On the Temporal Authentication of Digital Data

by

Michael K. Just

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

Ottawa-Carleton Institute for Computer Science
School of Computer Science
Carleton University
Ottawa, Ontario

December 1998

The undersigned hereby recommend to
the Faculty of Graduate Studies and Research
acceptance of the thesis,

On the Temporal Authentication of Digital Data

submitted by

Michael K. Just

Prof. Evangelos Kranakis
(Director, School of Computer Science)

Prof. Evangelos Kranakis
(Thesis Co-Supervisor)

Dr. Paul Van Oorschot
(Thesis Co-Supervisor)

Dr. Aviel Rubin
(External Examiner)

Carleton University

December 1998

Abstract

In this thesis, we examine the authentic provision, maintenance and verification of a time associated with data. We begin by assimilating the current techniques for authentically associating a time with digital data, i.e., time stamping protocols. This provides a basis for further classification and examination useful both within the thesis, as specifically related to time stamping and notarization, as well as for the area of digital authentication itself. We introduce the distinction between absolute and relative time stamps and classify the previous work based on this refinement. This work is subsequently analyzed and critiqued with respect to various measures of efficiency.

We define the notion of *temporal authentication*. General techniques for the provision and verification of both absolute and relative temporal authentication are examined. The usefulness of these distinctions is motivated by the discovery of protocol failures for the time stamping protocols of Haber and Stornetta (Journal of Cryptology '91) and of Benaloh and de Mare (Eurocrypt '93).

We analyze the provision of temporal authentication for certificate-based digital signatures. The necessity of time stamping digital signatures is motivated, and protocols for the production, verification and adjudication of time stamped digital signatures are presented. Beyond the time stamping of the signature, the need for the maintenance of temporal storage over the long-term (e.g., for revocation information), in anticipation of possible disputes is also identified as a requirement. Additionally, protocols for notarizing and extending the lifetime of digital signatures are also presented and reviewed.

The time stamping of both signatures and revocation information aids in determining whether a signed message is acceptable or not, e.g., if the message was signed before the corresponding verification key was revoked. However, if the owner of the signing key is unaware of any need for revocation (e.g., resulting from an undetected key compromise), then other solutions are required. We identify the problem of *undetected signature key compromise* and introduce for the first time in the literature, techniques that allow one to maintain the provision of temporal authentication even when a signing key has been compromised. Various techniques for detecting a compromise and preventing forged signature acceptance are proposed.

Acknowledgements

Thanks to Paul Van Oorschot for taking the time to provide extensive, constructive comments on earlier drafts of this thesis. Thanks also for his guidance and support in directing the content, precision and detail of this thesis and his consideration for always responding to my queries in a timely manner.

Many thanks to Evangelos Kranakis for his constant encouragement, advising me to always question what I read. He helped me to surpass one of the larger hurdles encountered when writing a thesis: finding a research topic. His encouragement to search and study the current literature for a topic that I found interesting, allowed me to work on a topic that I wanted to work on and was hence able and willing to pursue and complete. More importantly, he emphasized the importance of thinking about a potential research topic *every day*, whether it be for only five or ten minutes. This constant analysis of a topic allowed me to discover the issues that form the basis of this thesis.

Contents

List of Tables

List of Figures

List of Protocols

Chapter 1

Introduction and Overview

Traditional cryptographic authentication techniques allow for assurances with respect to an action performed (*what* was done) as well as what entity performed the action (*who* did it). The what may be the application of a digital signature such that what was done (e.g., a previous commitment or assertion) cannot be altered without detection. In verifying the maintenance of this integrity, it is necessary to determine who performed the original action.

Equally as important is the ability to recognize *when* something was done. The roles associated with and privileges afforded to an individual entity can change with time. Determining simply what action an entity performed may not be sufficient. It is often important to determine the time at which the action took place. In this thesis, we examine the importance and relevance of time for cryptographic authentication.

Chapter Outline

In this chapter, we motivate the importance of the relationship between time and cryptographic authentication. In Section 1.1, the importance of authentication for both digital and paper-based information processing techniques is examined. The different challenges encountered with digital technology are also discussed, with an introduction to the importance of time. In Section 1.2, the role of time in a cryptographic infrastructure providing for message authentication is motivated. In Section 1.3, we present an overview of the remainder of the thesis. Emphasis is directed towards our particular contributions and as well to the overall contribution of the thesis as a whole to the area of cryptographic authentication.

1.1 From Physical to Digital Data

The world is changing. At one time, a handshake referred only to the clasping of open hands for the purpose of either an introduction, a meeting, or possibly to indicate some sort of agreement. It is now also used to indicate the initiation of a connection between two computing devices.

Computers are certainly having an impact on our everyday lives. With each task they help us to solve, our reliance on them increases. They exist because there are problems that we need solved and we constantly push them to their limit in order to aid in solving newer challenges that are presented to us.

The language is often the same, as with the "handshake" described above. Consider the signature. Our handwritten or physical signature is still used often in a world that is arguably in a transitionary phase, from paper-based to digital. Our signature is used to signify agreement, authorize action or prove membership and may be accepted as legally binding. The powers of authorization given to a signer rely on the assumption that the signature is not easily forgeable.

The parallel in the (arguably superior) digital world is the digital signature. The prevalence of the digital (versus the physical) signature is becoming more noticeable each day. What advantages does a digital signature offer? Realizing that the digital signature is not the cause of the digital revolution, but rather an effect of it, its greatest advantage is that it allows for the provision of a signature for digital data. Why is such a technique required? In a world where paper-based transactions are becoming less frequent, the corresponding physical services become less desirable. It is cheaper to store a disk full of information rather than boxes of paper. It is cheaper to transport digital data than paper-based information. Though certainly influenced by economic concerns, the convenience is equally as responsible.

Familiarity can help to improve convenience. Though dealing with digital media, it is important to offer the same services as those provided for the physical counterpart. This includes the signature. This is why a digital counterpart to the handwritten signature is required. It provides a familiar counterpart to a physical signature, for use with digital data. But how does one apply a signature to digital data? The exact parallel with paper does not work. Consider how one might digitize their physical signature and simply append it to the end of a digital file. However, notice the lack of binding or association between the document data and the digitized signature. There is nothing to stop one from simply copying this digitized signature and appending it to the end of another file, or even altering the file in which the original signature was applied. After all, this is what the legitimate user would do with their digitized signature for subsequent documents anyways.

It is the different media that makes this forgery of the digitized signature achievable. This is the challenge for those wishing to offer digital services in place of physical-based ones. Along with the desireable properties of digital media, are those that make it difficult to offer the familiar, paper-based services. As the example above indicates, digital data

1. is easily transferable as it is no longer attached to a physical medium. Copying is easy, making it difficult to determine which copy is the original.

2. can be modified without detection. Documents on digital media can be modified in any manner, including both the data body as well as any appendage such as a digitized signature.

Note that for our example above, it is not that easy to copy a physical signature. One can typically distinguish an original document as authentic by physical means or appearance. Interestingly though, it is quite serendipitous (from the point of view of digital media proponents) that digital technology is also making tampering of physical data less susceptible to detection, e.g., by using high quality digital photocopiers or scanners. Hence, digital technology is creating a need for digital security techniques.

1.1.1 The Importance of Time

The application of a conventional signature often has legal implications, e.g., it can be viewed as an acceptance of the statements contained therein. Often accompanying the signature is a line indicating the date of the signature. What is the purpose of this date? Suppose a contract were signed by Mr. X, a representative of company C, stating that company C agreed to build an attachment onto the business property of Ms. Y. The contract was signed on March 12, 1998 (though this date was not indicated on the contract). Ms. Y expects that this is sufficient time for the work to be completed by September 1, 1998.

However, subsequent to a disagreement between Mr. X and company C, Mr. X is fired on March 31, 1998. On May 1, 1998, Ms. Y contacts company C, concerned that they have not begun work on her building. Company C representative Ms. Z claims that the purported contract is invalid since it was signed subsequent to the dismissal of Mr. X (possibly by a vindictive Mr. X). Ms. Y claims that the contract was signed prior to the dismissal of Mr. X and demands that company C honour their contract. Other physical evidence may exist to resolve this dispute. For example, physical analyses of the paper and pen markings may ultimately prove that the contract was indeed signed prior to the dismissal. However, there exists the intolerable possibility for Ms. Y that the contract may be deemed invalid.

For a digital contract, such physical evidence is less likely to exist. The dating provides a potential solution for the paper contract, but simply appending the time to the end of the digital contract may not be sufficient. However, by including a time as part of the original document and signing the entire combination, digital authentication of the contract can be provided.

Beyond the inclusion of the date and signing of the contract, additional properties must be met in order to ensure the authenticity of the contract:

1. Mr. X must be who he claims to be;

2. The correct 'date of signing' must be indicated on the contract;

3. Mr. X must work for company C as of the time of signing of the contract.

The first point requires an authorization (certification) of Mr. X's signature privilege and can be provided for the physical signature by requiring Mr. X, for example, to possess a universally verifiable validation card identifying Mr. X as an employee of company C and containing a reference copy of his signature, allowing for subsequent signature verification. This trusted, verifiability of the signature refers to the *message authentication* of the signed data. The digital certification of users is discussed in Section 5.2.1. The latter two points relate to the *temporal authentication* of the contract.

1.1.2 Message and Temporal Authentication

The undesirable properties of digital data (e.g., inability to detect tampering) lead us to the conclusion that the prevention of digital signature forgery requires that the signature have the following properties:

1. the signature must be dependent upon the entire data file to which it is applied, thus preventing the alteration of some or all of the file without rendering the signature invalid;

2. the signature must be tied to a particular individual so that no one other than this individual can produce a digital signature in the name of the individual.

