proof must clearly consist of two parts:

a) proof that C happened *after* t_1; and

b) proof that C happened *before* t_2.

Of course, in order for our time-stamping system to work properly, the following transitivity-resembling condition has to hold as well:

c) if it is proven that C happened before t and D happened after t then it is possible to prove that C happened before D.

Let the TSA have reached the state where the next item would be x_{j+1} and the user A needs to sign a document X together with interval time stamp. Then A needs some additional information that for any item x_i, $i \leq j$ lets him to prove that x_i occurred before the signature. Hence, this additional information must depend on all the previous items x_i — and freshness token FT_j is sufficient for this purpose. The first steps of obtaining a time stamp look like as follows[2]:

1. $A \rightarrow TSA$: request for the freshness token
2. $TSA \rightarrow A$: FT_j
3. A: computes $\sigma = \mathrm{Sig}_A\{X, \mathsf{FT}_j\}$

Now the signature σ depends in one-way fashion on all the previous requests x_i. How can we give an upper bound to the time moment of signing? This can be done simply by letting the signature to be the next time-stamping item x_k:

4. $A \rightarrow TSA$: σ
5. TSA: adds $x_k = \sigma$ to the linking scheme, computes FT_k etc.

[2]Here we use standard cryptographic protocol syntax where the expression $A \rightarrow B : X$ means that the party A sends the party B message X and the expression $A : Z$ means that the party A takes action Z.

By the condition c) above, in future it may of course be necessary to prove that all the later freshness tokens FT_l, $l > k$ depend on x_k. This holds also for the very last freshness token $\mathsf{FT}_n = \{r\}$. The smallest proof of dependence of the root r on x_k is given by the authenticator $\mathsf{Auth}(x_k, r)$; later on, this set will also be called *existence token* (for x_k)[3]. Clearly, the existence token cannot be issued before the whole graph is completed. Therefore, the time-stamping procedure is finished as follows:

6. TSA: completes the scheme by computing the root value r;

7. $TSA \to A$: $\mathsf{Auth}(x_k, r)$.

But what about the other freshness tokens FT_l, $r > l > k$? Do we need special authenticators for all of them? This would clearly be too resource-consuming and hence we state a much simpler requirement. Namely, we require that the very same authenticator $\mathsf{Auth}(x_k, r)$ should be enough for proving all the other necessary dependencies as well:

$$\forall k < l \quad \exists y \in \mathsf{FT}_l : \quad \mathsf{Auth}(x_k, y) \subseteq \mathsf{Auth}(x_k, r). \tag{3.2}$$

The next theorem shows an important class of graphs that satisfy this requirement.

Theorem 1 *For any tree T with linearly ordered leaves the condition (3.2) holds.*

Proof. Let the leaves of T be ordered as $x_1, x_2, \ldots, x_n$ and let x_k be an arbitrary leaf. As T is a tree, there exists the unique authentication path

$$x_k \to h_1 \to h_2 \to \ldots \to r, \tag{3.3}$$

and also the authenticator $\mathsf{Auth}(x_k, r)$ is unique. Even more, for any vertex v on the authentication path (3.3) it holds that

$$\mathsf{Auth}(x_k, r) = \mathsf{Auth}(x_k, v) \dot{\cup} \mathsf{Auth}(v, r)$$

(where $\dot{\cup}$ denotes disjoint union) which implies

$$\mathsf{Auth}(x_k, v) \subseteq \mathsf{Auth}(x_k, r).$$

[3]It was suggested to the author by several readers that a notation symmetric to FT_i should be used for this notion as well; so it could be something like ET_i or $\mathsf{ET}(x_i)$. Still, this notation was not accepted in the current thesis as by the author's opinion this would cause more loss than gain in understandability of Chapter 4.

Hence the theorem is proven if we can prove that for any freshness token FT_l $(l > k)$ there is a vertex $y \in \mathsf{FT}_l$ that belongs to the authentication path (3.3). But this is a direct implication of Definitions 3 and 4. □

Running a bit ahead, we can say that all the particular time-stamping schemes that will be proposed in the current thesis belong to the class of graphs described in Theorem 1. In what follows, we will not refer to the theorem explicitly but keep it in mind every time a new scheme is constructed.

After doing all the work above, we are finally ready to define the notion of interval time stamp.

Definition 5 *Let G be a time-stamping scheme with leaves $x_1, \ldots, x_n$ and $1 \leq i < j \leq n$.* Interval time stamp *for the interval $[i, j]$ is the pair*

$$(\mathsf{FT}_i, \mathsf{Auth}(x_j, r)).$$

3.2 Optimization goal

When time stamps are used to establish relationships between digital signatures, it is convenient to have the time stamps attached to the signatures. Still, we do not want to add too much storage overhead because of the time stamps. Hence, it is important to reduce the size of time stamps as much as possible and this is the main goal of the current thesis.

In Section 3.1, we saw that interval time stamps consist of two parts – freshness token FT_i and existence token $\mathsf{Auth}(x_j, r)$. In order to estimate the size of the whole time stamp, we will use the following definitions.

Definition 6 *By the* width *of the computation graph G we mean the value*

$$W(G) = \max_{i=1,\ldots,n} |\mathsf{FT}_i|.$$

Definition 7 *By the* depth *of the computation graph G we mean the value*

$$D(G) = \max_{i=1,\ldots,n} |\mathsf{Auth}(x_i, r)|.$$

It is clear that $W(G)$ and $D(G)$ are the upper bounds for the sizes of freshness and existence tokens, respectively.

It may happen that the freshness and existence tokens for some digital signature have some elements in common, so we conclude that the size of time stamps is upper bounded by the value

$$W(G) + D(G),$$

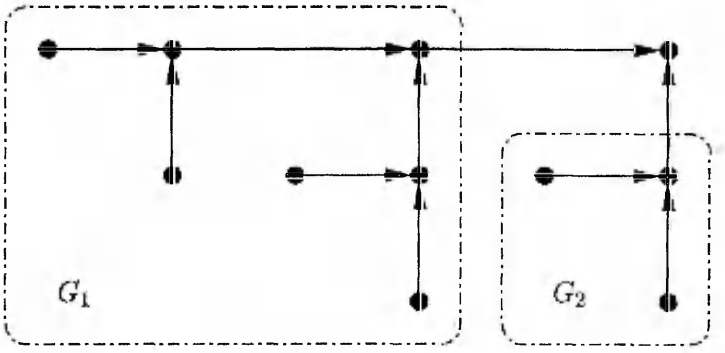

Figure 3.2: The construction $G_1 \otimes G_2$.

but this estimate is not necessarily sharp. It is an interesting open question to obtain exact bounds for the size of time stamps.

As all the currently known efficient linking schemes provide time-stamp sizes logarithmic in the number of items, we are interested in comparing this value to $\log_2 \|G\|$ where $\|G\|$ denotes the number of leaves of the graph G. Hence, in what follows we will be optimizing the following quantity:

$$\beta(G) = \frac{W(G) + D(G)}{\log_2 \|G\|}. \tag{3.4}$$

3.3 Composition of linking schemes

In the construction of our new schemes we need the following composition operation which is similar to that proposed in [BL98].

Definition 8 *The graph with one vertex and no arcs is I.*

Definition 9 *Let G_1 and G_2 be two rooted directed acyclic graphs with root vertices r_1 and r_2, respectively. Then by $G_1 \otimes G_2$ we denote the tree with vertex set $V(G_1 \otimes G_2) = V(G_1) \cup V(G_2) \cup \{r\}$ and arc set $E(G_1 \otimes G_2) = E(G_1) \cup E(G_2) \cup \{(r_1, r), (r_2, r)\}$, where r is a new vertex. The subgraphs G_1 and G_2 will be called* left *and* right subgraphs, *respectively.*

The construction $G_1 \otimes G_2$ is depicted in Figure 3.2.

It is clear that starting from the tree I and applying this construction recursively, we obtain only binary trees, and even more, every rooted directed binary tree can be constructed this way. In order to use these trees as time-stamping schemes, a linear order has to be defined on their leaf set (see Subsection 3.1.1).

3.4 Topologically sorted binary trees

Definition 10 *We say that the binary tree T is* topologically sorted *if for every non-leaf node one of its children is marked as left and the other one as right child.*

Note that this definition induces a natural linear order (which we will also call topological) for all the leaves of the topologically sorted binary tree. This order can be formalized in the following way.

1. Let the root be labeled by the empty string λ.
2. For every vertex labeled by a string σ let its left child be labeled by the string σL and the right child by the string σR.
3. Order the leaves into the lexicographic order of their labels (note that L precedes R in the alphabet).

It is clear that the leaves of all binary trees can be topologically sorted by defining the right and the left children for every inner node in some way. In what follows, we will assume such an order from the leaves of $\otimes$-constructed trees, if not otherwise explicitly stated.

For topologically sorted binary trees the following lemma holds.

Lemma 2 *Let T be a topologically sorted binary tree and $T = T_1 \otimes T_2$ (such a presentation being obviously unique). Then the following equalities hold.*

$$\begin{aligned} W(T) &= \max\{W(T_1), W(T_2)+1\}, \\ D(T) &= \max\{D(T_1), D(T_2)\}+1. \end{aligned}$$

Proof. When the TSA builds the freshness tokens in the graph T, it first generates the the ones corresponding to the left subtree T_1 and then the ones corresponding to the right subtree T_2. In the latter case we see from Definition 3 that some of the nodes in the freshness tokens must capture all the leaves of T_1 as well. Definition 4 requires the number of these nodes to be as small as possible (note that the freshness tokens for the two subtrees are independent). Hence the best solution is to add the root of the subtree T_1 to all the freshness tokens of the subtree T_2. This proves the first equality.

In order to prove the second equality, that let r, r_1 and r_2 be the roots of the trees T, T_1 and T_2, respectively. Then for any item x_i

$$\mathsf{Auth}(x_i, r) = \begin{cases} \mathsf{Auth}(x_i, r_1) \cup \{r_2\}, & \text{if } x_i \in T_1; \\ \mathsf{Auth}(x_i, r_2) \cup \{r_1\}, & \text{if } x_i \in T_2. \end{cases}$$

The proof is now straightforward. □

Remark. Some care has to be taken here in order to make sure what an equation like

$$\mathsf{Auth}(x_i, r) = \mathsf{Auth}(x_i, r_1) \cup \{r_2\}$$

actually means considering that the sets $\mathsf{Auth}(x_i, r)$ and $\mathsf{Auth}(x_i, r_1)$ are not, in general, uniquely determined. One should read this equation in the following way: "For any possible authenticator $\mathsf{Auth}(x_i, r_1)$ we obtain an authenticator $\mathsf{Auth}(x_i, r)$ by adding the node r_2 to it". A similar clarification is applicable for the freshness tokens as well. [4]

It is interesting to note that the following lemma holds.

Lemma 3 *If T is a topologically sorted binary tree with leaves $x_1, x_2, \ldots, x_n$ and root r then for each index i*

$$\mathsf{FT}_i \subseteq \mathsf{Auth}(x_i, r)$$

holds.

As we do not need this lemma in the current thesis, we do not prove it here but refer to [BLS00] for the proof of a completely analogous result.

3.5 Complete trees

The complete (binary) tree $\mathfrak{C}^d$ of depth d is defined by the following recursive scheme[5]:

$$\mathfrak{C}^d = \begin{cases} I, & \text{if } d = 0, \\ \mathfrak{C}^{d-1} \otimes \mathfrak{C}^{d-1}, & \text{if } d > 0. \end{cases}$$

Clearly, $\|\mathfrak{C}^d\| = 2^d$ and $W(\mathfrak{C}^d) = D(\mathfrak{C}^d) = d$. Hence,

$$\beta(\mathfrak{C}^d) = \frac{d+d}{d} = 2.$$

It is also easy to see that the estimate obtained does not change if we consider the actual size of time stamps instead of the value $W(\mathfrak{C}^d) + D(\mathfrak{C}^d)$. This claim follows from the fact that interval time stamps in this tree can

[4] It is not difficult to see that in the case of topologically sorted binary trees both authenticators and freshness tokens are in fact unique. Still the above remark must be taken into account in future arguments.

[5] In this section and further on, the equality of graphs is actually an isomorphism. The author will use both $=$ and $\simeq$ to represent the isomorphism, whichever symbol seems more suitable in a particular case.

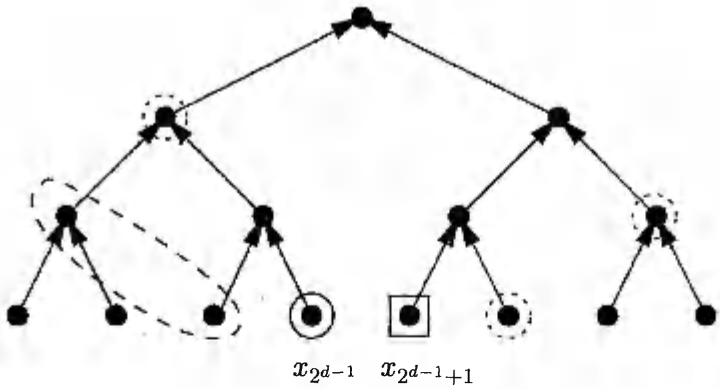

Figure 3.3: Complete tree with an interval time stamp of size $2d-1$.

have size $2d-1$. Indeed, if the requests are $x_1, x_2, \ldots, x_{2^d}$, the freshness token for the item $x_{2^{d-1}}$ (i.e. the set $\mathsf{FT}_{2^{d-1}-1}$) has size $d-1$ and the existence token of the item $x_{2^{d-1}+1}$ has size d, where the tokens are clearly disjoint; see Figure 3.3.