We say that *data integrity* is provided if it is ensured that a message has not been altered in an unauthorized manner since its latest authorized alteration (or more correctly, that any unauthorized changes are detectable). This property satisfies the first point. *Message authentication (data-origin authentication)* is provided when a recipient is assured of the source (i.e., entity authorized to alter the data) of a given message. This property satisfies the second point. Notice that if data-origin authentication is provided, then so is data integrity, and vice-versa. For if data integrity is not provided then the data can be altered in an unauthorized manner without detection, hence the source (creator) of the message has changed. As well, if message authentication is not provided then the authorized (or unauthorized) alteration of a document is not well defined. As indicated in the previous subsection, the time at which the message authentication is provided is also important. This motivates the following definition for temporal authentication.

Definition 1.1 *Temporal authentication* combines message authentication with the notion of timeliness of messages.

The temporal authenticity of data produced by a time stamping protocol was first discussed by Just [Jus98]. This concept is discussed in greater detail in Chapter 4. In the following section, we further motivate the importance of time for the provision of temporal authentication.

1.2 Time and Cryptographic Authentication

Cryptographic authentication has inherent properties that necessitate a concept of time, related both to the participating entities as well as any operations that are performed. In this section we briefly highlight the requirement for time within current practice for the provision of cryptographic authentication.

At a very high level, a cryptographic infrastructure is composed of entities and protocols. The protocols provide for privacy and authenticity among the entities. The role of an entity changes over time. Employees are hired, fired and promoted each day. Privilege and responsibility are often associated with the role an entity plays, rather than with the (name of the) entity itself. Thus, it is important to know not only the identity of the entity that performed a certain action, but also when that action was performed.

Consider a private signature key held by some entity. Contracts and promisory notes may be signed by this entity each day. Later disputes over these signatures may require the determination of when the actions were performed, e.g., when the contract was signed. As well, for the provision of non-repudiation, it is important to determine when signatures were produced by a given entity in anticipation of the revocation of the entity's certificate. Determining the validity of a message signed with the private signature key requires detecting whether the message was signed before or after the corresponding certificate was revoked. Digital signatures alone do not allow for this determination. Incorporating an authentic notion of time does.

Protocols are similarly dynamic. Note that the security of many cryptographic protocols is based on computational complexity. As the amount of computing power available to adversaries increases, protocols may become vulnerable, e.g., if weak signature algorithms were used. Increases in computing power are a catalyst for larger key sizes and stronger algorithms. Renewing the privacy or authenticity associated with some information protected with "outdated" algorithms is therefore an important possibility.

Messages signed by a given entity with a particular version of a digital signature scheme (e.g., a given algorithm and key size) can only be deemed authentic for a finite amount of time. However, some signatures may require a period of authenticity that outlasts the lifetime of the digital signature scheme. Periodic increments in the security of time stamping protocols (either by advancing the size of the security parameter or using a new scheme all together) allows older, potentially vulnerable stamps or signatures to be renewed with a current timestamping scheme. A notion of time for these operations is important since the security will often depend on *when* something was done, and it is important that this temporal notion be maintained even when something is renewed.

1.3 Outline and Overview of Contributions

The objective of this thesis is

> to assimilate, conceptualize and analyze techniques for the provision, maintenance and verification of an authentic time, allowing one to determine the time of existence of digital data relative to the occurrence of other cryptographic events.

In the remainder of this section, we summarize the contents of each of the remaining chapters, emphasizing the novel contributions to the study of cryptographic authentication.

Chapter 2: A Taxonomy of Time Stamping Protocols

This chapter provides an assimilation and classification of full protocol descriptions of the previous literature on the time stamping of digital data. The specific contributions of this chapter are:

1. provision of the first comprehensive survey of known literature related to the time stamping of digital data;

2. an introduction of the distinction between absolute, relative and hybrid time stamps and a classification of the previous work under these terms.

Chapter 3: Critical Analysis of Previous Work

This chapter provides a critical and comparative analysis of previously published time stamping protocols. The specific contributions of this chapter are:

1. the first self-contained analysis of previous time stamping protocols;

2. the formalization and analysis of so-called group hashing techniques and the introduction of *incremental group hashing*.

Chapter 4: A Framework for Temporal Authentication

This chapter abstracts techniques from previous time stamping protocols and provides a general framework under which subsequent time stamping protocols, that provide temporal authentication, can be constructed. The specific contributions of this chapter are:

1. an introduction of the notion of temporal authentication and definition of the provision of absolute, relative and hybrid temporal authentication of digital data;

2. description of a first general framework (see Figure 4.2) allowing one to subsequently construct time stamping protocols that allow for the provision and verification of the temporal authentication of digital data. This construction

involves describing techniques for the provision of absolute, relative or hybrid time for the data as well as subsequent authentication and storage;

3. an identification of protocol failures for two previously proposed time stamping protocols (see Section 4.4 and Just [Jus98]). In particular, protocol failures are identified for

   (a) Protocol RL1 from Haber and Stornetta [HS91] in which we identify unreasonable requirements regarding the trust in the time stamp provider which allow the production of false time stamps (see Section 4.4.2), and

   (b) Protocol GH4 from Benaloh and de Mare [BdM93] in which the provision of an absolute time during time stamp production is shown to be unmeasurable during time stamp verification (see Section 4.4.1);

4. the presentation of Protocol HY2, a hybrid time stamping protocol that follows the new framework for the provision of temporal authentication (see Section 4.5).

Chapter 5: Time Stamping Digital Signatures

This chapter provides a comprehensive examination of the provision, verification, adjudication and renewal of time stamped or notarized digital signatures and relevant certificate information. The specific contributions of this chapter are:

1. a first identification of the requirements of temporally authenticated signatures and their storage, allowing for digital signature production and subsequent verification and adjudication of digital signatures even in the event of the expiry or revocation of a user's verification certificate or a change in the trust that signature recipients may have in a signature originator's certificate (see Section 5.3.1);

2. the first description of steps that must be taken and information or evidence required during the production, verification and adjudication of time stamped digital signatures (see Section 5.3.2);

3. defining the role of a notary and presenting Protocol NT2 which allows for the notarization of digital signatures (see Sections 5.4.1 and 5.4.2);

Chapter 6: Detecting Signature Key Compromise

This chapter provides an examination of techniques allowing one to address the problem of a signature key compromise so that one can detect a compromise and prevent the acceptance of signatures produced during a period of *undetected signature key compromise* (see Just and van Oorschot [JvO98]). The specific contributions of this chapter are:

1. an introduction of the problem of signatures being forged subsequent to a key compromise but prior to the detection of the compromise and identifying the limitations of current techniques for solving the problem (see Sections 6.1 and 6.2);

2. the introduction of a second level of authentication to solve the problem of undetected key compromise in which the signing user is required to obtain additional corroborative evidence from a trusted third party (through a secondary identification) for a signature to be accepted by a verifying party (see Section 6.3);

3. the presentation of solutions in which an independent, secret key is shared with the trusted third party to allow for secondary identification of the signing user (see Section 6.4);

4. the presentation of general techniques and specific solutions in which signing users are synchronized with the trusted authority so that a forged signature causes the synchronization property to be destroyed, and subsequently detected (see Section 6.5);

5. a first use of a cooling-off requirement for the acceptance of signatures combined with a check-in period for legitimate signers, allowing for the detection and prevention of acceptance of forged signatures (see Section 6.6).

Chapter 7: Concluding Remarks

In this chapter, we conclude with the positioning of the thesis as a novel contribution and aid to the study of cryptographic authentication.

Chapter 2

A Taxonomy of Time Stamping Protocols

This chapter surveys and classifies protocols in which time is authentically associated with digital data. This time may relate to the time of construction of the data, or more commonly, the time of submission of the data to some entity, and can be interpreted as evidence of the existence of the data at the given time.

The first discussions regarding the *time stamping* or *notarization*[1] of digital data can be traced to around the time of the origins of public-key cryptography. In 1979, Popek and Kline [PK79, page 353] state the following as the function of a so-called notary public machine upon receipt of a submitted message:[2]

> The notary public machine time-stamps the message, signs it itself (thereby encoding it a second time), and returns the result to the author.

Diffie [Dif82, page 67] makes reference to "a digital 'notary public' which dates [a] document and signs the date with its own private key." The earliest time stamping protocol was presented by Merkle [Mer80, Mer82] (see Protocol NT1 or the simplified

[1]References to notarization here refer only to the provision of a time stamp. More recent consensus regarding the interpretation of a notary differs from that of a time stamper (see Section 5.4 and Appendix A).

[2]Popek and Kline acknowledge David Redell for initially suggesting that the role of a notary public machine should be based on the notaries public in the paper world.

Protocol AB1). Adleman [Adl83] makes use of a tamper-proof device to limit the amount of trust required in a so-called notary public. Davies and Price [DP84, page 287] also enlist a "notary to sign and time-date stamp" input documents (in their particular case, digital signatures).

The growing need for some sort of digital *temporal authentication* (though this term was not used) providing for the authentic association of time with data was noted by Kanare [Kan86] though this was 7 years after the work of Popek and Kline. More recent investigations into digital time stamping were undertaken by Haber and Stornetta [HS91] in 1991.

Despite the Haber and Stornetta revival, the remaining history related to digital time stamping, from 1991 to the present, is fragmented and comprised of works which rarely reference each other (most likely because of their low profile, rather than for malicious reasons). This is best evidenced by the fact that there does not exist a paper that surveys the current state of the art of digital time stamping techniques (though see the partial work of Massias and Quisquater [MQ97]). This chapter provides such a survey.

Chapter Outline

In Section 2.1, the components of a time stamping protocol are identified and some basic primitives (e.g., hash and signature functions) are reviewed. In Section 2.2 a simple time stamping protocol is critiqued for motivational purposes. Section 2.3 illustrates many-to-one group hashing techniques in which a number of data submissions can be input to a function that produces only a single resultant value. These techniques allow for the possibility of more efficient time stamp production since the time provision techniques of Section 2.4 and Section 2.5 can be applied to a single representative group hash result rather than a number of data items. In Section 2.4, we begin our classification of time stamping techniques by reviewing protocols in which an *absolute time stamp* (see Definition 2.3) is provided. In Section 2.5, we complete our classification by reviewing methods for providing *relative time stamps* (see Definition 2.4).