Binary trees were used as the basis for BLS-scheme [BLS00], and although proven to be the optimal ones for time-stamping in case of patent scenario, the estimate $\beta(\mathfrak{C}^d) = 2$ is not the best possible for interval time stamps. In Section 3.6, we will see that the ratio $\beta(G)$ can be improved by at least 25% by considering a different family of trees.

3.6 Schemes with $\beta(G) = 1.5 + o(1)$

Definition 11 *Let the family $\mathfrak{S}_w^d$ of trees be defined as follows*[6]*:*

$$\mathfrak{S}_w^d = \begin{cases} I, & \text{if } d = 0 \text{ or } w = 0; \\ \mathfrak{S}_w^{d-1} \otimes \mathfrak{S}_{w-1}^{d-1}, & \text{if } w > 0 \text{ and } d > 0. \end{cases}$$

See Figure 3.4 for an illustration of this recursive process. It is not hard to come up with the following hypothesis (which, as we will prove, is true) looking at the figure.

Theorem 2 *If $w \geq d$ then $\mathfrak{S}_w^d \sim \mathfrak{C}^d$.*

[6]The notation $\mathfrak{S}_w^d$ using Gothic capital S originates from the phrase "Signature time-stamping scheme" having in mind the primary application of the defined family of trees as time-stamping schemes for digital signatures.

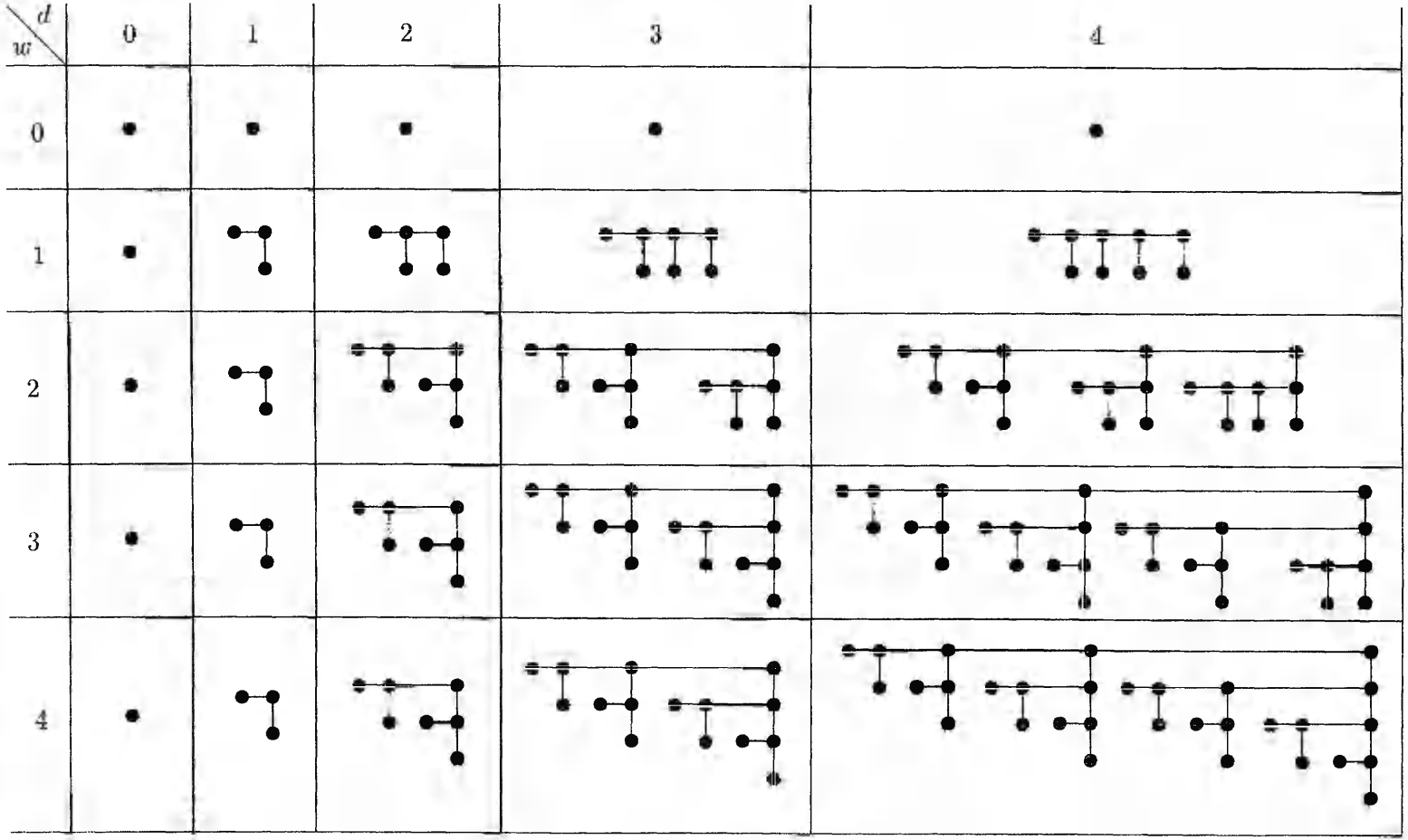

Figure 3.4: The process of forming the trees $\mathfrak{S}_w^d$.

Proof. We use induction on d. For $d = 0$

$$\mathfrak{S}_w^0 = I = \mathfrak{C}^0,$$

so the claim holds in this case. Assume now the claim of the Theorem is true for some d and consider a tree $\mathfrak{S}_w^{d+1}$, where $w \geq d+1$. By Definition 11, we have

$$\mathfrak{S}_w^{d+1} = \mathfrak{S}_w^d \otimes \mathfrak{S}_{w-1}^d = \mathfrak{C}^d \otimes \mathfrak{C}^d = \mathfrak{C}^{d+1}$$

by induction hypothesis, as $w \geq d+1$ implies both $w > d$ and $w - 1 \geq d$. □

The case of complete trees was already considered in Section 3.5. Next we will look at the case $w < d$ to try to locate trees $G = S_w^d$ for which $\beta(G) < 2$. Assuming the inequality $w < d$, we now prove the following theorem.

Theorem 3 *If $w < d$ then the following claims hold:*

1. $W(\mathfrak{S}_w^d) = w$;
2. $D(\mathfrak{S}_w^d) = \begin{cases} d, & \text{if } w > 0; \\ 0, & \text{if } w = 0; \end{cases}$
3. $\|\mathfrak{S}_w^d\| = \sum_{k=0}^{w} \binom{d}{k}$.

Proof.

1. We use induction on d. If $d = 1$ then $w = 0$ and

   $$W(\mathfrak{S}_0^1) = W(I) = 0,$$

   hence the claim holds for $d = 1$.

   For the step of induction we first note that the trees $\mathfrak{S}_w^d$ are all topologically sorted and hence Lemma 2 can be applied. We obtain

   $$W(\mathfrak{S}_w^d) = W(\mathfrak{S}_w^{d-1} \otimes \mathfrak{S}_{w-1}^{d-1}) = \max\{W(\mathfrak{S}_w^{d-1}), W(\mathfrak{S}_{w-1}^{d-1}) + 1\}.$$

   If now $w < d - 1$, we have

   $$\begin{aligned} W(\mathfrak{S}_w^d) &= W(\mathfrak{S}_w^{d-1} \otimes \mathfrak{S}_{w-1}^{d-1}) = \max\{W(\mathfrak{S}_w^{d-1}), W(\mathfrak{S}_{w-1}^{d-1}) + 1\} = \\ &= \max\{w, (w - 1 + 1)\} = w \end{aligned}$$

   because of the induction hypothesis.

   If $w = d - 1$ then $\mathfrak{S}_w^{d-1} = \mathfrak{C}^{d-1}$, consequently

   $$W(\mathfrak{S}_w^{d-1}) = W(\mathfrak{C}^{d-1}) = d - 1 = w$$

   and hence the above computation holds for this case as well.

2. If $w = 0$ then

$$D(\mathfrak{S}_w^d) = D(I) = 0.$$

If $w > 0$ we once again we use induction on d. The induction basis is verified exactly as above. For the induction step we obtain from Lemma 2 that the equality

$$D(\mathfrak{S}_w^d) = \max\{D(\mathfrak{S}_w^{d-1}), D(\mathfrak{S}_{w-1}^{d-1})\} + 1$$

holds. Now for $d+1$ (under assumption $w > 0$) we have:

$$\begin{aligned} D(\mathfrak{S}_w^{d+1}) &= \max\{D(\mathfrak{S}_w^d), D(\mathfrak{S}_{w-1}^d)\} + 1 = \\ &= \max\{d, (d \vee 0)\} + 1 = d + 1, \end{aligned}$$

where the notation $(d \vee 0)$ means an entity with the value being equal to d or 0 (depending on w). Note that the assumption $w < d$ is not actually needed for this part of the theorem.

3. First note that the claim holds for $w = 0$ as

$$\|\mathfrak{S}_0^d\| = \|I\| = 1 = \sum_{k=0}^{0} \binom{d}{k}.$$

Straightforwardly, the claim holds also for $w = d$:

$$\|\mathfrak{S}_d^d\| = \|\mathfrak{C}^d\| = 2^d = \sum_{k=0}^{d} \binom{d}{k}.$$

Now we use induction on d. For $d = 1$ we have $w = 0$ and the required equality holds as shown above. For $d > 1$ and $0 < w < d+1$ we have

$$\begin{aligned} \|\mathfrak{S}_w^{d+1}\| &= \|\mathfrak{S}_w^d\| + \|\mathfrak{S}_{w-1}^d\| = \sum_{k=0}^{w} \binom{d}{k} + \sum_{k=0}^{w-1} \binom{d}{k} \\ &= \binom{d}{0} + \sum_{k=1}^{w} \binom{d}{k} + \sum_{k=1}^{w} \binom{d}{k-1} \\ &= \binom{d+1}{0} + \sum_{k=1}^{w} \left[\binom{d}{k} + \binom{d}{k-1} \right] = \\ &= \binom{d+1}{0} + \sum_{k=1}^{w} \binom{d+1}{k} = \sum_{k=0}^{w} \binom{d+1}{k}. \end{aligned}$$

□

Unfortunately, there is no known closed formula for $\sum_{k=0}^{w} \binom{d}{k}$. Still, if $d = 2w + 1$ we can compute the exact value as follows:

$$\begin{aligned}
\|\mathfrak{S}_w^{2w+1}\| &= \sum_{k=0}^{w} \binom{2w+1}{k} = \frac{1}{2} \cdot \left[\sum_{k=0}^{w} \binom{2w+1}{k} + \sum_{k=0}^{w} \binom{2w+1}{k} \right] = \\
&= \frac{1}{2} \cdot \left[\sum_{k=0}^{w} \binom{2w+1}{k} + \sum_{k=w+1}^{2w+1} \binom{2w+1}{k} \right] = \\
&= \frac{1}{2} \cdot \sum_{k=0}^{2w+1} \binom{2w+1}{k} = \frac{1}{2} \cdot 2^{2w+1} = 2^{2w}.
\end{aligned}$$

So if we use $\mathfrak{T}_w = \mathfrak{S}_w^{2w+1}$ in a tree-based time-stamping scheme, the number of elements in time certificate for a digital signature is upper bounded by

$$W(\mathfrak{T}_w) + D(\mathfrak{T}_w) = w + 2w + 1 = 3w + 1.$$

Hence,

$$\beta(\mathfrak{T}_w) = \frac{W(\mathfrak{T}_w) + D(\mathfrak{T}_w)}{\log_2 \|\mathfrak{T}_w\|} = 3/2 + o(1),$$

which about 25% less than in the complete tree scheme. The next chapter shows that this estimate can be improved even further, but not too much.

4 OPTIMAL LINKING SCHEMES

4.1 Optimal family of schemes

In order to find the optimal schemes in the sense of the ratio (3.4) we study the situation in more detail. Let $\mathfrak{G}$ be the class of all finite rooted directed acyclic graphs. Our aim is to prove the following theorem.

Theorem 4 *For the characteristic β the following is true:*

$$\inf\{\beta(G) : G \in \mathfrak{G}\} = \frac{1}{\log_2\left(\frac{1+\sqrt{5}}{2}\right)}.$$

This infimum is approached for the trees $\mathfrak{S}_w^d$ with

$$\frac{w}{d} \approx \frac{3-\sqrt{5}}{2} \quad \textit{and} \quad d \to \infty.$$

We present a proof in several steps by proving a sequence of lemmas; the proof methodology is similar to the one found in [BLS00]. Each of the first four lemmas shows one reduction from more general class of graphs to more specific ones until we end up with the trees $\mathfrak{S}_w^d$. The crucial point is to show how to carry all the reductions out without increasing the value $\beta(G)$. The final lemma determines the optimal value of $\beta(\mathfrak{S}_w^d)$. Note that as the value

$$\frac{1}{\log_2\left(\frac{1+\sqrt{5}}{2}\right)}$$

is irrational[1], but all the values $\beta(G)$ are rational no graph G can have this value for $\beta(G)$, it can only be approached asymptotically. The statements of the lemmas are the following.

Lemma 4 *For any rooted directed acyclic graph G there exists a tree T such that $\beta(T) \leq \beta(G)$.*

[1]This claim is not difficult to prove by the following standard argument. If $\log_2\left(\frac{1+\sqrt{5}}{2}\right) = \frac{a}{b}$ with $a, b \in \mathbb{N}$, we get $\frac{1+\sqrt{5}}{2} = 2^{a/b}$ and $1+\sqrt{5} = 2^{(a+b)/b}$. Hence, $(1+\sqrt{5})^b = A + B\sqrt{5}$ $(A, B \in \mathbb{N}\backslash\{0\})$ should be an integer, a contradiction.

Lemma 5 *For any tree T there exists a binary tree T' such that $\beta(T') \leq \beta(T)$.*

Lemma 6 *For any binary tree T' there exists a topologically sorted binary tree T'' such that $\beta(T'') \leq \beta(T')$.*

Lemma 7 *For given non-negative integers w and d, any topologically sorted binary tree T having the greatest number of leaves and $W(T) = w$, $D(T) = d$, is isomorphic to $\mathfrak{S}_w^d$.*

Lemma 8 *The equality*

$$\inf\{\beta(\mathfrak{S}_w^d) : w < d\} = \frac{1}{\log_2\left(\frac{1+\sqrt{5}}{2}\right)}$$

holds. This infimum is approached for

$$\frac{w}{d} \approx \frac{3-\sqrt{5}}{2} \quad \textit{and} \quad d \to \infty.$$

4.2 Proof of Lemma 4

Assume first that we have any rooted directed acyclic graph G as our time-stamping scheme. If it is not a tree (otherwise, the lemma is done), we must have vertices with out-valency greater than 1. Let v be a vertex with out-valency $k \geq 2$ such that all of its predecessors have out-valency 1; hence v is the root of an induced subtree T of G (such a v exists because G is acyclic).

Consider the authenticator $\mathsf{Auth}(v, r)$. Let $v_1, \ldots, v_k$ be all the direct successors of v and let the authentication path corresponding to $\mathsf{Auth}(v, r)$ start with the arc (v, v_1). We will show that deleting the arcs (v, v_2),..., (v, v_k) (and possibly some other arcs and vertices) from G does not increase the value of $\beta(G)$.

As a result of edge deletion, cardinalities of the sets FT_i cannot increase, but cardinalities of the sets $\mathsf{Auth}(x_i, r)$, in principle, can. We study this problem in more detail. The analysis will be carried out for two different cases.

1. Consider first the items x_i not belonging to the subtree rooted in v together with their authentication paths and the corresponding authentication path support arcs. It is clear that if none of the arcs

$(v, v_2), (v, v_3), \ldots, (v, v_k)$ belongs to any of the sets of authentication path support arcs then deleting the arcs $(v, v_2), (v, v_3), \ldots, (v, v_k)$ does not affect any of the sets $\mathsf{Auth}(x_i, r)$. But if we deleted some authentication path support arc then it may happen that the cardinalities of some sets $\mathsf{Auth}(x_i, r)$ increase. There are two closely connected cases how this may be possible.

As none of the items x_i under consideration belong to the subtree rooted in v, then no authentication path support arc under consideration belongs to this subtree. Hence, deleting the arcs (v, v_2),(v, v_3),..., (v, v_k) essentially means that the vertex v is removed from some proof sets. From Definitions 1 and 2 it follows that if we want to retain as much as possible from an old proof set (say, proving the dependence of r on the vertex x_{i_0}), we must replace v with some (possibly several) of its successors. This way the cardinality of one proof set can increase and thus the same can happen to the corresponding authenticator as well.

It is also possible that after the authenticator corresponding to some authentication path has considerably increased, some other authenticator corresponding to some other authentication path turns out to be smaller (but still larger than the original authenticator $\mathsf{Auth}(x_{i_0}, r)$). Hence, this case may result in increase of the cardinality of $\mathsf{Auth}(x_{i_0}, r)$ as well.

In both cases, it is enough to show how to modify the graph some more so that the new authenticators will either coincide with the original ones or even have one element (namely v) less.

This modification will be done by removing some more vertices (and of course the arcs that loose one end-vertex) from the graph. The nodes to be removed will be the ones from the set $\{v_2, \ldots, v_k\}$ that had no other parents than just v in the original graph G; and recursively all their successors that had no other predecessors than v and the ones already deleted.

After such modification there are two possibilities.

(a) If (v, v_1) was an authentication path support arc for some authenticator $\mathsf{Auth}(x_i, r)$ in the original graph, then v is not removed from $\mathsf{Auth}(x_i, r)$. Still, all the successors of v added to authenticators in the meantime are deleted. Hence, all in all, the cardinality of $\mathsf{Auth}(x_i, r)$ did not increase.

(b) If (v, v_1) was not an authentication path support arc for some

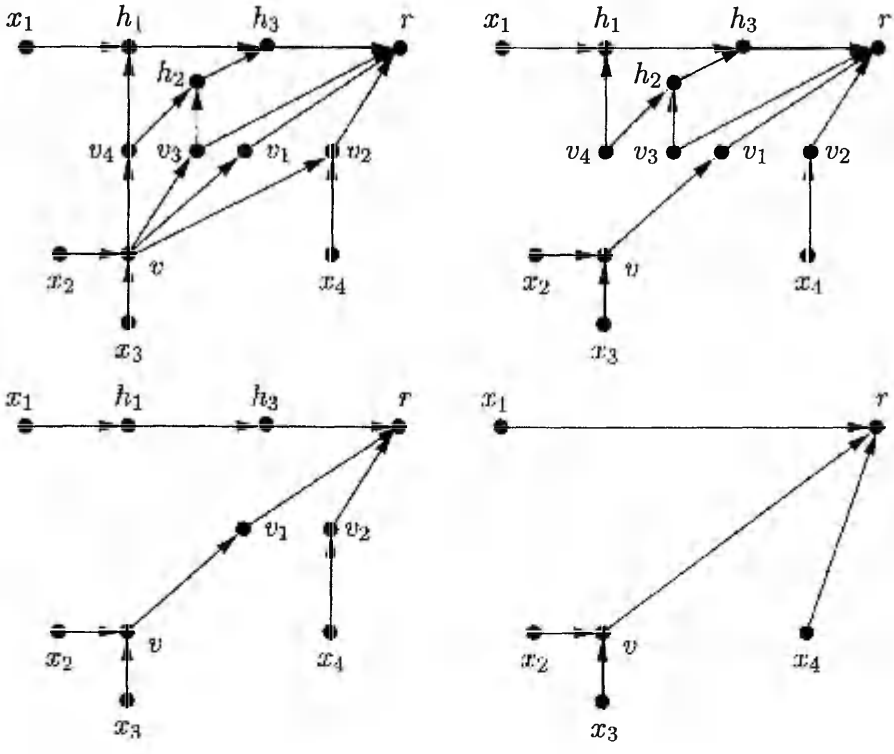

Figure 4.1: Transformations of a linking scheme.

authenticator $\mathsf{Auth}(x_i, r)$ in the original graph then the new authenticator has lost one element v.

It is possible that after such a transformation some nodes still have in-valency 1. Then the corresponding edges can also be deleted if the nodes are joined with their parent.

An example of all transformations is depicted in Figure 4.1. In the figure, we have already denoted the successors of v so that $v \to v_1 \to r$ is the authentication path having one of the proof sets of the smallest possible size, namely $\{v_2, v_3, h_3\}$. As the vertices v_3, v_4 and h_2 depend only on v they are deleted (together with the outgoing arcs, of course). At the very last step, we also delete the nodes of in-valency 1. Note that finally, v is once again a member of the proof set for the authentication path from x_4 to r!

2. Let x_i be a leaf in the subtree with root v. It is clear that

(a) every authentication path from x_i to r must go through v, and

(b) every proof set (proving the dependence of r on x_i) consists of two subsets of $V(G)$ – a subset A of vertices of the subtree rooted in v and a subset B of the vertices of the remaining graph. Besides that, from observation (a) we see that $v \notin A$ and $v \notin B$.

The arc deletion process described above does not influence the set A, but it may influence the set B. If we consider the proof set to be $\mathsf{Auth}(x_i, r)$ in the original graph G, then we must have $|B| = |\mathsf{Auth}(v, r)|$. Disregarding for a moment the whole subtree rooted in v and keeping only v, this vertex becomes a leaf in the modified graph. Applying exactly the same argument as in the first part of the proof we see that deletion of the arcs $(v, v_2), \ldots, (v, v_k)$ does not increase the cardinality of $\mathsf{Auth}(v, r)$.

As a result of these transformations in the graph G we have decreased the number of nodes with out-valency more than 1

- without increasing any of the sets $\mathsf{Auth}(x_i, r)$;
- without increasing any of the sets FT_i; and
- without changing the number of leaves.

By repeating the process with the remaining nodes of out-valency more than 1, we eventually arrive at a tree T with $\beta(T) \leq \beta(G)$.

4.3 Proof of Lemma 5

If T is not a binary tree, there must be a vertex v with only one child or more than two children.

In the first case, we may simply delete v and join its only child with its only parent (if this vertex is the root, we simply delete it, leaving the child as the new root). After such a vertex deletion the cardinalities of the sets FT_i and $\mathsf{Auth}(x_i, r)$ can only decrease.

In the second case, we introduce some additional vertices as shown in Figure 4.2.

These additional vertices contain information about several children of v in the original tree. Hence, it may be possible to decrease the cardinalities of FT_i and $\mathsf{Auth}(x_i, r)$. For example, in Figure 4.2 (left) the nodes b, c, d, e are necessary in order to prove the dependence of v on a, but in Figure 4.2 (right) only the vertex u is enough.

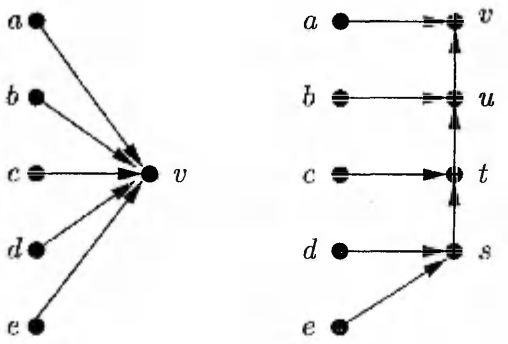

Figure 4.2: Transformation from an arbitrary tree to a binary tree.

By continuing this process we eventually arrive at tree T' where every vertex has either 2 or 0 children, i.e. a binary tree. From the proof above it is also clear that $\beta(T') \leq \beta(T)$.

4.4 Proof of Lemma 6

Let T' be a binary tree with its leaves sorted in some (not necessarily topological) order. We show how to reorder the leaves without changing the basic structure of the tree. By doing so, we do not change the size of authenticators and neither the number of leaves. Hence, in order to complete the proof, we must show that reordering can be done without changing the size of freshness tokens.

First we label the nodes of T' in the fashion of Section 3.4.

1. Let the root be labeled with the empty string λ.
2. For every vertex labeled with a string σ, label the child that has the leftmost predecessor as σL and the other one as σR. We call σL the left child and σR the right child of σ.

From Definition 10 we get that the whole tree becomes topologically sorted. Hence, we are only required to prove that this rearrangement does not increase any freshness tokens. We do it by proving the following lemma.

Lemma 9 *If the vertices are labeled as described above, the freshness token* FT_{i-1} *must have at least as many elements as there are letters* R *in the label of the leaf* x_i.

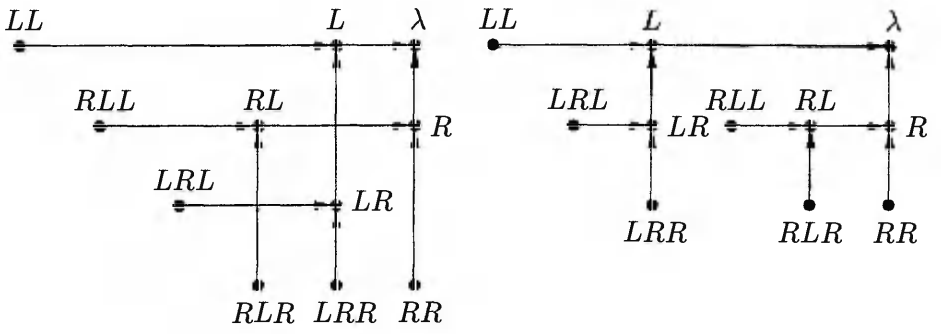

Figure 4.3: Transformation from a binary tree to a sorted tree.

Proof. The number of letters R in the label of the leaf x_i shows how many times in the process of proving the dependence of the root node on x_i it is necessary to invoke information from earlier time, i.e. from the "left" on the time-line of the items. As T' is a tree, these invocations are independent and hence, the set FT_{i-1} must contain separate elements for all of the invocations. Consequently, FT_{i-1} contains at least the same number of elements as there are letters R in the label of the leaf x_i. □

To conclude the proof it remains to note that for a topologically sorted tree the sets FT_i contain nothing but the necessary information and hence resorting the tree topologically can only decrease their cardinalities.

The process of reordering is depicted in Figure 4.3.

4.5 Proof of Lemma 7

If $d = 0$ or $w = 0$ we must have $T = I$ and hence the Lemma is proven in this case.

Assume that for some $w, d > 0$ there exist topologically sorted binary trees such that the claim does not hold. Let T be a tree among them such that the sum $w + d$ is the smallest possible; so $\|T\| > \|\mathfrak{S}_w^d\|$. Obviously, $T \neq I$. Thus it is possible to represent T as $T = T_1 \otimes T_2$. From Lemma 2 we get the following equalities:

$$\begin{aligned} w = W(T) &= \max\{W(T_1), W(T_2) + 1\}, \\ d = D(T) &= \max\{D(T_1), D(T_2)\} + 1. \end{aligned}$$

Consequently, the inequalities

$$\begin{aligned} D(T_1) &\leq d-1, \\ W(T_1) &\leq w, \\ D(T_2) &\leq d-1, \\ W(T_2) &\leq w-1. \end{aligned}$$

hold.