2.1 Time Stamping Components

In this section we review and present several definitions and components related to the time stamping of digital data. The purpose of these intuitive definitions is to facilitate a classification and discussion of the previous work related to the time stamping of digital data. More formal definitions, generalizing on the previous work in this chapter and absorbing the critiques of Chapter 3, are presented in Chapter 4.

2.1.1 Stamping and Verification Protocols

In this subsection, we review and present several definitions and components used in the provision and recovery of a "time" for digital data. Several kinds of "time" are identified and used for later classifications of time stamping protocols reviewed in this chapter. The purpose of this subsection is only to provide an intuitive understanding of definitions and concepts related to time stamping. In Chapter 4, we provide more precision through the presentation of a framework that allows for the provision of time stamps.

Definition 2.1 A *non-cryptographic time stamp (NCTS)* is the output $w \in \mathcal{W}$ of the function $F : \mathcal{Y} \times \mathcal{T} \to \mathcal{W}$ where $y \in \mathcal{Y}$ is a binary message of arbitrary length and $t \in \mathcal{T}$ is a representation of a date or time.

The purpose of the NCTS w is to allow an indication of *when* the data y existed. The same data y may be time stamped on multiple occasions.

The cryptographic "authentication" of a NCTS is useful in anticipation of possible disputes, or more generally, in anticipation of low confidence in the "time" associated with the data by the NCTS. For example, if user u were to construct a time stamp w for a message y, u would compute $w = F(y,t)$. If u is honest, t would, for example, refer to the time of construction of w. However, there is no reason that one should or would necessarily trust u. "Authentication" serves to increase the confidence in the time provided during the construction of the NCTS, by establishing a verifiable, trustworthy process upon which the resultant, authenticated time stamp is constructed.

Definition 2.2 A *(cryptographic) time stamp* s (or *temporal stamp* or simply *stamp*) associated with a message y is the result of the "authentication" of a non-cryptographic time stamp $w = F(y, d)$ (see Definition 2.1) and for which the time $t \in \mathcal{T}$ has some "consistent meaning" for every time stamp. ■

The "consistent meaning" of a time stamp refers to the consistent notion of what the time t implies and how it is provided. (The discussed further in Section 4.3.1.) We briefly introduce different types of time below (and elaborate in Section 4.3).

Various Types of Time

A time stamp s can be one of three types, depending on the type of "time" provided. Specifically, s can be either

1. an *absolute time stamp*,
2. a *relative time stamp* or
3. a *hybrid time stamp*.

Definition 2.3 An *absolute time stamp* s is a (cryptographic) time stamp (see Definition 2.2) for which a universal, *absolute time* t (see Definition 4.7) is cryptographically bound to data y.

The provision of an absolute time stamp is *memoryless* with respect to any other time stamps that are produced. An absolute time stamp can be either *explicit* or *implicit*. An (explicit) absolute time stamp (the default) has the precise time directly recoverable or verifiable from the time stamp. For example,

Mon Oct 5 10:31:35 EDT 1998

is an example of an explicit, absolute time. An implicit absolute time stamp contains information from which the precise time can be uniquely determined. For example, an implicit absolute time stamp might include the values from several stocks at a particular time of day. The underlying assumption is that one can uniquely[3] determine

[3]Although the granularity of the time may vary (e.g., using the closing stock values may only provide a granularity of one day), a mapping to a single explicit time is required.

the time at which this information was generated, from this stock information. The use of an implicit time is discussed in Section 3.3.1.

Definition 2.4 A *relative time stamp* s is a (cryptographic) time stamp (see Definition 2.2) for which a *relative time* t (see Definition 4.12), ordering the data y after previously stamped data and before subsequently stamped data, is cryptographically bound to y.

Unlike an absolute time stamp, the time specified by a relative time stamp is meaningless on its own until compared to the times associated with other time stamps, i.e., the ordering of two or more stamps is determined.

Definition 2.5 A *hybrid time stamp* s is an (cryptographic) time stamp (see Definition 2.2) for which both absolute (see Definition 4.7) and relative times (see Definition 4.12) are cryptographically bound to y.

Stamping and Verification Protocols

Let $u \in U$ represent a user in a distributed network who would like to obtain a time stamp s for a message y that can thereafter be verified for its authenticity by a verifier (challenger) v. To accomplish this task, we make use of the following *Stamping* and *Verification* sub-protocols (similar to those presented by Benaloh and de Mare [BdM91] and mentioned briefly by Haber and Stornetta [HS91]). The production and subsequent authentication of a time stamp s is referred to as the *temporal authentication* of the message m, and is performed by a *time stamping protocol (TP)*. Henceforth we refer to the *time stamping* (or simply *stamping*) of a message (or document or data) y.

Definition 2.6 A *time stamping protocol (SP)*, when input a message y, outputs a time stamp s, as given by Definition 2.2.

Although the message y may possess special form (e.g., a document-signature pair), a SP treats y as a finite string of bits. The message y can be some data that u would like to stamp, or in the more likely case, y will be the output of a collision-resistant hash of the actual data x, i.e., $y = h(x)$ (see Definition 2.10 for definition and Section 2.2 for motivation of use). In Chapter 5, we discuss how a notary protocol takes advantage of the special form of the input, in particular, a digital signature. A more detailed analysis of a time stamping protocol is given in Chapter 4.

The "consistent meaning" of the time associated with digital data through the provision of a time stamp allows for subsequent comparisons of the times associated with several events. The verification of a time stamp s therefore involves a validation of the authenticity of the time stamp allowing a subsequent"temporal measurement" involving the associated time(s) t.

Definition 2.7 A *time stamp capsule* (or *timecapsule* or simply *capsule*) $cap_y = (y, t, s, \ldots)$ is a collection of data necessary to allow for the verification (see Definition 2.8) of the time t associated with the data y through the time stamp s.

Definition 2.8 A *time stamping verification protocol (VP)*, on input a time capsule cap_y performs a "validation" of the authenticity of the time stamp s, relative to (at least) the purported message y and time t.

Definition 2.9 A *time stamping (temporal) measurement* determines the ordering of two times t_1 and t_2, i.e., which of t_1 or t_2 is earlier or later, or whether they are equal.

When respectively associated with data y_1 and y_2 through time stamps s_1 and s_2, a temporal measurement determines which of the stamps, for example, may have been produced first. The verification and temporal measurement of a time stamp(s) is discussed briefly for schemes described in this chapter, and is expanded on in Chapter 4.

2.1.2 Hashing and Signing

In this subsection, we review the concepts of a hash function and signature algorithms. The use of each with regard to time stamping is motivated in Section 2.2. More detailed coverage regarding hashing and signing can be obtained from Menezes *et al.* [MvOV97] or Stinson [Sti95].

Definition 2.10 A *hash function* h takes as input an arbitrarily, finite sized bitstring x and produces an l-bit output, e.g., $l = 160$ for SHA-1 [FIP95].

When several data are input to h, the data are concatentated and input as a single data item. Concatenation will be denoted using commas as in $h(x, y)$ which represents the hash of the concatenation of data x and y. For use in this thesis, h will have the following properties

1. *Ease of computation.* Given x, $y = h(x)$ is easy to compute.
2. *Collision resistance.* It is difficult to find inputs $x \neq x'$ such that $h(x) = h(x')$.
3. *One-wayness.* It is difficult to find x given $y = h(x)$.

Under suitable conditions (and for most hash functions used in practice) a collision-resistant hash function is also one-way.

A *digital signature* provides for the following properties for signed data.

1. *Data-origin authentication.* Authentically binds the identity of the signing user to a message.
2. *Non-repudiation.* Does not allow the legitimate signer to repudiate the legitimate production of a valid signature, i.e., in an attempt to deny having produced the signature.

Definition 2.11 A *signature scheme* SS is a five-tuple $(\mathcal{M}, \mathcal{Q}, \mathcal{K}, \mathcal{SA}, \mathcal{VA})$, where the following conditions are satisfied ([Sti95, Def. 6.1]):

1. $\mathcal{M}$ is a finite set of possible messages
2. $\mathcal{Q}$ is a finite set of possible signatures
3. $\mathcal{K}$, the keyspace, is a finite set of possible keys
4. For each user $u \in U$, possessing a key $K \in \mathcal{K}$, there is a *signing algorithm* $sig_u \in \mathcal{SA}$ and a corresponding *verification algorithm* $ver_u \in \mathcal{VA}$. Each sig_u :

$\mathcal{M} \rightarrow \mathcal{Q}$ and $ver_u : \mathcal{M} \times \mathcal{Q} \rightarrow \{\text{true, false}\}$ are functions such that the following equation is satisfied for every message $m \in \mathcal{M}$ and for every signature $c \in \mathcal{Q}$:

$$ver_u(m, c) = \begin{cases} \text{true} & \text{if } c = sig_u(m) \\ \text{false} & \text{if } c \neq sig_u(m) \end{cases}$$

A particular instatiation of a signature scheme is given as Protocol SG1.

Protocol SG1 The Digital Signature Algorithm (DSA) [FIP94].

Initialization

Note: This initialization is performed by each user $u \in U$.

1: u selects a 512-bit prime p with the property that computing discrete logarithms in $\mathbb{Z}_p$ is computationally infeasible.
2: u selects a 160-bit prime q such that q divides $p - 1$.
3: u repeatedly selects an element $g \in \mathbb{Z}_p{}^*$ and computes $\alpha = g^{(p-1)/q} \bmod p$ until $\alpha \neq 1$.
4: u selects a random a where $1 \leq a < q$ and computes $w = \alpha^a \bmod p$.
5: u's *public key* is the tuple (p, q, α, w), while the private key is a.