As $w+(d-1) < w+d$ and $(w-1)+(d-1) < w+d$, the Lemma holds for both T_1 and T_2. Thus,

$$\|T\| = \|T_1\| + \|T_2\| \leq \|\mathfrak{S}_w^{d-1}\| + \|\mathfrak{S}_{w-1}^{d-1}\| = \|\mathfrak{S}_w^{d-1} \otimes \mathfrak{S}_{w-1}^{d-1}\| = \|\mathfrak{S}_w^d\|,$$

a contradiction.

4.6 Proof of Lemma 8

In the light of Theorem 3, the ratio $\beta(G)$ to be estimated can be written as

$$\frac{w+d}{\log_2 n} - \frac{w+d}{\log_2(\sum_{k=0}^{w}\binom{d}{k})}. \tag{4.1}$$

In order to give a better upper bound to the expression (4.1) we need an asymptotic formula for the sum $\sum_{k=0}^{w}\binom{d}{k}$ (which does not have a known closed formula). In [GKP89], problem 9.42, it is proven that if $\frac{w}{d} = \alpha < \frac{1}{2}$ then

$$\sum_{k=0}^{w}\binom{d}{k} \approx 2^{d\cdot K(\alpha) - 0.5\cdot\log_2 d + O(1)}, \tag{4.2}$$

where

$$K(\alpha) = \alpha\cdot\log_2\frac{1}{\alpha} + (1-\alpha)\cdot\log_2\frac{1}{1-\alpha}.$$

Substituting (4.2) into (4.1), we get

$$\begin{aligned} \lim_{d\to\infty}\frac{w+d}{\log_2(\sum_{k=0}^{w}\binom{d}{k})} &= \lim_{d\to\infty}\frac{\alpha d + d}{d\cdot K(\alpha) - 0.5\cdot\log_2 d + O(1)} \\ &= \lim_{d\to\infty}\frac{1+\alpha}{K(\alpha) - 0.5\cdot\frac{\log_2 d}{d} + O(\frac{1}{d})} \\ &= \frac{1+\alpha}{K(\alpha)}. \end{aligned}$$

For deriving the approximation formula for $0.5 \leq \alpha < 1$, we note first that for such α and for sufficiently large d,

$$\begin{aligned}\sum_{k=0}^{w}\binom{d}{k} &= 2^d - \sum_{k=w+1}^{d}\binom{d}{d-k} = 2^d - \sum_{j=0}^{d-w-1}\binom{d}{j} \approx \\ &\approx 2^d - 2^{d\cdot K(\alpha')-0.5\cdot\log_2 d+O(1)},\end{aligned}$$

where $\alpha' = (d-w-1)/d = 1-\alpha-1/d < 1-\alpha \leq 0.5$. Hence, for $0.5 \leq \alpha < 1$ we get

$$\begin{aligned}\lim_{d\to\infty}\frac{w+d}{\log_2(\sum_{k=0}^{w}\binom{d}{k})} &= \lim_{d\to\infty}\frac{\alpha d + d}{\log_2\{2^d\cdot(1-2^{[K(\alpha')-1]-0.5\cdot\log_2 d+O(1)})\}} \\ &= \lim_{d\to\infty}\frac{1+\alpha}{1+\frac{1}{d}\cdot\log_2\left(1-2^{d\cdot[K(1-\alpha-1/d)-1]-0.5\cdot\log_2 d+O(1)}\right)} \\ &= 1+\alpha.\end{aligned}$$

Therefore,

$$\Phi(\alpha) = \lim_{d\to\infty}\frac{w+d}{\log_2(\sum_{k=0}^{w}\binom{d}{k})} = \begin{cases}\frac{1+\alpha}{K(\alpha)}, & \text{if } 0<\alpha<\frac{1}{2};\\ 1+\alpha, & \text{if } \frac{1}{2}\leq\alpha<1.\end{cases} \tag{4.3}$$

Note that Φ is continuous at $\frac{1}{2}$ because $K\left(\frac{1}{2}\right) = 1$. The graph of the function Φ is depicted in Figure 4.4.

For finding minima of $\Phi(\alpha)$ we solve the equation

$$\Phi'(\alpha) = \frac{2\cdot\log_2\frac{1}{1-\alpha} - \log_2\frac{1}{\alpha}}{H^2(\alpha)} = 0.$$

Hence, $2\cdot\log_2\frac{1}{1-\alpha} - \log_2\frac{1}{\alpha} = 0$ which implies that $(1-\alpha)^2 = \alpha$. This quadratic equation has a unique solution in the interval $[0, 0.5]$, namely

$$\alpha_0 = \frac{3-\sqrt{5}}{2}.$$

Let $\phi = \frac{1+\sqrt{5}}{2}$ denote the Golden ratio. It is easy to verify that

$$\frac{1}{1-\alpha_0} = \phi \qquad \text{and} \qquad \frac{1}{\alpha_0} = \phi^2 \approx 2.61803.$$

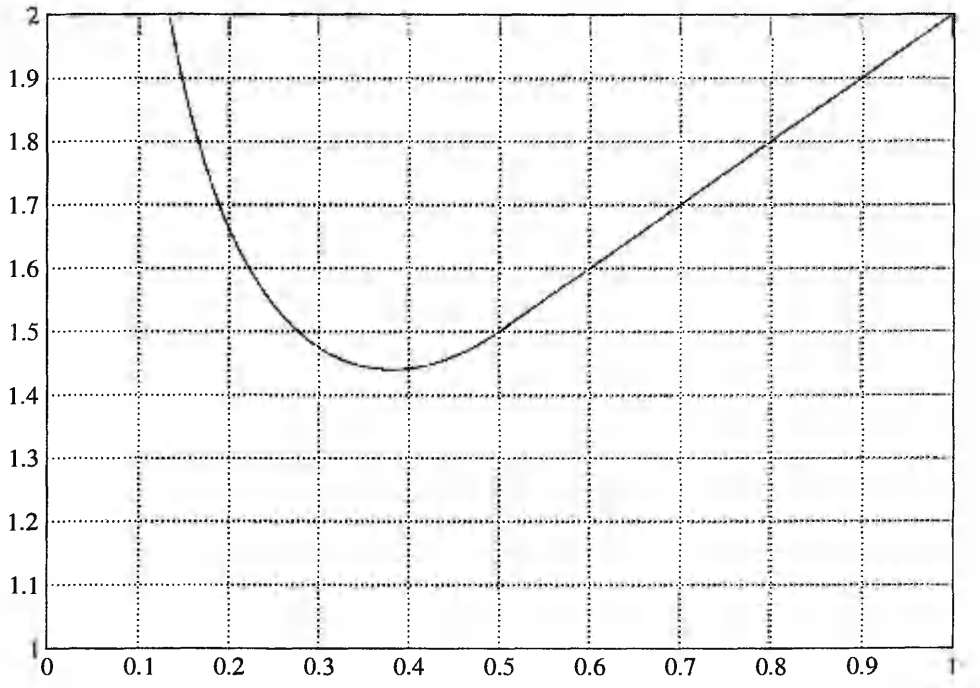

Figure 4.4: Graph of the function $\Phi(\alpha)$.

Hence,

$$\begin{aligned}
\frac{1+\alpha_0}{K(\alpha_0)} &= \frac{1+\alpha_0}{\alpha_0 \cdot \log_2 \frac{1}{\alpha_0} + (1-\alpha_0) \cdot \log_2 \frac{1}{1-\alpha_0}} \\
&= \log_2^{-1} \left[\left(\frac{1}{\alpha_0} \right)^{\frac{\alpha_0}{1+\alpha_0}} \cdot \left(\frac{1}{1-\alpha_0} \right)^{\frac{1-\alpha_0}{1+\alpha_0}} \right] \\
&= \log_2^{-1} \left[\phi^{\frac{2\alpha_0}{1+\alpha_0}} \cdot \phi^{\frac{1-\alpha_0}{1+\alpha_0}} \right] = \frac{1}{\log_2 \phi} \approx 1.44042.
\end{aligned}$$

We conclude that based on the trees $\mathfrak{S}_w^d$ asymptotically optimal time-stamping schemes are obtained if $w \approx \frac{3-\sqrt{5}}{2} d \approx \frac{d}{2.61803}$.

In order to get some idea about the speed of convergence we provide Table 4.1. The table shows the values of the ratio (3.4) for the previous best schemes (with $d = 2w+1$), for the schemes with $w = \lfloor \frac{d}{2.618\ldots} \rceil$ (where $\lfloor \cdot \rceil$ denotes the closest integer function) and for size-optimal schemes. The numbers of leaves of size-optimal schemes are also shown.

d	$w = \frac{d-1}{2}$: ratio	$w = \left\lfloor \frac{d}{2.618\ldots} \right\rfloor$: ratio	w_{opt}: ratio	n_{opt}
9	4: 1.62500	3: 1.70833	4: 1.62500	256
19	9: 1.55556	7: 1.57355	8: 1.55412	169766
27	13: 1.53846	10: 1.54249	12: 1.53015	47050564
29	14: 1.53571	11: 1.53353	12: 1.52562	123012781
49	24: 1.52083	19: 1.50157	20: 1.49946	$7 \cdot 10^{13}$
89	44: 1.51136	34: 1.48020	36: 1.47902	$3 \cdot 10^{25}$
129	64: 1.50781	49: 1.47064	51: 1.46970	$7 \cdot 10^{36}$
239	119: 1.50420	91: 1.45907	93: 1.45875	$3 \cdot 10^{68}$
589	294: 1.50170	225: 1.44947	227: 1.44943	$3 \cdot 10^{169}$

Table 4.1: Convergence of the linking scheme parameters.

5 LINKING USING TREES $\mathfrak{S}_w^d$

In the previous chapters, we mainly concentrated on building linking schemes that provide minimal sizes for time stamps. Still, in order to ensure suitability for practical use, we must address the problem of efficiency in the process of computation by following a particular scheme.

There are two main concerns we should address in more detail.

1. It is impractical for the TSA to construct first the whole scheme as some empty data structure and then start filling it with data. This data structure is in fact necessary only for keeping track of the computations and not for storing all the values; only a very small number of values are needed for further computations.

2. As noted in [ABSW01], the availability of time-stamping service is a major issue. If the TSA's server crashes, it must be possible to restore the last complete set FT_i in order to ensure that the causal relationships between the time stamps issued before and after the crash do not break. Note that the original definition of the trees $\mathfrak{S}_w^d$ is recursive and hence the algorithm following this definition closely must be recursive as well. Collecting the data necessary to restore the work of a recursive-algorithm-based server basically means backuping the recursion stack at every step. This in turn means implementing a recursion stack independent of the compiler's one. The author is currently unaware of any compiler having primitives for generating recursion stack dumps and restoring the processes later by these dumps.

The state of the algorithm presented in this chapter is stored in $2l + 3$ variables where l is the largest number of nodes in the sets FT_i. This is much less than the storage space required to keep the whole empty data-structure. These variables are also considerably easier to handle in the case of server recovery than the recursion stack.

5.1 Alternative description of the trees $\mathfrak{S}_w^d$

In this section, we are going to present a non-recursive description for the graphs $\mathfrak{S}_w^d$ enabling us also to find an efficient algorithm for generating these graphs.

Definition 12 *The set $\Sigma_{w,d}$ is defined as the following set of words in the alphabet $\{1,2,\ldots,d\}^*0$:*

$$\Sigma_{w,d} = \{\sigma 0 : \sigma = \sigma_1\sigma_2\ldots\sigma_{|\sigma|} \in \{1,2,\ldots,d\}^*, |\sigma| \leq w, \sigma_1 > \sigma_2 > \ldots > \sigma_{|\sigma|}\}.$$

Here, if $d = 0$ we assume that $\{1,2,\ldots,d\} = \emptyset$ and $\{1,2,\ldots,d\}^* = \{\lambda\}$, where λ is the empty word.

That is, the set $\Sigma_{w,d}$ consists of all strictly decreasing vectors of length up to $w+1$, having the elements from the set $\{0,1,\ldots,d\}$ and ending with 0. Note that every element occurs in every vector at most once, 0 occurs exactly once. In the rest of the paper we will write the vectors as words, without parentheses and commas. Greek lowercase letters will denote the words and Latin lowercase letters x, y their elements.

Definition 13 *For $w, d \geq 0$ let $m_{w,d}$ denote the vector*

$$m_{w,d} = \begin{cases} 0, & w = 0 \vee d = 0, \\ d(d-1)\ldots(d-w+1)0, & d > w > 0, \\ d(d-1)\ldots 10, & w \geq d > 0. \end{cases}$$

Denote also the set of the elements of the vector $m_{w,d}$ as $M_{w,d}$.

Theorem 5 *The set $\Sigma_{w,d}$ is linearly ordered with respect to lexicographic order $\preceq$. It's least element is the vector 0 and the largest element is $m_{w,d}$.*

Proof. Linearity of the lexicographic order is a well-known fact and minimality of the vector 0 is obvious. Maximality of the vector $m_{w,d}$ can also be easily established. If $w = 0$ or $d = 0$, then $\Sigma_{w,d} = \{0\}$ and also $m_{w,d} = 0$. Otherwise, it is clear that the largest vector has to start with the largest element of the alphabet, d. As all the letters in the vectors of $\Sigma_{w,d}$ must be unique, the next letter in the largest vector must be $d-1$ etc. The length of the vector is bounded by either the maximal allowed length $w+1$ if $d > w$, or by lack of possible elements if $w \geq d$. □

Theorem 6 *For $w, d \geq 1$ the equality*

$$\Sigma_{w,d} = \Sigma_{w,d-1} \cup d\Sigma_{w-1,d-1}$$

holds.

Proof.

$$\begin{aligned}
\Sigma_{w,d} &= \{\sigma 0 : \sigma \in \{1,2,\ldots,d\}^*, |\sigma| \leq w, \sigma_1 > \sigma_2 > \ldots > \sigma_{|\sigma|}\} = \\
&= \{\sigma 0 : \sigma \in \{1,2,\ldots,d-1\}^*, |\sigma| \leq w, \sigma_1 > \sigma_2 > \ldots > \sigma_{|\sigma|}\} \cup \\
&\cup \{\sigma 0 : \sigma \in \{1,2,\ldots,d\}^*, |\sigma| \leq w, d = \sigma_1 > \sigma_2 > \ldots > \sigma_{|\sigma|}\} = \\
&= \Sigma_{w,d-1} \cup \{d\tau 0 : \tau \in \{1,2,\ldots,d-1\}^*, |\tau| \leq w-1, \tau_1 > \ldots > \tau_{|\tau|}\} = \\
&= \Sigma_{w,d-1} \cup d\Sigma_{w-1,d-1}
\end{aligned}$$

□

Definition 14 *For each $\sigma \in \Sigma_{w,d}$ we define the set*

$$c_{w,d}(\sigma) = \{x \in \sigma : \sigma = \rho x \pi, \forall \tau \in \Sigma_{w,d}\ [\tau = \rho x \varsigma \Rightarrow \tau \preceq \sigma]\}.$$

That is, $c_{w,d}(\sigma)$ consists of such elements x of the vector σ that σ is the greatest vector among the vectors having the same initial segment up to the element x as σ does (recall form Definition 12 that each occurrence of every element in σ is unique).

Lemma 10 *The following properties of function c hold.*

1. *Let σ be presented in the form $\sigma = \rho\tau$, where $\tau = (x-1)(x-2)\ldots y0$ for some $x, y \in \{1,\ldots,d+1\}$ and τ has maximal possible length (if $\sigma = 0$ we have $\rho = \lambda$ and $\tau = 0$). If $(y = 1) \vee (|\sigma| = w+1)$ then $c_{w,d} = \{x-1, x-2, \ldots, y, 0\}$, otherwise $c_{w,d} = \{0\}$.*
2. *$d \in c_{w,d}(\sigma) \Leftrightarrow c_{w,d}(\sigma) = M_{w,d} \Leftrightarrow \sigma = m_{w,d}$;*
3. *$\sigma \in \Sigma_{w,d-1} \Rightarrow c_{w,d}(\sigma) = c_{w,d-1}(\sigma)$;*
4. *$\sigma \in \Sigma_{w-1,d-1} \Rightarrow c_{w,d}(d\sigma) = \begin{cases} c_{w-1,d-1}(\sigma), & \sigma \neq m_{w-1,d-1} \\ M_{w,d}, & \sigma = m_{w-1,d-1} \end{cases}$*

Proof.

1. The maximality condition on τ implies that ρ does not end with x and the condition $(y = 1) \vee (|\sigma| = w+1)$ essentially means that the vector σ can not be made longer by adding elements before the last 0. If the latter is not the case, there exists $z \in \{1,2,\ldots,d\}$ such that $\rho(x-1)\ldots yz0 \in \Sigma_{w,d}$ and as $\rho(x-1)\ldots y0 \prec \rho(x-1)\ldots yz0$, only 0 can be in the set $c_{w,d}$ by Definition 14. On the other hand, if we can not add such an z, each element of the set $\{x-1, x-2, \ldots, y, 0\}$ satisfies the condition given in Definition 14. It is also clear that no element of ρ can be in $c_{w,d}(\sigma)$ as $\rho x0 \in \Sigma_{w,d}$ and $\rho(x-1)\ldots y0 \prec \rho x0$.

2. We will prove that $d \in c_{w,d}(\sigma) \Rightarrow c_{w,d}(\sigma) = M_{w,d} \Rightarrow \sigma = m_{w,d} \Rightarrow d \in c_{w,d}(\sigma)$.
   If $d \in c_{w,d}(\sigma)$, we must have $a = d$ and $\rho = \lambda$ in the previous claim of the Lemma. If $d = 0$ the present claim is obvious. If $w \geq d > 0$, we get the case $b = 1$ and $c_{w,d}(\sigma) = \{d, d-1, \ldots, 1, 0\}$, if $d > w > 0$, the length of σ is bounded by $w+1$ and we get $c_{w,d}(\sigma) = \{d, d-1, \ldots, d-w+1, 0\}$. In both cases we have $c_{w,d}(\sigma) = M_{w,d}$.
   If $c_{w,d}(\sigma) = M_{w,d}$, we see that every element of $M_{w,d}$ also belongs to σ, hence $\sigma = m_{w,d}$.
   If $\sigma = m_{w,d}$, we obviously have $d \in c_{w,d}(\sigma)$, thus concluding the proof.

3. We know from the proof of Theorem 6 that $\Sigma_{w,d-1} = \{\sigma \in \Sigma_{w,d} : \sigma_1 \neq d\}$. As $\sigma \in \Sigma_{w,d-1}$, we can write

$$\begin{aligned} c_{w,d}(\sigma) &= \{x \in \sigma : \sigma = \rho x \pi, \forall \tau = \rho x \varsigma \in \Sigma_{w,d}\, \tau \preceq \sigma\} = \\ &= \{x \in \sigma : \sigma = \rho x \pi, \forall \tau = \rho x \varsigma \in \Sigma_{w,d-1}\, \tau \preceq \sigma\} = \\ &= c_{w,d-1}(\sigma). \end{aligned}$$

4. We know from Theorem 6 that $d\Sigma_{w-1,d-1} = \{\sigma \in \Sigma_{w,d} : \sigma_1 = d\}$. For a vector $d\sigma \in d\Sigma_{w-1,d-1}$ we can distinguish two cases.

   (a) $d \in c_{w,d}(d\sigma)$. From the second claim of the Lemma this holds iff $c_{w,d}(d\sigma) = M_{w,d}$ and $d\sigma = m_{w,d}$, which is equivalent to $\sigma = m_{w-1,d-1}$.

   (b) $d \notin c_{w,d}(d\sigma)$. From the second claim of the Lemma this is equivalent to $\sigma \neq m_{w-1,d-1}$. We compute:

$$\begin{aligned} c_{w,d}(d\sigma) &= \{x \in d\sigma : d\sigma = d\rho x \pi, \forall \tau = d\rho x \varsigma \in \Sigma_{w,d}\, \tau \preceq d\sigma\} = \\ &= \{x \in \sigma : \sigma = \rho x \pi, \forall \pi = \rho x \varsigma \in \Sigma_{w-1,d-1}\, \pi \preceq \sigma\} = \\ &= c_{w-1,d-1}(\sigma). \end{aligned}$$

□

Definition 15 *The directed rooted graph $S_{w,d}$ has the vertex set*

$$V(S_{w,d}) = \bigcup_{\sigma \in \Sigma_{w,d}} \{(\sigma, x) : x \in c_{w,d}(\sigma)\}$$

and the edge set

$$\begin{aligned} E(S_{w,d}) &= \{((\sigma, x_1)(\sigma, x_2)) \in (V(S_{w,d}))^2 : \sigma = \tau x_2 x_1 \rho\} \cup \\ &\cup \{((\tau x \rho, x)(\tau y \pi, y)) \in (V(S_{w,d}))^2 : y = x + 1\}. \end{aligned}$$

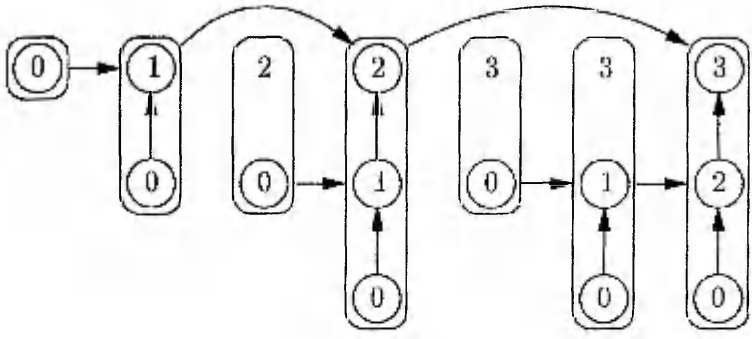

Figure 5.1: The graph $S_{2,3}$.

An example of the graph $S_{2,3}$ is presented in Figure 5.1. The vectors in the figure are depicted vertically and the elements of $c_{w,d}(\sigma)$ are circled. The edges of $S_{w,d}$ are of two kinds: the edges joining the vertices belonging to the same vector σ and the edges joining the vertices belonging to different vectors. The first ones will be called *vertical* and the second ones *horizontal edges* (following the intuition of Figure 5.1).

Now we are ready to state the following theorem claiming that the trees $\mathfrak{S}_w^d$ and $S_{w,d}$ are essentially the same implying that it is enough to give an algorithm for building the graphs $S_{w,d}$.

Theorem 7 $\mathfrak{S}_w^d \simeq S_{w,d}$. *The root of the graph* $S_{w,d}$ *is* $(m_{w,d}, d)$.

Proof. The proof will be given using induction by the definition of the graph $\mathfrak{S}_w^d$ (see Definition 11). The basis of induction consists of the cases where either $w = 0$ or $d = 0$. In both cases we see that $\Sigma_{w,d} = \{0\}$ and $c_{w,d}(0) = \{0\}$. Hence $V(S_{w,d}) = \{(0,0)\}$ and there can be no edges in the graph following Definition 15. Consequently $S_{w,d} \simeq I = \mathfrak{S}_w^d$ if $w = 0$ or $d = 0$, with the root being obviously the only vertex of the graph.

Now assume that $w, d \geq 1$, $\mathfrak{S}_{w-1}^{d-1} \simeq S_{w-1,d-1}$ and $\mathfrak{S}_w^{d-1} \simeq S_{w,d-1}$. In order to prove that

$$S_{w,d} \simeq \mathfrak{S}_w^d = \mathfrak{S}_w^{d-1} \otimes \mathfrak{S}_{w-1}^{d-1},$$

we first need to establish a one-to-one correspondence between the vertex sets $V(S_{w,d})$ and $V(S_{w,d-1} \otimes S_{w-1,d-1})$. The necessary correspondence can

be easily derived from Theorem 6 and Lemma 10 as follows.

$$\begin{aligned}
V(S_{w,d}) &= \bigcup_{\sigma\in\Sigma_{w,d}} \{(\sigma,x) : x \in c_{w,d}(\sigma)\} = \\
&= \bigcup_{\sigma\in\Sigma_{w,d-1}} \{(\sigma,x) : x \in c_{w,d-1}(\sigma)\} \cup \bigcup_{\sigma\in d\Sigma_{w-1,d-1}} \{(\sigma,x) : x \in c_{w,d}(\sigma)\} = \\
&= V(S_{w,d-1}) \cup \bigcup_{\sigma\in d\Sigma_{w-1,d-1}} \{(\sigma,x) : x \in c_{w,d}(\sigma)\} = \\
&= V(S_{w,d-1}) \cup \bigcup_{\sigma\in\Sigma_{w-1,d-1}} \{(d\sigma,x) : x \in c_{w,d}(d\sigma)\} = \\
&= V(S_{w,d-1}) \cup \bigcup_{\sigma\in\Sigma_{w-1,d-1}} \{(d\sigma,x) : x \in c_{w-1,d-1}(\sigma)\} \cup \{(m_{w,d},d)\} = \\
&= V(S_{w,d-1}) \cup \{(d\sigma,x) : (\sigma,x) \in V(S_{w-1,d-1})\} \cup \{(m_{w,d},d)\}.
\end{aligned}$$

Hence we can conclude that the function $\varphi : V(S_{w,d}) \to V(S_{w,d-1} \otimes S_{w-1,d-1})$ acting as follows

$$\varphi(\sigma,x) = \begin{cases} (\sigma,x), & \sigma \in \Sigma_{w,d-1} \\ (\sigma',x), & \sigma' \in \Sigma_{w-1,d-1}, \sigma = d\sigma',\ x \in \sigma' \\ (m_{w,d},d), & x = d \end{cases}$$

is a bijection.

Now it remains to prove that the mapping induced by φ between the sets $E(S_{w,d})$ and $E(S_{w,d-1} \otimes S_{w-1,d-1})$ is also a bijection. We will divide the edges of $S_{w,d}$ into five categories and consider the categories separately.

- $((\sigma,x_1)(\sigma,x_2)) \in E(S_{w,d})$, $\sigma = \tau x_2 x_1 \rho$ and the first element of σ is not d. By Theorem 6 we obtain $\sigma \in \Sigma_{w,d-1}$ and following the definition of the mapping φ we see that

  $$\varphi((\sigma,x_1)(\sigma,x_2)) = ((\sigma,x_1)(\sigma,x_2)) \in (V(S_{w,d-1}))^2.$$

  As $\sigma = \tau x_2 x_1 \rho$, we obtain $((\sigma,x_1)(\sigma,x_2)) \in E(S_{w,d-1})$. Note also that this way we get all the vertical edges of the graph $S_{w,d-1}$.