Signature Production

Input: message m
Output: signature $c = sig_u(m) = (r_1, r_2)$ for message m

1: For a message m, u computes $y = h(m)$ using SHA-1 [FIP95] and selects a random k where $1 \leq k < q$. The signature (r_1, r_2) for the message m is defined as

$$\begin{aligned} r_1 &= (\alpha^k \bmod p) \bmod q \\ r_2 &= (y + ar_1)k^{-1} \bmod q \end{aligned}$$

Signature Verification

Input: message m' and signature (r_1', r_2') purportedly corresponding to message m'
Output: indication of whether (r_1', r_2') is a mathematically correct signature for m'

1: Verifier v computes $y' = h(m')$, as well as

$$\begin{aligned} z_1 &= w(r_2')^{-1} \bmod q \\ z_2 &= r_1'(r_2')^{-1} \bmod q. \end{aligned}$$

2: $ver_u(m', c') = \text{true iff } (\alpha^{z_1} y^{z_2} \bmod p) \bmod q = r_1'$.

The signature function sig_u is used to digitally sign a message m and is denoted $c = sig_u(m)$. In most cases, m will be hashed first giving $c = sig_u(h(m))$ and unless otherwise indicated, we assume that messages are hashed before they are signed, but do not explicitly show this in the notation. ver_u can be used by all other users to verify the authenticity of a signature (i.e., that a given message was indeed signed by u). If the message has been hashed first, this requires delivery of m along with s and computation of $h(m)$ for signature verification. The verification key of u is likewise typically, or for our discussion assumed to be signed by a trusted certification authority (CA), resulting in a public key certificate $cert_u$. Certificates allow one to trust the binding, i.e., that the key does indeed belong/is associated with the named entity. (We briefly discuss certificates here. A more detailed examination is given in Section 5.2.) This certificate typically contains at least, a unique identification number (certificate serial number), the distinguished name of u, u's verification key, and a validity period. In this way, u is bound to signatures that are successfully verified using u's public verification key, which has been bound to u's name through the CA-signed certificate $cert_u$. The strength of this binding is determined by, among other things, the thoroughness of the identity check performed by the CA before issuing a certificate to u.

2.2 Critique of a Simple Time Stamping Protocol

Consider the time stamping protocol given by Haber and Stornetta [HS91] in Protocol SM1 (a similar scheme was described by Kanare [Kan86]). The scheme is useful for motivating the use of the hashing and signing primitives described in Section 2.1.2 and illuminating some important requirements concerning the authentic provision of a time for data.

Some concerns with such a scheme (expanding on points made by Haber and Stornetta [HS91]) are:

1. *Privacy of the document.* This is not necessarily a concern for all submitted documents. However, where there is concern, it results from the fact that T as well as any eavesdropper, is able to read the contents of the submission in transit to T.

Protocol SM1 Document storage by a trusted authority [HS91, Kan86]

```
Stamping
Require: The time stamping service $T$ is required to maintain the authenticity of
    the database in which the time stamps are stored for each user.
Input: document $x$
Output: central storage of the time stamp $s = (x, t)$, i.e., the absolute time $t$ ap-
    pended to $x$
1: user $u$ submits the original document $x$ to a time stamping service T
2: T appends the time of submission to $x$ producing $s = (x, t)$ and stores the result
Verification
Input: document $x$, query of the time(s) associated with $x$
Output: absolute time(s) associated with document $x$
1: user $v$ submits document $x$ to the time stamping service T, with a request for the
    time(s) associated with $x$
2: T searches through its entire database outputting the time(s) associated with $x$
```

2. *Size (storage) of the document.* This is especially relevant considering that T is accepting submissions from a potentially large number of users, that may require long (e.g., 10 years) storage periods. Multiple, variable-sized submissions from a number of users can soon make storage prohibitive for T.

3. *Bandwidth required to transmit the document.* Similar to the storage required for T, the task of transmitting the document to T may impose a burden on the distributed system. (Although the transmission of the entire data appears to be necessary, we see below how the submission size can be significantly decreased.)

4. *Authenticity of the document.* The document may be altered during transit by an attacker or erroneously recorded (i.e., the data or the time) by T. Any such errors (malicious or otherwise) are not necessarily detectable to T.

5. *Trust.* For example, a malicious T might create false stamps. More specifically, T can append a 'time' of its choosing to any document of its choosing. For example, T has the ability to time stamp a document with a time that is one year earlier than the current time. Also, during verification, a user does not have a method for determining the trustworthiness of the response from the

alleged T.

Although the privacy of the submission may not be necessary for the submitter of the document, it is certainly not necessary for T to have knowledge of the contents. (Recall that a time stamping protocol is not concerned with the form of a message, but rather, treats the input document only as a finite string of bits.) Encrypting the document is one option. Indeed, since u may encrypt the document for himself on local storage, it may make sense to also send an encrypted copy to T. (Notice that this complicates the verification procedure since a copy of the decryption key and algorithm would be required by the verifier.)

The size of the transmitted document and storage required by T are not aided by the encryption of the document. However, compression of the document will, in many cases, reduce the size of the submitted document. Indeed, the document can be compressed, followed by an encryption.

However, a better solution exists. As suggested by Haber and Stornetta [HS91], u can deal with the privacy, storage and transmission concerns by using a collision-resistant hash function (see Section 2.1.2), i.e., by submitting $y = h(x)$ as opposed to simply x. u still maintains a copy of the original document x. A user $v \neq u$, verifying the time associated with y would obtain x from u and submit $y = h(x)$ to T for a verification of the time. The one-wayness property precludes u from claiming that an alternative document is the actual input to the hash. Privacy concerns on u's machine can still be solved by encrypting the document.

Transmission or storage errors can be dealt with by having T return an acknowledgement for each submission. u can verify that the correct hash has been recorded with the correct corresponding submission time. However, this does not preclude the possibility of the 'loss' of the information at T's end or an active attack or impersonation of T to the submitting or verifying user. An alternative to this problem, as well as some limiting of malicious incompetence can be achieved by having the acknowledgement for a given submission authenticated by T, along with the time of submission, and returned to u and verified for its authenticity. For example, this may involve the returning of a T-signed time stamp to u. Altering Protocol SM1 to take into account the concerns above, one obtains Protocol AB1 as given in Section 2.4.

The question of trust in a central time stamping authority may be handled in many ways, including the following. One is to decentralize the stamp computation. The verification of valid stamps then becomes one of designing a secure multi-party stamping protocol. A second solution, linking, may be used to restrict the stamps that T is allowed to produce, and more importantly, the times at which he is able to produce them (see Section 2.5). Restricting T's ability to alter the temporal ordering serves to reduce the trust required in him.

2.3 Group Hashing

In this section we review techniques for allowing a coarser granularity for the time provided by the time stamping of data. This granularity is provided by the time stamping of a *round's* worth of documents, rather than of individual data. More than one document is input per round, producing a single representative group hash value; one value will be produced for a given round. The duration of the round can be parameterized by a fixed length of time or a maximum number of documents that might be group hashed during a particular round.

The temporal ordering of each document is performed by having the submitting user

1. demonstrate that a document was submitted to a given round; and
2. recover the time associated with the resultant round value in which the document was asserted as being submitted.

In this section, Item 1 is dealt with. In Section 3.2, we analyze these techniques. Techniques for providing a time for the group hash result (i.e., Item 2) are reviewed in Sections 2.4 and 2.5.

Two motivations for using a group hash over a round of submitted documents are:

1. *Storage efficiency [HS91, Mer80].* For time stamping protocols in which multiple copies of a time stamp are required or a centralized storage facility is used, some group hash techniques allow for more efficient storage to be realized (see

Section 3.2.2 for further analysis). Rather than one stamp for each document, there is one stamp for every m documents. The size of the resultant group hash affects the success of this technique at decreasing the storage.

2. *Decentralized computation [BdM93, Nyb96].* The resultant group hash value a_r for round r can (in most cases) be computed individually by each user; it may not be necessary to have a centralized T compute the resultant value.

An important practical concern (especially with a decentralized computation) involves the ordering of the documents input to a particular round, to allow each participating user to recover the same round value (in the case that the group hash computation is non-commutative). Benaloh and de Mare [BdM91] suggest that each user in the entire system have a regular position for each round (their suggestion applied particularly to Protocol GH3). In rounds in which users do not participate, a default value is used in place of a real document. Thus, if there were m documents submitted per round and n total users in a particular distributed system, then n documents would be submitted to the group hash function. This is clearly impractical since it may often be the case that $n >> m$. Using a centralized T for stamp computation is useful for cases when an ordering of the participants' input to the group hash function is required and as well, avoids the need for expensive broadcasting and impractical user interactions. The beginning and end of a round can be delimited by publically known times (e.g., every 10 minutes starting on the hour) or may be more dynamic if orchestrated by a central authority.

Let $y_1, y_2, \ldots, y_m$ be the documents submitted during round r; submitted here means either broadcast by each user to all others in the case where each user must be able to compute a_r or simply transmitted to a centralized time stamp authority (T). For simplicity, let u_i be the submitter of y_i, though in reality, one user can submit more than one document. Let $|y_i| = \log_2 y_i = \lg y_i = n$ denote the number of bits contained in y_i. In most cases, beyond the production of a_r for a given round, user u_i will also have some additional information, necessary (if it exists) for demonstrating that the data y_i was indeed used in the computation of a_r. We refer to this information as $member_{y_i}$.

The potential decrease in the amount of storage required for the resultant group hash value has the effect of potentially increasing the amount of storage required for $member_{y_i}$ for each participant in a round. The next three subsections respectively review techniques in which this amount is decreased.

2.3.1 Linear Storage

In this subsection we discuss schemes for which the amount of storage for each user participating in the round is linearly proportional to the number of participants (= number of documents) submitted in the round. In other words, if u_i submits y_i, then $member_{y_i}$ contains a number of components (each the same size as y_i) that is a linear function of the number of users participating in the round.