- $((\sigma,x_1)(\sigma,x_2)) \in E(S_{w,d})$, $\sigma = \tau x_2 x_1 \rho$ and the first element of σ is d. This case has two sub-cases.

  ✶ $\tau = \lambda$, $x_2 = d$. As $(\sigma,x_2) \in V(S_{w,d})$, by the definition of the set $V(S_{w,d})$ we have $d = x_2 \in \sigma$. For this case Lemma 10 implies $\sigma = m_{w,d}$. Hence, $x_1 = d-1$ or $x_1 = 0$, which can be the case

iff $w = 1$. We claim that the image of the edge $((\sigma, x_1)(\sigma, x_2))$ under φ is the edge connecting the root of $S_{w-1,d-1}$ and the new vertex g. From our definition of φ we know that $\varphi(\sigma, x_1) = (m_{w-1,d-1}, x_1)$ and $\varphi(\sigma, x_2) = g$. The vertex $(m_{w-1,d-1}, x_1)$ is the root of the graph $S_{w-1,d-1}$. Indeed, for the case $x_1 = d-1$ we use the induction hypothesis. For $x_1 = 0$ and $w = 1$ we simply have $(m_{w-1,d-1}, x_1) = (0, 0)$ and $S_{w-1,d-1} = I$. We have proven the claim.

$\star$ $\tau \neq \lambda$. Let $\tau = d\tau'$ and $\sigma = d\sigma'$, then $x_1, x_2 \in \sigma' \in \Sigma_{w-1,d-1}$. Hence

$$\varphi((\sigma, x_1)(\sigma, x_2)) = ((\sigma', x_1)(\sigma', x_2)) \in (V(S_{w-1,d-1}))^2.$$

As $\sigma' = \tau' x_2 x_1 \rho$, we see that $((\sigma', x_1)(\sigma', x_2)) \in E(S_{w-1,d-1})$. Note also that this way we get all the vertical edges of the graph $S_{w-1,d-1}$.

- $((\tau x\rho, x)(\tau y\pi, y)) \in E(S_{w,d})$, $y = d$, $x = d-1$. This implies $\tau = \lambda$ and $y = d \in c_{w,d}(\tau d\pi)$. Hence, by Lemma 10 we have $\tau d\pi = d\pi = m_{w,d}$. Following the definition of φ, we see that $\varphi(\tau y\pi, y) = g$. We claim that the image of the edge $((\tau x\rho, x)(\tau y\pi, y))$ under φ is the edge connecting the root of the graph $S_{w,d-1}$ and the new vertex g. So, it only remains to prove that the image of the vertex $(\tau x\rho, x)$ is the root of the graph $S_{w,d-1}$. By induction hypothesis, we need to show that the equality $(\tau x\rho, x) = (m_{w,d-1}, (d-1))$ holds. As $(\tau x\rho, x) \in V(S_{w,d})$, we know that $x \in c_{w,d}(\tau x\rho)$. But as $\tau = \lambda$ and $x = d-1$, Lemma 10 implies the necessary condition $(\tau x\rho, x) = (m_{w.d-1}, (d-1))$ and the claim is proven.

- $((\tau x\rho, x)(\tau y\pi, y)) \in E(S_{w,d})$, $y = x+1$ and $\tau \neq \lambda$ does not start with d. Then we claim that the edges $\varphi((\tau x\rho, x)(\tau y\pi, y))$ are exactly all the horizontal edges of the graph $S_{w,d-1}$. First note that as τ does not start with d (but with something less), we have $\tau x\rho, \tau y\pi \in \Sigma_{w,d-1}$ and consequently

$$\varphi((\tau x\rho, x)(\tau y\pi, y)) = ((\tau x\rho, x)(\tau y\pi, y)) \in (V(S_{w,d-1}))^2.$$

As $y = x+1$, by Definition 15 we obtain the required implication $\varphi((\tau x\rho, x)(\tau y\pi, y)) \in E(S_{w,d-1})$.

- $((\tau x\rho, x)(\tau y\pi, y)) \in E(S_{w,d})$, $y = x+1$ and τ starts with d. This implies $x, y < d$ and hence

$$\varphi((\tau x\rho, x)(\tau y\pi, y)) = ((\tau' x\rho, x)(\tau' y\pi, y)) \in (V(S_{w-1,d-1}))^2,$$

where $\tau = d\tau'$. As $y = x+1$, we obtain all the horizontal edges of the graph $S_{w-1,d-1}$ this way.

Hence, we have proven a one-to-one correspondence between the sets $E(S_{w,d})$ and $E(S_{w,d-1} \otimes S_{w-1,d-1})$ as well, thus concluding the proof. □

5.2 The algorithm

In this section we introduce an algorithm for building the graphs $S_{w,d}$. We will present the algorithm in three steps:

1) generation of all the vectors of the set $\Sigma_{w,d}$ in their lexicographic order;

2) finding the elements of the sets $c_{w,d}(\sigma)$ (and hence creating the set $V(S_{w,d})$);

3) accomplishing the hash steps represented by the edges (and hence completing the graph $S_{w,d}$).

It will be convenient to have all the vectors of the same length, so we will pad all the vectors having length less than $l = \min\{w+1, d+1\}$ (which is the maximal length of the elements of $\Sigma_{w,d}$) with 0s at the end. We also introduce l integer variables $\sigma_1, \sigma_2, \ldots, \sigma_l$ and consider them as elements of σ, thus $\sigma = \sigma_1\sigma_2\ldots\sigma_l$. Now we claim that Algorithm 1 produces all the elements of the set $\Sigma_{w,d}$ in lexicographic order.

In order to prove the correctness of the algorithm, we need the following lemma describing consecutive vectors of the set $\Sigma_{w,d}$ (which we still consider as padded with 0s).

Lemma 11 *Let the vector $\sigma \in \Sigma_{w,d}$ be represented in the form*

$$\sigma = \rho(x-1)(x-2)\ldots y0\ldots 0,$$

where $x, y \in \{1, 2, \ldots, d+1\}$ and the substring $(x-1)(x-2)\ldots y$ is as long as possible (if $\sigma = 00\ldots0$ then $\rho = (x-1)(x-2)\ldots y = \lambda$). If $y = 1$ or $|\rho(x-1)(x-2)\ldots y| = w$ then the vector directly succeeding σ in terms of the order $\preceq$ is $\tau = \rho x0\ldots0$, otherwise it is $\tau = \rho(x-1)(x-2)\ldots y10\ldots0$.

Proof. Consider first the case $y = 1 \vee |\rho(x-1)(x-2)\ldots y| = w$. Note that maximality of the substring $(x-1)(x-2)\ldots y$ implies that ρ does not end with x and hence $\tau = \rho x0\ldots0 \in \Sigma_{w,d}$ in this case. We also see

Algorithm 1 Generate the vectors of the set $\Sigma_{w,d}$

Require: $w > 0, d > 0$

Set $l := \min\{w+1, d+1\}$
Set $\sigma := \sigma_1\sigma_2 \ldots \sigma_l = 00 \ldots 0$
for $i = 1$ to $|\Sigma_{w,d}| - 1$ **do**
  Output σ
  Set j to be the least index such that $\sigma_j = 0$
  if $j = l$ **then**
    Set $j := j - 1$
  end if
  while $j > 1 \,\&\, \sigma_j = \sigma_{j-1} - 1$ **do**
    Set $\sigma_j := 0$
    Set $j := j - 1$
  end while
  Set $\sigma_j := \sigma_j + 1$
  Reset σ
end for
Output σ

that $\sigma \preceq \tau$, hence it remains to prove that there can be no vectors between them. Suppose on the contrary that such a vector exists. It clearly must begin with $\rho(x-1)(x-2)\ldots$. It is not possible to increase any element in the part $(x-1)(x-2)\ldots y$ as it consists of consecutive elements, all the elements in the vector must be unique and in this part less than x. Hence the only way to create a vector between σ and τ is to append something smaller than y at the end of this part. But this is not possible as we have one of two cases: either $y = 1$ or the vector $\rho(x-1)(x-2)\ldots y0$ already has the maximal allowed length $w+1$.

Now consider the other case $y \neq 1 \,\&\, |\rho(x-1)(x-2)\ldots y| < w$ (which includes the case $\sigma = 00\ldots0$). Reasoning exactly the way we did in the previous case, we see that $\tau = \rho(x-1)(x-2)\ldots y10\ldots0 \in \Sigma_{w,d}$, $\sigma \preceq \tau$ and that there can be no vectors between them. □

Now we can explain why Algorithm 1 generates all the elements of the set $\Sigma_{w,d}$ in lexicographic order. As the algorithm starts with the least vector $00\ldots0$, it is enough to prove that each run of the algorithm (i.e. each step in the for-cycle), taking vector σ as input, outputs its immediate successor. Note that the algorithm makes $|\Sigma_{w,d}| - 1$ runs, so the output of the last run is exactly the greatest vector $m_{w,d}$ (see Theorem 5).

Of course we must know the value $|\Sigma_{w,d}|$ beforehand. This value can be computed from Theorems 2 and 3 using the formula (see Section 5.3 for efficiency considerations concerning this formula)

$$|\Sigma_{w,d}| = \begin{cases} 2^d, & w \geq d \\ \sum_{k=0}^{w} \binom{d}{k}, & w < d \end{cases} . \tag{5.1}$$

Now consider one run of the algorithm with input σ. Following Lemma 11, in order to generate its immediate successor we first have to find the representation of the vector σ in the form $\rho(x-1)(x-2)\dots y0\dots 0$ (where the part $(x-1)(x-2)\dots y$ has maximal length possible). If $y = 1$ or $|\rho(x-1)(x-2)\dots y| = w$ we must replace the part $(x-1)(x-2)\dots y$ with $x00\dots 0$ and otherwise just increase the first 0 by 1. Note that the latter is exactly the same operation as the first one, if we consider the part $(x-1)(x-2)\dots y$ to be just the first 0.

In order to perform the necessary changes, we must find the first 0 in σ (line 5), as that is the last position where the change can occur. In what follows, j will be the counter indicating the current position in σ.

If σ has maximal allowed length l (i.e. $j = l$), we know we can not change the last 0 of the vector, so we must start at the position $l-1$. This is what the lines 6-8 do.

The essential part of the algorithm is the while-loop on the lines 9-12. If we haven't reached the beginning of the vector yet (i.e. $j > 1$) and the current element is the predecessor of the element just before it, we are still on the part $(x-1)(x-2)\dots y$. We set the current position to 0 and move a step towards the beginning. If we reach the beginning of the vector or the beginning of the part $(x-1)(x-2)\dots y$, we stop the loop and increase the current element (which is equal to x) by 1 (line 13). If either

1) the first 0 was discovered at the position 1, or

2) the first 0 was discovered at a position later than 1 and earlier than l, but the element just before it is greater than 1,

we just need to increase this first 0 to 1. In this case the while-loop is not entered at all and the increase is once again performed on line 13.

Now we have created the next σ and we can take the next run of the algorithm. The algorithm is finished by outputting the result of the last run, which we know, equals $m_{w,d}$.

Now we add vertex set generation to Algorithm 1, which by Definition 15 means generating the sets $c_{w,d}(\sigma)$. They can be generated at the same time when producing at the next σ in one run of Algorithm 1.

Algorithm 2 Generate the vertex set of the graph $S_{w,d}$

Require: $w > 0, d > 0$

Set $l := \min\{w+1, d+1\}$
Set $\sigma := \sigma_1\sigma_2 \dots \sigma_l = 00 \dots 0$
for $i = 1$ to $|\Sigma_{w,d}|$ **do**
  Set $c_{w,d}(\sigma) = \emptyset$
  Set j to be the least index such that $\sigma_j = 0$
  Include the element σ_j to the set $c_{w,d}(\sigma)$
  if $j = l$ **then**
    Set $j := j - 1$
  end if
  while $j > 1 \,\&\, \sigma_j = \sigma_{j-1} - 1$ **do**
    Set $\sigma_j := 0$
    Set $j := j - 1$
    Include the element σ_j to the set $c_{w,d}(\sigma)$
  end while
  Set $\sigma_j := \sigma_j + 1$
  Output the set $c_{w,d}(\sigma)$ and reset σ
end for

Consider Algorithm 2. Note that Algorithm 2 runs one more time than Algorithm 1 does. The reason is that we also want to generate the set $c_{w,d}(\sigma)$ for the last vector $m_{w,d}$ as well.

In order to prove that Algorithm 2 generates the correct set $c_{w,d}(\sigma)$, write σ as above in the form $\rho(x-1)(x-2)\dots y0\dots 0$ (where the part $(x-1)(x-2)\dots y$ is as long as possible). We need to show that if the condition $(y = 1) \vee (|\rho(x-1)(x-2)\dots y0| = w+1)$ holds then the elements $x-1, x-2, \dots, y, 0$ are included into the set $c_{w,d}(\sigma)$ and otherwise $c_{w,d}(\sigma) = \{0\}$ (see Lemma 10).

As follows from the proof of Algorithm 1 presented above, the elements $x-1, x-2, \dots, y, 0$ (or just 0, if $y > 1$ and $|\rho(x-1)(x-2)\dots y0| < w+1$) are exactly the ones set to 0 or increased by 1. This means that we must add an element of σ to the set $c_{w,d}(\sigma)$ every time right before we set it to 0 or increase by 1 – and this is exactly what Algorithm 2 does.