Protocol GH1 Cumulative Group Hash [BdM91].

```
Group Hash
Require: An ordering of the users participating in each round is required to allow
    consistent computations of $a_r$ by each participating user.
Input: $\{y_1, \ldots, y_m\}$
Output: $a_r$, $member_{y_i}$
1: $y_i = h(x_i)$ for document $x_i$ is broadcast by each $u_i$
2: each user computes and stores $a_r = h(y_1, \ldots, y_m)$ and $member_{y_i} =$
    $(y_1, \ldots, y_{i-1}, y_{i+1}, \ldots, y_m)$ for their own $y_i$.
Verification
Input: $y_i$, $member_{y_i}$, $a_r$
Output: indication of whether $y_i$ contributed to the construction of $a_r$
1: user $v$ obtains $y_i$ and $member_{y_i}$ from $u_i$, computes $a =$
    $h(y_1, \ldots, y_{i-1}, y_i, y_{i+1}, \ldots, y_m)$ and accepts that $y_i$ contributed to the pro-
    duction of $a_r$ only if $a = a_r$.
```

Protocol GH1 was given by Benaloh and de Mare [BdM91]. Assuming that h is collision-resistant (see Section 2.1.2), it should be computationally infeasible for one to find inputs $y'_1, y'_2, \ldots, y'_{m'}$ such that

$$h(y_1, y_2, \ldots, y_m) = h(y'_1, y'_2, \ldots, y'_{m'}).$$

Notice that it may be that $m' \neq m$, as the verifier may not be aware of the number of documents submitted for the computation of a_r. If each user stores a_r, it is only necessary that the submitter has the inputs necessary to reproduce a_r, and hence demonstrate that his document belongs to the round in which a_r was computed.

Protocol GH1 can be altered slightly to allow for a recursive construction of a_r [BdM91]. Let

$$\begin{aligned} z_1 &= y_1, z_2 = h(z_1, y_2), \ldots, z_m = h(z_{m-1}, y_m) \\ a_r &= z_m \end{aligned} \tag{2.1}$$

On average, each user stores $\frac{m}{2}$ documents and asymptotically, the storage over the system is the same as the previous scheme, i.e., approximately m^2 documents are stored in total by the m users participating in a particular round. Protocol GH2 describes the scheme.

Protocol GH2 Recursive Group Hash [BdM91].

```
Group Hash
Require: An ordering of the users participating in each round is required to allow
    consistent computations of $a_r$ by each participating user.
Input: $\{y_1, \ldots, y_m\}$
Output: $a_r$, $member_{y_i}$
 1: $y_i = h(x_i)$ for document $x_i$ is broadcast by each $u_i$
 2: each user computes and stores $a_r$ (as computed in (2.1)) and $member_{y_i} =$
    $(z_{i-1}, y_{i+1}, y_{i+2}, \ldots, y_m)$ for their own $y_i$
Verification
Input: $y_i$, $member_{y_i}$, $a_r$
Output: indication of whether $y_i$ contributed to the construction of $a_r$
 1: user $v$ obtains $y_i$ and $member_{y_i}$ from $u_i$, computes $a = h(z_{i-1}, y_i, y_{i+1}, \ldots, y_m)$
    and accepts $y_i$ if $a = a_r$.
```

2.3.2 Logarithmic Storage

The schemes of Section 2.3.1 required that each user that submitted a document for group hashing during a particular round, stored an amount of information that was always linearly proportional to the total number of documents submitted to a given round. In what follows, we review a technique for reducing this storage, allowing one to achieve a logarithmic storage requirement.

We can think of the production of a group hash as a technique to authenticate a number of pieces $y_1, y_2, \ldots, y_m$ to produce a single authentic value a_r. However, in our case, we only require a resultant value whose authenticity is maintained by other means, i.e., the timing information is provided at a later time. A similar problem was tackled by Merkle [Mer80, Mer82] for the authentication of a file of public keys. Rather than having each user store all y_i (e.g., the public keys of all other users), compute a_r as

$$a_r = h(h(y_1, \ldots, y_{\lfloor \frac{m}{2} \rfloor}), h(y_{\lfloor \frac{m}{2} \rfloor + 1}, \ldots, y_m))$$

In this way, u_i need only store the *single* hash of the $\left\lfloor \frac{m}{2} \right\rfloor$ documents that doesn't contain his own y_i as well as the remaining $\left\lfloor \frac{m}{2} \right\rfloor$ documents, i.e. $\left\lfloor \frac{m}{2} \right\rfloor + 1$ total pieces as opposed to $m - 1$ for Protocol GH1. Notice as well the alternative computation for $h(y_1, \ldots, y_{\lfloor \frac{m}{2} \rfloor})$ (and similarly for $h(y_{\lfloor \frac{m}{2} \rfloor + 1}, \ldots, y_m)$) as

$$h(h(y_1, \ldots, y_{\lfloor \frac{m}{4} \rfloor}), h(y_{\lfloor \frac{m}{4} \rfloor + 1}, \ldots, y_{\lfloor \frac{m}{2} \rfloor})).$$

This additional measure reduces the storage to $\left\lfloor \frac{m}{4} \right\rfloor + 3$ for each user.

One can continue by recursively dividing until each is the hash of only a single y_i, in other words until $\frac{m}{2^d} = 1$, which occurs when $d = \lg m$. Generalizing from above, the amount of storage for each user is the number of intermediate hashes plus the number of documents required to compute the hash to which you belong, or in other words,

$$\frac{m}{2^d} + (d - 1)$$

which is $\lg m$ when $d = \lg m$. Therefore, the storage for each user, is logarithmic in the number of round participants. This computation is shown in Figure 2.1 when $m = 8$.

Specific implementations of this idea (as applied directly to digital time stamping) were given independently by Benaloh and de Mare [BdM91] and Bayer, Haber and Stornetta [BHS93]. They are essentially straightforward implementations of Merkle's tree authentication [Mer80, Mer82]. A variation of this technique is used in a commercial time stamping implementation given as Protocol HY1 [Tro95], where a central

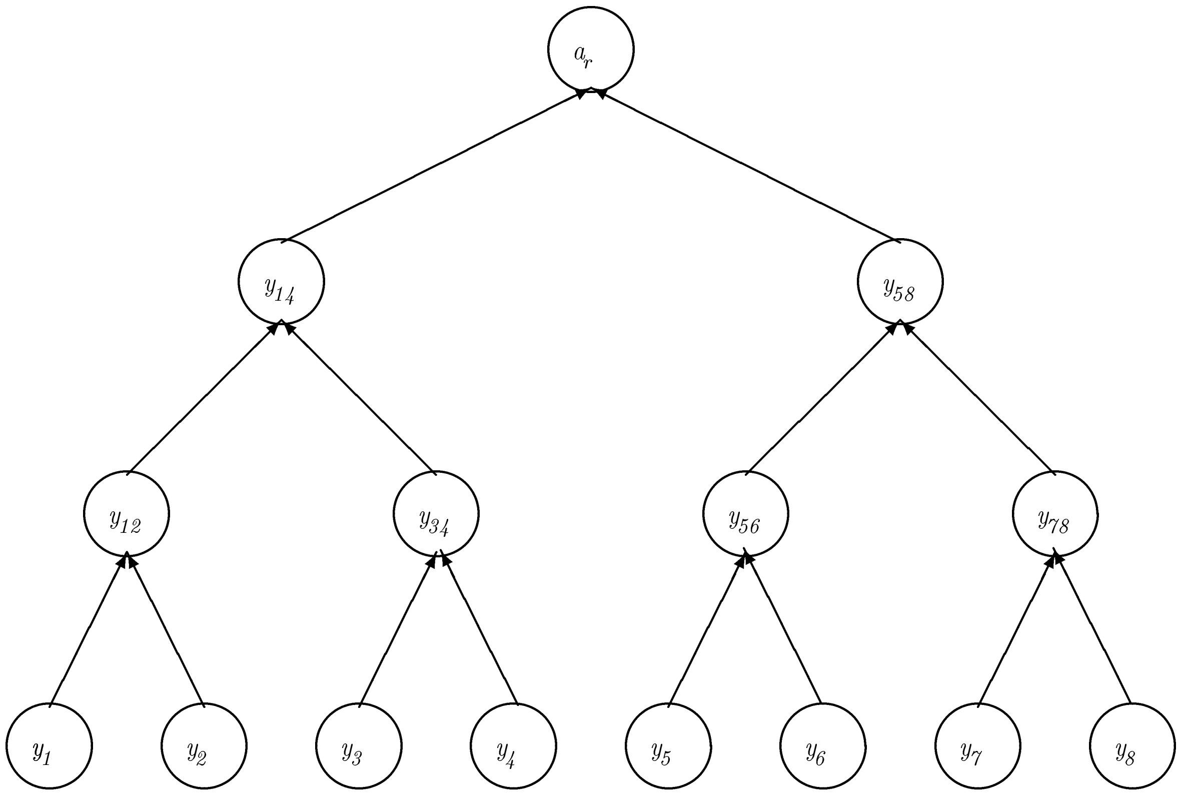

Figure 2.1: Logarithmic user storage group hash technique. A specific example is illustrated in which 8 data are group hashed to produce a_r. Each y_i is computed as the hash of the concatentation of the two children. For example, $y_{58} = h(y_{56}, y_{78})$ where ‘,’ denotes the concatentation of the bitstrings y_{56} and y_{78}.

entity performs the computation of the hash. The time stamp is returned to the user (including an indication of which leaf position the user has been given) who can then verify his inclusion in the round by comparing his compuation of the time stamp with the one that is published say weekly, for example, in the NY Times. A generic (decentralized) version of the tree-based hashing technique is given in Protocol GH3.