In order to complete the algorithm of generation of the graph $S_{w,d}$ we still need to show how to draw the edges. As we remember from Definition 15, the edges can be of two kinds – vertical and horizontal. With vertical edges there should be no problems, as they are drawn inside one set $c(\sigma_{w,d})$, i.e. during one run of our algorithm. Horizontal edges can cause more problems as in order to complete a horizontal edge we need to know both

its end-vertices. So it is necessary to keep some information about the started edges over several runs of the algorithm.

What kind of information is needed? Going back to the original motivation behind the trees $\mathfrak{S}_w^d$, we see that these trees are used to represent certain hash computations and actually we are only interested in the hash value attached to the root of the tree. In order to carry the necessary hash values along the computations, we introduce l new variables $h_1, h_2, \ldots, h_l$ and let h_i carry the last hash value attached to a vertex of the form (σ, σ_i).

Let H be the hash function used for hash computations and consider Algorithm 3.

Algorithm 3 Create the hash-edges of the graph $S_{w,d}$

Require: $w > 0, d > 0$

1: Set $l := \min\{w + 1, d + 1\}$
2: Set $\sigma_1 = \sigma_2 = \ldots = \sigma_l := 0$
3: **for** $i = 1$ to $|\Sigma_{w,d}|$ **do**
4: Set j to be the least index such that $\sigma_j = 0$
5: Set h_j to be the next input data item
6: **if** $j = l$ **then**
7: Set $j := j - 1$
8: Compute $h_j = H(h_j, h_{j+1})$
9: **end if**
10: **while** $j > 1 \,\&\, \sigma_j = \sigma_{j-1} - 1$ **do**
11: Set $\sigma_j := 0$
12: Set $j := j - 1$
13: Compute $h_j = H(h_j, h_{j+1})$
14: **end while**
15: Set $\sigma_j := \sigma_j + 1$
16: **end for**
17: Return h_1

The vertices of the graphs $\mathfrak{S}_w^d$ are divided into two subsets: vertices representing data items (leaves of the tree) and vertices representing the computations. At each round exactly one data item is added and in the construction of the tree $S_{w,d}$ it must correspond to the leaf, i.e. vertex $(\sigma, 0)$. This is expressed on line 5 of the algorithm.

All the other vertices we add represent hash computations. Hash computations on lines 8 and 13 carry exactly the same character. As soon as the algorithm has decided to move one step towards the beginning of the vector σ, the hash value corresponding to the new location in σ (or the vertex of the graph $S_{w,d}$) is replaced by the hash of the value at the previous location

and the old value at the new location.

By Definition 15, there are two vertices that are sources for the edges ending in the vertex $(\rho x(x-1)\tau, x)$, they are $(\rho(x-1)\tau', x-1)$ for some vector τ' and $(\rho x(x-1)\tau, x-1)$ (we assume $x > 0$, which is exactly the case with non-data-item vertices). Hence, it remains to prove that the last two vertices have the correct hash values attached to them. Let x be the ith element of the vector $\rho x(x-1)\tau$.

Consider first the vectors $\sigma = \rho x(x-1)\tau$ and $\sigma' = \rho(x-1)\tau'$. As the first part, ρ, is the same, it was not changed between the generation rounds of vectors σ' and σ. Moreover, the ith element was last changed at the time of generation of σ', as x and $x-1$ differ by 1 and Algorithm 3 changes elements of the vectors of $\Sigma_{w,d}$ by 1 at a time. Consequently the previous value of h_i, when processing the vertex $(\rho x(x-1)\tau, x)$, comes from the correct vertex $(\rho(x-1)\tau', x-1)$.

At last, consider the vertices $(\rho x(x-1)\tau, x)$ and $(\rho x(x-1)\tau, x-1)$. Following the algorithm we see that the vertex $(\rho x(x-1)\tau, x-1)$ was processed just before $(\rho x(x-1)\tau, x)$, hence the hash value attached to it is h_{i+1}, which is the correct one.

Recalling that the root of the tree $S_{w,d}$ is the vertex is $(m_{w,d}, d)$ (see Theorem 7) we see that the algorithm must output the last value of h_1 after the last step. As this is exactly the action taken on line 17, we conclude that Algorithm 3 represents the hash computations of the graph $S_{w,d} \simeq \mathfrak{S}_w^d$ correctly.

5.3 Efficiency and further optimizations

Despite its complicated look, Algorithm 3 is very efficient. The algorithm runs $|\Sigma_{w,d}|$ times and on each run $|c_{w,d}(\sigma)|$ steps are made. Hence the obvious estimate to the complexity of the algorithm is $O(|V(S_{w,d})|)$. Even more, the operations used in Algorithm 3 are "cheap": the only operations used are additions-subtractions by 1 and hash computations (where the latter ones can not be avoided anyway). Of course, we still need to take care about the search directive on line 4 of Algorithm 3 that just states: Set j to be the least index such that $\sigma_j = 0$. This search can be done in $\log_2 l$ steps, but it is also possible to introduce one extra variable and modify the algorithm so that at the end of run it is set to the least 0 of the newly generated σ.

Memory requirements of our algorithm are very low as well. We need to store the following data in order to restore the computations after the server's crash:

- l, j and $|\Sigma_{w,d}|$;
- $h_1, \ldots, h_l$ for hash values;
- $\sigma_1, \ldots, \sigma_l$ as counters.

As the counters σ_j can contain values up to d, the corresponding required storage space for the values of h_i and σ_j is $l \cdot \log_2 d + l \cdot k$, where k is the output length of the hash function h. A recent unpublished result by Helger Lipmaa [Lip02] shows that this requirement can actually be decreased to $\log_2 d + l \cdot k$ by using encoding of the leaves presented in Section 3.4.

One must also ask, how much resources does it take to compute $|\Sigma_{w,d}|$ and how much storage space does this value need. Formula (5.1) does not look promising at the first sight as it contains a sum of binomial coefficients. Still, if we are satisfied with the estimate $\beta(G) \approx 1.5$, we can use the trees $\mathfrak{T}_w = \mathfrak{S}_w^{2w+1}$ from Section 3.6. For these graphs $|\Sigma_{w,2w+1}| = 2^{2w}$ that can be computed very efficiently in binary format. But if we want the asymptotically optimal schemes described in Chapter 4 we can change the algorithm a little and substitute the for-cycle in rows 3-16 of Algorithm 3 with a while-loop working if $\sigma_1 < d+1$. The correctness of this substitution is justified by the observation that on its last run the algorithm changes the vector $\sigma = m_{w,d}$ to the vector $(d+1)00\ldots0$. Note that such a modification enables us to replace the need to store (a relatively large) value of $|\Sigma_{w,d}|$ with the need to store only the value of d.

6 INSTEAD OF THE CONCLUSIONS

Almost every PhD thesis starts off with the author's dream to achieve something new (preferably revolutionary) and useful (preferably something that could be sold right away). Of course there are exceptions, but I hope they are rare. Why? Mainly because I believe that ambition is the most important cause of every great discovery made on Earth and hence there is very little hope to achieve anything without any ambitions.

Does this mean that every thesis achieves the high goals set by the author (with the help of supervisor, of course)? Not at all, as it takes something more to come up with revolutionary results than just the goals themselves. This "something more" is hard to define, but for sure it contains

- a bit of talent needed to see deep under the core of things,
- a bit of luck to find the right problems to look at, and
- a lot of hard work (lasting approximately for 200 days without eating nor sleeping in the case of Tartu University, Estonia).

Have I had all the necessary components? Most of them probably yes. Talent and luck are of course difficult to measure, but at least the goals were decent and the work was hard (well, I admit I ate and slept from time to time, but the period of writing lasted considerably longer than just 200 days).

How well have the original goals been met and what is the actual practical value of the thesis? To the first question, the answer can be stated rather clearly and it consists of several parts.

- The thesis identified the need to look at the two separate time-stamping scenarios: patents and digital signatures. It turned out that linking schemes providing size-optimal time stamps for one scenario are not optimal for another.
- It was shown that the size of time stamps can be estimated from above by the value $\beta(G)$. It is not the best possible estimate but by our current state of knowledge, the exact expression of the size of time stamps is too complicated to deal with. Maybe one day ...
- We proved that the optimal value of $\beta(G)$ is

$$\frac{1}{\log_2\left(\frac{1+\sqrt{5}}{2}\right)} \approx 1.44042$$

and it is approached (but never achieved as this value is irrational) with the family of trees $\mathfrak{G}_w^d$. The effect gained in comparison with the previous best-known BLS-scheme is 28% which is quite a remarkable amount. Even more importantly we showed that just by estimating the value of $\beta(G)$ the result can not be improved any more. Of course, if one day the tools of dealing with the actual the size of time stamps become available, the estimate may be improved some more, but the author's wild guess is that not too much. At the moment, it seems that the asymptotic size of time-stamps remains the same as mentioned above.

- It turned out that the original definition of the optimum-providing family was not too suitable for actual implementation of a TSA. In order to improve the situation, the idea of representing the current state of computations with some simple and efficient encoding was proposed and one possible encoding designed.

The question of practical applicability is a bit more complicated. Of course, we can build nice tools and try to sell them but people will only buy them if they need to. Do they need time-stamping? At least in Estonia they do as the Law of Digital Signatures requires it. But do the users actually need linking? The answer is unfortunately – probably not at the moment.

Going back to Chapter 1, we recall that the easiest way to implement time-stamping is to let the TSA just sign the requests together with physical time. The solution is of course totally insecure as the TSA must be unconditionally trusted. Is this a serious obstacle keeping people from using this solution? No, it is not. Looking at the atmosphere of suspicion that we can see every day between different politicians, businessmen and even nations, it is hard to believe how trusting people are deep in their hearts. Why not to declare an authority trustworthy, if such a declaration saves us from the trouble of setting up linkage-based time-stamping! And there is really not much to do in order to change the human mind. The best solution is probably to wait, see and hope that after the first few incidents with cheating TSAs the need for more secure solutions arises.

But before that computer scientists all over the world still have some time to search for better solutions to propose when they will really be needed. And in that light I can say I do not regret writing this thesis even a bit.

REFERENCES

[ABRW01] Arne Ansper, Ahto Buldas, Meelis Roos, and Jan Willemson. Efficient long-term validation of digital signatures. In *Public Key Cryptography - PKC'2001*, volume 1992 of *LNCS*, pages 402–415, February 2001.

[ABSW01] Arne Ansper, Ahto Buldas, Märt Saarepera, and Jan Willemson. Improving the availiability of time-stamping services. In Vijay Varadharajan and Yi Mu, editors, *6th Australasian Conference, ACISP 2001*, volume 2119 of *LNCS*, pages 360–375, Sydney, Australia, July 2001. Springer-Verlag.

[ACPZ01] C. Adams, P. Cain, D. Pinkas, and R. Zuccherato. RFC3161: Time Stamp Protocol (TSP). August 2001.

[BdM91] Josh Benaloh and Michael de Mare. Efficient broadcast time-stamping. Technical Report 1, Clarkson University Department of Mathematics and Computer Science, August 1991.

[BHS93] Dave Bayer, Stuart Haber, and W. Scott Stornetta. Improving the efficiency and reliability of digital time-stamping. In *Sequences II: Methods in Communication, Security, and Computer Science*, pages 329–334. Springer-Verlag, 1993.

[BL98] Ahto Buldas and Peeter Laud. New linking schemes for digital time-stamping. In *Proc. 1st International Conference on Information Security and Cryptology – ICISC'98*, pages 3–13, Seoul, Korea, December 1998.

[BLL00] Ahto Buldas, Peeter Laud, and Helger Lipmaa. Accountable Certificate Management using Undeniable Attestations. In Sushil Jajodia and Pierangela Samarati, editors, *7th ACM Conference on Computer and Communications Security*, pages 9–18. ACM Press, November 2000.

[BLLV98] Ahto Buldas, Peeter Laud, Helger Lipmaa, and Jan Villemson. Time-stamping with binary linking schemes. In *Advances in Cryptology – CRYPTO'98*, volume 1462 of *LNCS*, pages 486–501, Santa Barbara, 1998. Springer-Verlag.

[BLS00] Ahto Buldas, Helger Lipmaa, and Berry Schoenmakers. Optimally efficient accountable time-stamping. In *Public Key Cryptography – PKC'2000*, volume 1751 of *LNCS*, pages 293–305, Melbourne, Australia, January 2000. Springer-Verlag.

[BRW02] Ahto Buldas, Meelis Roos, and Jan Willemson. Undeniable replies for database queries. To appear in the proceedings of Fifth International Baltic Conference on DB and IS, June 2002.

[BW01a] Ahto Buldas and Jan Willemson. A new linking scheme for interval time stamps. Manuscript, available from the authors, 2001.

[BW01b] Ahto Buldas and Jan Willemson. On interval time stamps of minimum size. Manuscript, available from the authors, 2001.

[Chr75] Nicos Christofides. *Graph theory: an algorithmic approach.* Academic Press, New York, London, San Francisco, 1975.

[GKP89] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics.* Addison-Wesley, Reading, MA, 1989.

[Har69] Frank Harary. *Graph theory.* Addison-Wesley, 1969.

[HS91] Stuart Haber and W.Scott Stornetta. How to time-stamp a digital document. *Journal of Cryptology*, 3(2):99–111, 1991.

[HS97] Stuart Haber and W.Scott Stornetta. Secure names for bit-strings. In *Proc. 4th ACM Conference on Computer and Communications Security*, 1997.

[Koh78] Loren M. Kohnfelder. Toward a practical public-key cryptosystem. 1978.

[Lip99] Helger Lipmaa. *Secure and efficient time-stamping schemes.* PhD thesis, Tartu University, 1999.

[Lip02] Helger Lipmaa. On Optimal Hash Tree Traversal. Manuscript, available from the author. First presented during the Estonian Winter School on Computer Science, Palmse, Estonia, on March 4th, 2002.

[MAM+99] Michael Myers, R. Ankney, A. Malpani, S. Galperin, and Carlisle Adams. RFC2560: X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. June 1999.

[Mer80] Ralph C. Merkle. Protocols for public key cryptosystems. In *Proceedings of the 1980 IEEE Symposium on Security and Privacy*, pages 122–134, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, April 1980. IEEE Computer Society Press.

[MvOV97] Alfred J. Menezes, Paul C. van Oorshot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, Boca Raton, New York, London, Tokyo, 1997.

[NIS95] NIST. Secure hash standard. Federal Information Processing Standards Publication 180-1, April 1995.

[NIS00] NIST. Digital Signature Standard (DSS). Federal Information Processing Standards Publication 186-2, January 2000.

[NIS01] NIST. Secure hash standard. Federal Information Processing Standards Publication 180-2, May 2001.

[PF96] Fernando Pinto and Vasco Freitas. Digital time-stamping to support non repudiation in electronic communications. In *Proc. SECURICOM'96 - 14th worldwide Congress on Computer and Communications Security and Protection*, CNIT, pages 397–406, Paris, June 1996.

[Pfi96] Birgit Pfitzmann. *Digital Signature Schemes*, volume 1100 of *LNCS*. Springer-Verlag, Heidelberg, August 1996.

[Pre93] Bart Preneel. *Analysis and design of cryptographic hash functions.* PhD thesis, Katholieke Universiteit Leuven (Belgium), January 1993.

[PSST01] A. Perrig, S.W. Smith, D. Song, and J.D. Tygar. SAM: A Flexible and Secure Auction Architecture using Trusted Hardware. In *ICEC01: First International Workshop on Internet Computing and Electronic Commerce*, April 2001.

[RG95] T. E. Rockoff and M. P. Groves. Design of an Internet-Based System for Remote Dutch Auctions. *Internet Research: Electronic Networking Applications and Policy*, 5(4):10–16, 1995.

[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):120–126, February 1978.

[Sch35] Erwin Schrödinger. Die gegenwärtige Situation in der Quantenmechanik. *Die Naturwissenschaften*, 23:807–812; 823–828; 844–849, 1935.

[Sti95] Douglas R. Stinson. *Cryptography: Theory and Practice.* CRC Press, Boca Raton, New York, London, Tokyo, 1995.

[Sur] Surety digital notary and timestamping service homepage. Available at `http://www.surety.com`.

[Tri80] John D. Trimmer. The Present Situation In Quantum Mechanics. *Proceedings of the American Philosophical Society*, 124:323–338, 1980.

[Wil01a] Jan Willemson. An algorithm for building efficient timestamping schemes on the fly. Manuscript, available from the author, 2001.

[Wil01b] Jan Willemson. Optimal trees for interval time stamps. Manuscript, available from the author, 2001.

INDEX

SUURUSE MÕTTES EFEKTIIVSED INTERVALLAJATEMPLID

Sisukokkuvõte

Erinevates digitaalse asjaajamise rakendustes tekib vajadus määrata digitaalse informatsiooni erinevaid parameetreid – millises vormingus on info esitatud, kes ja kuna dokumendi lõi jne. Enamasti pole andmete tekke täpset aega võimalik kindlaks teha, kasvõi sel põhjusel, et dokumendi looja ei pruugi loomise aktist aastaid teatada. Nii asendataksegi digitaalse informatsiooni *tekkeaja* kindlakstegemine tavaliselt *registreerimisaja* fikseerimisega, nõudes, et dokumendi looja peab dokumendi registreerima selleks ette nähtud autoriteedi juures. Niisugust protseduuri nimetatakse ajatembelduseks ja autoriteeti ajatempliteenuse osutajaks.

On olemas rida stsenaariume, mille korral andmete ühekordsest registreerimisest piisab. Töös nimetatakse seda tinglikult *patendistsenaariumiks*, pidades silmas võimalikku rakendust patendivaidluste lahendamisel, kus tuleb kindlaks teha, kes oma leiutisest esimesena teada andis. Sama ideoloogia abil saab ka välja selgitada, kes reserveeris esimesena lennukipileti jne.

Digitaalallkirjade tekkeaja kindlakstegemisel on olukord aga põhimõtteliselt teistsugune. Nimelt on dokumendi digitaalne signeerimine seotud privaatvõtme kasutamisega ja see operatsioon tuleb läbi viia võtmeomaniku täieliku kontrolli all. Niisiis ei saa dokumendi allkirjastamise täpset momenti mingi kolmanda osapoole juures fikseerida. Küll aga saab registreerida kaks ajahetke – ühe kindlalt enne signeerimist ja teise kindlalt pärast seda. Nii võime hiljem väita, et elektronallkiri on antud mingi kindla ajaintervalli jooksul ning sellest johtuvalt nimetatakse kirjeldatud ajatembelduse vormi *intervallajatembelduseks*.

Käesoleva doktoritöö eesmärk on uurida intervallajatembeldusskeeme, mis võimaldaksid ajatemplite suuruse miinimumini viia. Väitekirjas formaliseeritakse vastav optimeerimisülesanne, antakse ajatemplite suurusele (küllalt täpne) ülemine hinnang, leitakse antud hinnagu jaoks alampiir ning näidatakse ära graafidepere, mis lähendab seda piiri kuitahes hästi (kusjuures täpne piir on saavutamatu).

Töö viimane osa on pühendatud väljatöötatud graafipere praktilise realiseerimise küsimustele ajatempliteenuse osutaja serveris. Algse rekurrentse graafipere definitsiooni põhjal on küll võimalik luua vastav rekursiivne algoritm, kuid esiteks ei vasta rekursioon linkimispõhise ajatembelduse ideoloogiale ning teiseks pole naiivne rekursiivne realisatsioon käideldavuse mõttes

turvaline. Lahendusena pakutakse töös välja alternatiivne samm-sammuline algoritm, mis säilitab serveri töö jätkamiseks vajaliku informatsiooni efektiivselt ning varundataval kujul.

ACKNOWLEDGEMENTS

There are many people without whom this thesis would not have been born at all. I guess the first person to blame (or to thank) for dragging me into the topic is Helger Lipmaa who asked me to join Cybernetica in 1997/98 and introduced me to several other beautiful minds. Besides this Helger deserves some extra words of gratitude for reading preliminary versions of my thesis and making a number of valuable suggestions.

It would take long to list all the good colleagues in Cybernetica who have influenced me one way or another, and hence I just mention the most important one of them all – my supervisor Ahto Buldas. If I know anything about data security, time-stamping or paper writing at all, it's thanks to him. An important role has also been played by Cybernetica as an organization where beginning scientists and developers have found a fruitful ground to grow.

It is very hard to write a computer science thesis without a computer and its software, thus I would also like to thank the creators of heaps of great free software like Linux operating system, LaTeX text preparation system, Free Pascal Compiler, Nedit editor and others.

And last but not least – I have been happy to have an understanding and supporting family, represented by my wife Kairi. I promise to have more time for her when this thesis will be ready.

CURRICULUM VITAE

Jan Villemson

Citizenship: Estonian Republic
Born: July 30, 1974, Tartu, Estonia
Marital status: married
Address: Ravila 70-5, Tartu, Estonia
Contacts: phone: (+372 7) 302 667, e-mail: jan@ut.ee

Education

1981 – 1992 Tartu Secondary School No. 12
1992 – 1996 BSc, Faculty of Mathematics, Tartu University
1996 – 1998 MSc, Faculty of Mathematics, Tartu University

Professional updating stays

1996 — Eindhoven Technical University (TUE), Holland
1997 — Turku University, Finland
1998 — 3rd Estonian Winter School in Computer Science (EWSCS), Palmse, Estonia
2000 — 5th EWSCS, Palmse, Estonia
2001 — 6th EWSCS, Palmse, Estonia
2001 — EIDMA minicourse on Cryptographic Multiparty Protocols, TUE, Holland
2002 — 7th EWSCS, Palmse, Estonia

Professional employment

1998 – ... Research engineer, Cybernetica
2000 – ... Lecturer, Tartu University

The main fields of interest are combinatorial methods and their applications in data security and digital document management.

Results have been presented at the conferences in Santa Barbara (USA, 1998), New Orleans (USA, 2000), Cheju Island (Korea, 2001), Sydney (Australia, 2001) and Tallinn (Estonia, 2002).

CURRICULUM VITAE

Jan Villemson

Kodakondsus: Eesti
Sünniaeg ja -koht: 30. juuli 1974, Tartu, Eesti
Perekonnaseis: abielus
Aadress: Ravila 70-5, Tartu, Eesti
Kontaktandmed: telefon: (+372 7) 302 667, e-mail: jan@ut.ee

Haridus

1981 – 1992 Tartu 12. Keskkool
1992 – 1996 bakalaureus, Tartu Ülikooli matemaatikateaduskond
1996 – 1998 magister, Tartu Ülikooli matemaatikateaduskond

Erialane enesetäiendus

1996 — Eindhoveni Tehnikaülikool (TUE), Holland
1997 — Turu Ülikool, Soome
1998 — 3. Eesti Arvutiteaduse Talvekool (EATTK), Palmse, Eesti
2000 — 5. EATTK, Palmse, Eesti
2001 — 6. EATTK, Palmse, Eesti
2001 — EIDMA lühikursus "Cryptographic Multiparty Protocols", TUE, Holland
2002 — 7. EATTK, Palmse, Eesti

Erialane teenistuskäik

1998 – ... teadur, Cybernetica
2000 – ... lektor, Tartu Ülikool

Peamisteks tegevusvaldkondadeks on kombinatoorsed meetodid ning nende rakendused andmeturbes ja digitaalses dokumendihalduses.

Tulemused on publitseeritud konverentsidel Santa Barbaras (USA, 1998), New Orleansis (USA, 2000), Cheju Island'il (Korea, 2001), Sydneys (Austraalia, 2001) ja Tallinnas (Eesti, 2002).

DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS

1. Mati Heinloo. The design of nonhomogeneous spherical vessels, cylindrical tubes and circular discs. Tartu, 1991. 23 p.
2. Boris Komrakov. Primitive actions and the Sophus Lie problem. Tartu, 1991. 14 p.
3. Jaak Heinloo. Phenomenological (continuum) theory of turbulence. Tartu, 1992. 47 p.
4. Ants Tauts. Infinite formulae in intuitionistic logic of higher order. Tartu, 1992. 15 p.
5. Tarmo Soomere. Kinetic theory of Rossby waves. Tartu, 1992. 32 p.
6. Jüri Majak. Optimization of plastic axisymmetric plates and shells in the case of Von Mises yield condition. Tartu, 1992. 32 p.
7. Ants Aasma. Matrix transformations of summability and absolute summability fields of matrix methods. Tartu, 1993. 32 p.
8. Helle Hein. Optimization of plastic axisymmetric plates and shells with piece-wise constant thickness. Tartu, 1993. 28 p.
9. Toomas Kiho. Study of optimality of iterated Lavrentiev method and its generalizations. Tartu, 1994. 23 p.
10. Arne Kokk. Joint spectral theory and extension of non-trivial multiplicative linear functionals. Tartu, 1995. 165 p.
11. Toomas Lepikult. Automated calculation of dynamically loaded rigid-plastic structures. Tartu, 1995. 93 p. (in russian)
12. Sander Hannus. Parametrical optimization of the plastic cylindrical shells by taking into account geometrical and physical nonlinearities. Tartu, 1995. 74 p.
13. Sergei Tupailo. Hilbert's epsilon-symbol in predicative subsystems of analysis. Tartu, 1996. 134 p.
14. Enno Saks. Analysis and optimization of elastic-plastic shafts in torsion. Tartu, 1996. 96 p.
15. Valdis Laan. Pullbacks and flatness properties of acts. Tartu, 1999. 90 p.
16. Märt Põldvere. Subspaces of Banach spaces having Phelps' uniqueness property. Tartu, 1999. 74 p.

17. Jelena Ausekle. Compactness of operators in Lorentz and Orlicz sequence spaces. Tartu, 1999. 72 p.
18. Krista Fischer. Structural mean models for analyzing the effect of compliance in clinical trials. Tartu, 1999. 124 p.
19. Helger Lipmaa. Secure and efficient time-stamping systems. Tartu, 1999. 56 p.
20. Jüri Lember. Consistency of empirical k-centres. Tartu, 1999. 148 p.
21. Ella Puman. Optimization of plastic conical shells. Tartu, 2000. 102 p.
22. Kaili Müürisep. Eesti keele arvutigrammatika: süntaks. Tartu, 2000. 107 lk.
23. Varmo Vene. Categorical programming with inductive and coinductive types. Tartu, 2000. 116 p.
24. Olga Sokratova. Ω-rings, their flat and projective acts with some applications. Tartu 2000. 120 p.
25. Maria Zeltser. Investigation of double sequence spaces by soft and hard analitical methods. Tartu 2001. 154 p.
26. Ernst Tungel. Optimization of plastic spherical shells. Tartu 2001. 90 p.
27. Tiina Puolakainen. Eesti keele arvutigrammatika: morfoloogiline ühestamine. Tartu 2001. 138 p.
28. Rainis Haller. $M(r,s)$-inequalities. Tartu 2002. 78 p.

ISSN 1024–4212
ISBN 9985–56–651–3