Protocol GH3 Tree Group Hash [Mer80, BdM91, BHS93]

Group Hash

Require: An ordering of the users participating in each round is required to allow consistent computations of a_r by each participating user.
Input: $\{y_1, \ldots, y_m\}$
Output: a_r, $member_{y_i}$

1: $y_i = h(x_i)$ for document x_i is broadcast by each u_i
2: each user participating in the current round computes a_r as shown specifically for $m = 8$ in Figure 2.1 where the parent of nodes containing bit strings y and y' is computed as $h(y, y')$ and ',' denotes the concatenation of bit strings. $member_{y_i} = (z_1, \ldots, z_{\lg m})$ is a list containing the $\lg m$ values in the tree necessary to recompute a_r. From Figure 2.1 for example, we have $member_{y_3} = (y_4, y_{12}, y_{58})$

Verification

Input: y_i, $member_{y_i}$, a_r
Output: indication of whether y_i contributed to the construction of a_r

1: user v obtains y_i and $member_{y_i}$ from u_i, computes $a = h(\ldots(h(y_i; z_1); \ldots); z_{\lg m})$ and accepts y_i if $a = a_r$. We use a ';' here to indicate concatenation as before except that the order of these inputs varies for each user. For example, from Figure 2.1, for y_3 and $member_{y_3}$, v computes

$$a = h(\underbrace{h(y_{12}, \underbrace{h(y_3, y_4)}_{y_{34}})}_{y_{14}}, y_{58})$$

where if the correct y_3 and $member_{y_3}$ were maintained by u_3, $a = a_r$ is true. Therefore for u_3, $y_{34} = h(y_3, z_1)$ where z_1 is the first element of $member_{y_3}$ while for u_4, $y_{34} = h(z_1, y_4)$ using $member_{y_4}$.

As demonstrated by Benaloh and de Mare [BdM91] (and similar to the security provided for the schemes of Section 2.3.1), finding a $y' \notin \{y_1, \ldots, y_m\}$ such that a challenger could be fooled into believing that y' was indeed part of the hash to produce a_r would imply that a collision for the hash function h has been found. This, however, would contradict the assumption of collision resistance for the hash.

2.3.3 Constant Storage

In the previous two subsections, linear and logarithmic storage factors were respectively achieved for each user participating in a round. These solutions used a (non-specific) hash function for which the only assumption made was it be collision-resistant. In this subsection, additional assumptions on the hash function are used to allow one to achieve a constant amount of storage for each user.

Schemes Based on Associative Hash Functions

Continuing along the same lines as in Section 2.3.1, the resultant group hash a_r over the documents $y_1, y_2, \ldots, y_m$ is computed recursively as in (2.1).

Definition 2.12 A function $h : X \to Y$ is associative if $\forall x, y, z \in X$, $h(x, h(y, z)) = h(h(x, y), z)$.

The following ideas were presented by Benaloh and de Mare [BdM91]. Suppose that the hash function h is associative. Since the order of application of an associative function h is irrelevant, u_i need only store z_{i-1}, y_i and an accumulated hash of all y_j where $j = i + 1, \ldots, m$, namely w_{i+1}. The computation of the hash for the round by this user consists in computing $h(h(z_{i-1}, y_i), w_{i+1})$. If h is also commutative, then user u_i need only store y_i along with a single accumulated hash for all y_j, $j \neq i$.

Is the construction of an associative one-way hash an achievable goal? As of yet, the answer is no, and they do have other applications as well (which are beyond the scope of this thesis). An overview of this topic is given by Rabi and Sherman [RS97].

One-Way Accumulators

Benaloh and de Mare [BdM93] use the properties of quasi-commutativity (defined below) and one-wayness to develop a one-way accumulator which allows the resultant hash value for a round to be computed with only a constant amount of storage for each user. The scheme is described as a decentralized computation.

To achieve their goal, the definition of a hash function is slightly altered. A family of *one-way hash functions* is an infinite set of functions $h_i : X_i \times Y_i \to Z_i$ such that (i is subsequently omitted for simplicity):

1. *Ease of computation.* Given x and y, $z = h(x, y)$ is computationally feasible to compute.

2. *Collisions.* Given a pair (x, y) and given a y', it should be computationally infeasible to find an x' such that $h(x, y) = h(x', y')$. Note that it may be possible that given (x, y), one can easily find a pair (x', y').

A function $f : X \times Y \to X$ is *quasi-commutative* if

$$f(f(x, y_1), y_2) = f(f(x, y_2), y_1)$$

A family of *one-way accumulators* is a family of one-way hash functions that are each quasi-commutative. The one-way accumulator is useful in that computations such as

$$z = h(h(\ldots h(\ldots (h(h(x, y_1), y_2), \ldots, y_i), \ldots, y_{m-1}), y_m)$$

for initial value x, do not depend on the order of the y_i. Thus, given only a single intermediate hash value w_i where

$$w_i = h(h(\ldots h(h(\ldots (h(h(x, y_1), y_2) \ldots, y_{i-1}), y_{i+1}), \ldots, y_{m-1}), y_m),$$

$z = h(w_i, y_i)$ is computable.

Notice that addition, multiplication and exponentiation are all quasi-commutative. However, only exponentiation has the potential of being one-way. Therefore, let

$$e_n(x, y) = x^y \bmod n$$

be the accumulator, where $n = pq$ is the product of two primes, the form of which is discussed below. Group hashing using this accumulator is described in Protocol GH4.

Protocol GH4 Exponentiation Group Hash [BdM93].

```
Group Hash
Require: A rigid composite integer $n = pq$, initially constructed by a trusted authority. The primes $p$ and $q$ are destroyed subsequent to the computation of $n$
Input: $\{y_1, \ldots, y_m\}$
Output: $a_r$, $member_{y_i}$
1: $y_i = h(x_i)$ for document $x_i$ is broadcast by each $u_i$
2: users agree upon a value $x$ (though likely there is a common public value used by all users for each round, e.g., as suggested by Benaloh and de Mare, a representation of the current time), from which the starting seed $x_0 = x^2 \bmod n$ is obtained. Each user $u_i$ computes and stores $a_r = x_0^{y_1 \cdots y_m} \bmod n$ and $member_{y_i} = x_0^{y_1 \cdots y_{i-1} y_{i+1} \cdots y_m} \bmod n$ for their own $y_i$.
Verification
Input: $y_i$, $member_{y_i}$, $a_r$, $n$
Output: indication of whether $y_i$ contributed to the construction of $a_r$
1: user $v$ obtains $y_i$ and $member_{y_i}$ from $u_i$, computes $a = (member_{y_i})^{y_i} \bmod n$ and accepts $y_i$ if $a = a_r$.
```

To achieve a temporal authentication of this group hash, the authors suggest that the value x might represent the current date or time. The fault with this suggestion is discussed in Section 4.4.1. With regards to one-wayness, for a suitably chosen composite n, the product of two primes, Shamir [Sha81] has shown that if root finding is difficult (i.e., for a given y' and a_r, finding $member_{y'}$ such that $a_r \equiv (member_{y'})^{y'} \pmod{n}$), then $e_n(x, y)$ is one-way for suitably chosen n. However, because of the repeated exponentiations, the worry that small subgroups may be reached necessitates a stricter construction for n, as now discussed.

A prime p is *safe* if $p = 2p' + 1$ where p' is an odd prime. A *rigid integer* $n = pq$ is composed of distinct safe primes p and q such that $|p| = |q|$, denoting that the bitlengths of the primes are equal [BdM93]. Given that $\gcd(y, n') = 1$ where $n' = \frac{p-1}{2}\frac{q-1}{2}$, computations of e_n will stay in the large subgroup of squares modulo n provided $x \neq \pm 1$ and $y \neq 0$. The construction of n may be undertaken via a trusted outside source, a special purpose physical device, or a secure multiparty computation.

One thing that might help an attacker is the access to a number of other roots modulo n, i.e., the stamps of each of the other users for a particular round. Benaloh and de Mare [BdM93] go on to show that such an attack is computationally infeasible.

Fast Accumulated Hashing

Nyberg [Nyb96] describes a scheme for accumulated hashing. It is an improvement over Protocol GH4 in that it does not have a trapdoor. In Protocol GH4, the parameters were chosen such that there exists a trapdoor for easily cheating the scheme (i.e., the factorization of n). Despite the existence of this information, the trapdoor is not used in the scheme nor is it even supposed to remain known as it is ideally destroyed during the public parameter creation period. An equally attractive advantage for Nyberg's scheme is that besides the submitted data, the user need not store any additional information, i.e., $|member_{y_i}| = 0$.

Assume that y_i, $i = 1, \ldots, m$ are the l-bit one-way transformations of documents submitted for accumulation (the process of obtaining the l-bit transformations is described later). Further assume that the y_i are randomly chosen elements from a set Y with uniform distribution. Each y_i can be decomposed as

$$y_i = \{y_{i1}, y_{i2}, \ldots, y_{ir}\} \tag{2.2}$$

where $|y_{ij}| = d$ for $i = 1, \ldots, m$, $j = 1, \ldots, r$.

Using this partitioning, for each y_i, an r-bit $b_i = \{b_{i1}, \ldots, b_{ir}\}$ is obtained using the following rule:

$$\text{for } i = 1, \ldots, m, j = 1, \ldots, r, \; b_{ij} = \begin{cases} 0 & \text{if } y_{ij} = \{0\}^d, \\ 1 & \text{otherwise} \end{cases} \tag{2.3}$$

This first compression by a factor of d is a function $f : \{0,1\}^{rd} \rightarrow \{0,1\}^r$ where each d-bit y_{ij} is mapped to a single bit b_{ij}.

The r-bit accumulation value (time stamp) is obtained by performing bitwise multiplication modulo 2 of the components of the b_i. In other words, the accumulation value $a = \{a_1, a_2, \ldots, a_r\}$ is obtained as

$$a_j = \prod_{i=1}^{m} b_{ij} \bmod 2, j = 1, \ldots, r \tag{2.4}$$

This second compression by a factor of m can be described by the function $g : \{0,1\}^{rm} \rightarrow \{0,1\}^r$. Notice also that we have $member_{y_i} = \emptyset$. The protocol steps are described in Protocol GH5.

Protocol GH5 Bit Group Hash [Nyb96].

```
Group Hash
Input: ${y_1, ..., y_m}$
Output: $a_r$
1: For their own document $x_i$, each user $u_i$ computes the $l = rd$ bit $y_i = gen(h(x_i))$
   as described below for (2.5).
2: $y_i$ is decomposed as in (2.2).
3: $y_i$ is compressed to obtain the $r$ bit $b_i$ as in (2.3).
4: $b_i$ is broadcast to all users.
5: Each user computes and stores $a$ as computed in (2.4).
Verification
Input: $x_i$, $a_r$
Output: indication of whether $y_i$ contributed to the construction of $a_r$
1: user $v$ obtains $x_i$ from $u_i$, computes $b_i$ (as described for computation of the Group
   Hash shown above in the first part of the protocol) and accepts $x_i$ if the $j$th bit
   in $a$ is 0 when the $j$ bit in $b_i$ is also 0.
```

Analysis of Protocol GH5. Suppose that a user claims that a value x' contributed to the production of a. Firstly, x' is submitted to obtain the l-bit y'. y' is split into r d-bit pieces and the first part of the compression is performed to obtain the r-bit value $b' = \{b'_1, b'_2, \ldots, b'_r\}$.

Comparing the submitted b' with the authenticated accumulation value a, a sufficient condition to have $a_j = 0$ is that $b'_j = 0$ (it is only sufficient since any one of m b'_i's can contribute to having $a_j = 0$). Now if $b'_j = 0$ in the test string, the claim is that b' belongs to the round in which a was produced. Therefore, for b' to "belong to a" it must be true that

$$\text{if } b'_j = 0 \text{ then } a_j = 0, \text{ where } j = 1, \ldots, r.$$

For the security of Protocol GH5, it is important to know if such a $y' \notin \{y_1, \ldots, y_m\}$ can easily be found that passes the verification step. To simplify the analysis, assume a given b', produced subject to the restraints of Protocol GH5. For this b', we can state the following probabilities:

$$\begin{aligned} P(b'_j = 0) &= 2^{-d}, \\ P(b'_j = 1) &= 1 - P(b'_j = 0) = 1 - 2^{-d}. \end{aligned}$$

For the group hash value a,

$$\begin{aligned} P(a_j = 1) &= \prod_{i=1}^{m} P(b_{ij} = 1) \\ &= \prod_{i=1}^{m} (1 - 2^{-d}) \\ &= (1 - 2^{-d})^m \\ P(a_j = 0) &= 1 - P(a_j = 1) \\ &= 1 - (1 - 2^{-d})^m. \end{aligned}$$

Returning to the candidate b', it is important to determine its probability of success for being deemed to "belong to a". Thereafter, one can choose parameters to minimize this probability. In the theorem below, let $N = 2^d$ be the upper bound on the number of messages that can be hashed, i.e., $m \leq N$.

Theorem 2.1 *[Nyb96] Given a candidate b' and an authenticated, accumulated hash, denoted a, created subject to the restraints of Protocol GH5, then*

$$P(b' \text{ “belongs to” } a) \approx e^{\frac{-r}{Ne}}$$

Proof The following steps will determine the probability that b' will pass the verification step. Let q_j be the proposition "if $b'_j = 0$ then $a_j = 0$". The negation $\neg q_j$ of q_j is the statement "$b'_j = 0 \wedge a_j = 1$".

$$\begin{aligned} P(\neg q_j) &= P(b'_j = 0) P(a_j = 1) \\ &= 2^{-d}(1 - 2^{-d})^m \\ P(q_j) &= 1 - P(\neg q_j) \\ &= 1 - 2^{-d}(1 - 2^{-d})^m \end{aligned}$$

t	l
30	835K
50	1.4M
75	2.1M
100	2.8M

Table 2.1: BitLength l required for the output of the one-way transformation to allow a random value b' to have e^{-t} probability of "belonging to" accumulator a. $l = rd$ where $r = Net$ and $d = \lg N$. $e = 2.71828$ is the natural logarithm. For this particular example, $d = 10$ (so that $N = 1024$).

$$\begin{aligned} P(b' \text{ ``belongs to'' } a) &= P(q_1 \wedge \cdots \wedge q_r) \\ &= P(q_1) \cdots P(q_r) \\ &= (1 - 2^{-d}(1 - 2^{-d})^m)^r \end{aligned}$$

From here, we have

$$(1 - \frac{1}{2^d}(1 - \frac{1}{2^d})^m)^r \leq (1 - \frac{1}{N}(1 - \frac{1}{N})^N)^r \approx (1 - \frac{1}{Ne})^r \approx e^{\frac{-r}{Ne}}$$

■

Let e^{-t} be the probability of success for a candidate b' giving

$$l = \overbrace{Net}^{r} \underbrace{\lg N}_{d}.$$

Table 2.1 displays some possible choices where $N = 1024$ (hence $d = 10$) for a variable t. It demonstrates that for even the most minimal security – e^{-30} – the output bitlength of the one-way transformation required for the production of the y_i is quite large, i.e., 835 Kilobits. In contrast, most hash functions produce an output of ≈ 160 bits, given an arbitrarily large input size. To obtain such an output, Nyberg [Nyb96] suggests use of a hash function h in conjunction with a pseudo-random bit generator (PRBG) gen. The document x' is input to h whose short output is input as a seed to gen. From the PRNG gen, l bits can be produced. Therefore,

$$y' = gen(h(x')) \tag{2.5}$$

Notice that this large number of bits is only temporarily required at the time of computation of the authenticated accumulator a or during the verification of any values at some subsequent time. However, note that the size of a itself (which must be stored long-term) is also quite large. For the example given in Table 2.1, the size of a is $\frac{1}{10}$ the size of l (since $d = 10$).

2.4 Absolute Time Stamps

Stamping protocols in which an *absolute time stamp* (see Definition 2.3) is issued have the specific time at which a message was stamped contained in the time stamp (or at least a piece of data from which the time is directly and uniquely obtained). The granularity of this time may depend on the application in which the time is to be provided, but can include time to the nearest minute, second, etc. This timing information for one message can be obtained independently of the timing material provided for other messages. In this way, the production of an absolute time stamp is memoryless with respect to (i.e., independent of) other stamps that are produced. Assigning an absolute stamp in a distributed network assumes the existence of an authentic clock from which each potential verifying user maintains their local clocks. A reasonable amount of clock drift between respective clocks is assumed to be tolerated.

The schemes reviewed in this section are critiqued in Section 3.3. Generalizations regarding the production of absolute time stamps are discussed in Section 4.3.2. In this section, we review the current literature related to the provision of absolute time stamps. Subsection 2.4.1 reviews techniques in which a time stamp is produced by a central time stamp authority (or several such authorities). Subsection 2.4.2 reviews techniques in which individual user entities participate in the time stamp production.

2.4.1 Using a Time Stamping Service

Consider the time stamping protocol given in Protocol AB1 [HS91]. Notice the difference from Protocol SM1 where T need now maintain only the secrecy of his signature key, while each user must have (or be able to obtain) an authentic copy of T's signature verification key. The exact same scheme (termed the "Anonymous Service") was later given by Pinto and Freitas [PF96]. Yet these are both pre-dated by the presentation of a similar scheme by Merkle [Mer80, Mer82] (see Protocol NT1).

Protocol AB1 Centralized Time Stamp Construction [Mer80, HS91, PF96].

```
Stamping
Require: A time stamp authority $T$ that is trusted to provide the correct absolute
    time to submitted data.
Input: the hash $y = h(x)$ of a document $x$
Output: the time stamp $s = sig_T(y, t)$ produced by $T$
1: User $u$ sends the hash $y$ of the document $x$ to the time stamp authority T.
2: T verifies that the request is of the proper form, appends the current time $t$ and
   returns $t$ along with the T-signed response $s = sig_T(y, t)$.
3: $u$ verifies the signature applied by T and ensures that an acceptable time $t$ is
   associated with $y$ (where acceptable might mean a time greater than when $y$ was
   submitted and prior to when the T-signed response was received).
4: $u$ stores $\{x, s, t\}$ as the time stamp capsule.
Verification
Input: $\{x, s, t\}$
Output: indication of whether $x$ is time stamped with time $t$
1: user $v$ obtains $\{x, s, t\}$ from $u$, computes $y' = h(x)$ and ensures that $(y', t)$ was
   signed by T, producing $s$.
```

One concern with Protocol AB1 is that trust is required in a single trusted authority, for the honest production of time stamps, as well as the secure maintenance of private keying material. To reduce the trust required, Adams *et al.* [ACPZ98] build on top of Protocol AB1 with a scheme that uses so-called temporal data authorities (TDAs) for additional corroborations regarding the time t.

Continuing from the submission of y in Protocol AB1, T submits y to a number of TDAs (as specified by u), requesting additional, unpredictable "timing information" that associates y with a particular event. This timing information is appended to y, signed by the TDA and returned to T. Suggestions for the unpredictable timing information include [ACPZ98]

1. stock market information,

2. sports results,

3. official weather for a specific location,

4. lottery results,
5. birth or death announcements in specific newspapers,
6. headlines in specific newspapers,
7. information linking the request with previous and subsequent requests (e.g., hash values) that can be verified against information that is made public by the TDA, and
8. a signed packet from a secure time source.

Items 1 to 6 refer to what we define as implicit absolute time (see Definition 3.4). The use of such a time is discussed in Section 3.3.1. Item 7 refers to the relative ordering of time stamp submissions by linking them. The public storage allows a subsequent recovery of the corresponding absolute time so long as one is authentically provided along with the public storage (see Sections 4.3.3 and 4.2.1). Item 8 indicates the option in which the TDAs are used as the providers of corroborating absolute times. Note that each TDA can provide a different kind of temporal data. The entire scheme is described as Protocol AB2.

2.4.2 Decentralized Solutions: User-Constructed Stamps

In this section, we review three schemes that remove the requirement for a central time stamping service T. The stamps are constructed by individual users. Since a user has the capability to construct a stamp at any time, some additional properties must be used. These vary from distributing the stamp storage to distributing its construction.

As in Section 2.4.1, absolute time stamps are assigned to each of the submissions. For the first two schemes, the stamp computation is performed by the submitting user, while the stamp authentication is distributed (via a distributed storage). The third scheme distributes the construction of the stamp among a number of users. Storage of the stamp is maintained by the submitter of the document.

Protocol AB2 Centralized Time Stamp Construction with Additional Corroboration [ACPZ98].

$\boxed{\text{Stamping}}$

Require: Temporal data authorities (TDAs) are an optional enhancement, whose function is to accept a request from the time stamp authority and return a signed response over the received request with corroborative temporal data information appended.

Input: the hash $y = h(x)$ of a document x

Output: time stamp $s = sig_T(y, t, tempData_1, \dots, tempData_k)$

```
1: User $u$ sends the hash $y$ of the document $x$ and optionally, a request for $k$ additional corroborative times, to the time stamp authority T.
2: T verifies that the request is of the proper form. Should any corroborative temporal data be requested, T sends $y$ to $k$ TDAs.
3: Each $TDA_i$ accepts $y$, appends appropriate, unpredictable information, $tempData_i$, signs and returns the result $tempDataToken_i$ to $T$.
4: $T$ verifies the signature for $tempDataToken_i$, for each $i$, ensuring that it is computed over $y$. $T$ may, but is not required to verify the time provided by $tempData_i$.
5: $T$ appends the current time $t$ and returns $t$ along with the T-signed response $s = sig_T(y, t, tempDataToken_1, \dots, tempDataToken_k)$.
6: $u$ verifies the signature applied by T and ensures that an acceptable time $t$ is associated with $y$ (where acceptable might mean a time greater than when $y$ was submitted and prior to when the T-signed response was received).
7: $u$ may also verify that the times provided by the $tempData_i$ are within acceptable bounds, e.g., close to the time $t$.
8: $u$ stores $\{x, s, t, tempDataToken_1, \dots, tempDataToken_k\}$ as the time stamp capsule.
```

$\boxed{\text{Verification}}$

Input: $\{x, s, t, tempDataToken_1, \dots, tempDataToken_k\}$

Output: indication of whether s is a valid signature, i.e., for x time stamped with time t

```
1: user $v$ obtains $\{x, s, t, tempDataToken_1, \dots, tempDataToken_k\}$ from $u$, computes $y' = h(x)$ and ensures that $(y', t, tempDataToken_1, \dots, tempDataToken_k)$ was signed by T, producing $s$.
2: should $v$ also require additional corroborative evidence regarding the time of stamping, each $tempData_i$ may also be verified against the time $t$, possibly involving interacting with the appropriate TDA.
```

Protocol AB3 Broadcast-and-save time stamping technique [BdM91].

```
Stamping
Input: the hash $y = h(x)$ of a document $x$
Output: distributed storage of $y$
1: User $u$ broadcasts $y = h(x)$ to all other users at time $t$.
2: Every other user stores $y$, along with the time $t$ corresponding to when $y$ was
   received.
Verification
Input: $\{x, t\}$
Output: indication of whether $x$ is time stamped with time $t$
1: User $v$ obtains $x$, computes $y = h(x)$ and determines (by lookup in $v$'s own
   records) whether $y$ was recorded at time $t$.
```

Benaloh and de Mare [BdM91] make reference to a scheme which we describe as Protocol AB3. Stinson [Sti95] alters Protocol AB3 so that a centralized entity is required for the coordination of the storage of the time stamps, where this storage is distributed. In particular, the resultant time stamp is recorded in a publically verifiable medium, e.g., a local newspaper. Alternatives for authenticating the resultant time stamp are discussed in Section 4.2.1. Protocol AB4 describes Stinson's scheme. The use of so-called unpredictable information such as *pub* is discussed further in Section 3.3.

A solution that specifies more fully the storage responsibilities for the users was given by Haber and Stornetta [HS91]. Each user u_i has access to a secure signature scheme sig_{u_i} as well as a pseudo-random number generator *gen*. Protocol AB5 describes the steps taken for user u to obtain a time stamp. y is used as a seed for *gen*, where the deterministically generated output can be used to select some subset of users.

2.5 Relative Time Stamps

In this section, we review schemes in which the stamps for several rounds are linked, allowing for the provision of a temporal ordering of the stamps. The idea of linking is similar to the linking used in message passing protocols. (See Menezes *et al.*

Protocol AB4 Posting of time stamp to a distributed, publically verifiable medium [Sti95].

```
Stamping
Require: A publically verifiable storage medium for which information can be added
    by users, but not modified nor deleted.
Input: the hash $y = h(x)$ of a document $x$
Output: Storage of $u$'s signature $s = sig_u(y, t)$ in a publically verifiable medium.
1: At time $t$, user $u$ computes the digital signature $s$ over the concatentation of
   $y = h(x)$ and $pub$, i.e., $s = sig_u(y, pub)$ where $pub$ refers to public information that
   could not have been predicted before time $t$. For example, $pub$ might represent
   the hash of the closing values of the New York Stock exchange.
2: The triple $c = \{y, s, pub\}$ is published in a publically verifiable medium, e.g., local
   newspaper.
Verification
Input: $\{x, t\}$
Output: indication of whether $x$ is time stamped with time $t$
1: User $v$ obtains $\{x, t\}$ from user $u$, computes $y = h(x)$, obtains the $pub$ information
   corresponding to time $t$ and searches for (and determines the correctness of) the
   entry $\{y, s, pub\}$ in the publically verifiable medium.
```

Protocol AB5 Decentralized time stamp construction with local storage [HS91].

```
Stamping
Require: Each user requires possession of or access to verification keys of other users.
    The participation of users is required for the production of a time stamp.
Input: the hash $y = h(x)$ of a document $x$
Output: $[(y, u), (z_1, \ldots, z_k)]$ where $z_i = sig_{u_i}(t, y)$
1: For document $x$, $u$ computes $y = h(x)$ as well as $gen(y) \rightarrow (u_1, \ldots, u_k)$ denoting
   that the output of the pseudo-random function $gen(y)$ is used to select to some
   subset of users.
2: $u$ gives $y$ to each $u_i$.
3: Each recipient computes and returns to $u$, $z_i = sig_{u_i}(t, y)$.
4: $u$ stores $[(y, u), (z_1, \ldots, z_k)]$.
Verification
Input: $\{x, t, (z_1, \ldots, z_k)\}$
Output: indication of whether $x$ is time stamped with time $t$
1: User $v$ obtains $[(y, u), (z_1, \ldots, z_k)]$ from $u$, computes $(u_1, \ldots, u_k)$ from $gen(y)$.
2: $v$ verifies each of the $z_i$ (i.e., verifies that the time in each certificate is within the
   time in question).
```

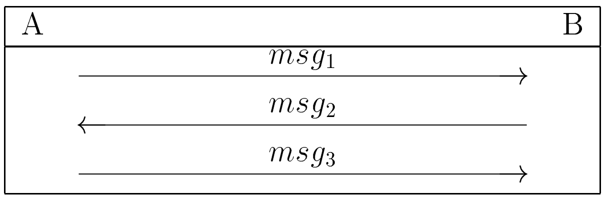

Figure 2.2: Generic Message Passing

[MvOV97, Chapter 10], Meyer *et al.* [MM82, Chapter 8] or Davies *et al.* [DP84, Chapter 5] for background.) The goal of linking messages is to prevent attacks such as message replay, message insertion, message deletion. For example, consider the simple message exchange between users A and B given in Figure 2.2. One would like to prevent, for example, the malicious insertion of $msg_{2'}$ between messages msg_1 and msg_2. One of the means for preventing such attacks is the use of *time-variant parameters* (TVPs). Random numbers, sequence numbers and date or time stamps are examples of TVPs.

The analogous attack for relative time stamping protocols is the insertion of a message with a false time stamp into the current temporal ordering of messages. Just as TVPs allow for the distinguishing of several protocol instances in a message passing protocol, they can also be used to distinguish (or order) one round from another in a time stamping protocol. Each time stamp issued (i.e., the result of a round) can be thought of as a single message in a large, ongoing time stamping protocol.

The basic linking relation can be described by the simple recurrence relation

$$a_r = h(a_{r-1}, y_r)$$

where a_r is the time stamp for the rth round and $y_r = h(x_r)$ is the hash of the document to be stamped during the rth round (or alternatively, may represent the output of a group hash). This equation provides an ordering of the y_i. Further authentication of the resultant a_r allows for a recovery of this ordering during stamp verification. Variations result from the particular construction of a_{r-1} and the amount of user versus central entity cooperation required for the validation of a relatively ordered time stamp. These linking techniques are discussed further in Section 3.4.

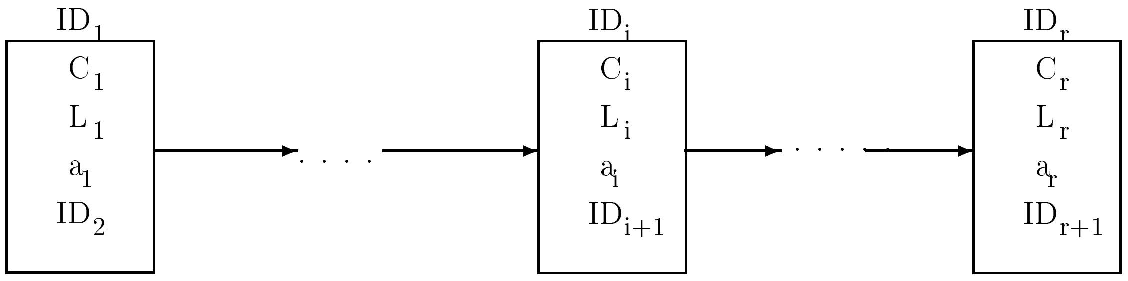

Figure 2.3: Chain of time stamps in Protocol RL1.

Haber and Stornetta [HS91] provide for a recovery of the relative ordering of the documents by requiring user interactions during stamp verification. The protocol uses a central time stamping service T that requires no record-keeping. Each user stores information pertaining to their own submission and as well, information explicitly defining a relationship with the stamp produced immediately prior to their own. Here, r denotes the rth round, where one document is stamped per round. Protocol RL1 describes their protocol. The resultant chain of stamps is shown in Figure 2.3. In Section 4.4.2, the security of this protocol is analyzed.

A similar recursive linking to Protocol RL1 is given by Pinto and Freitas [PF96] with Protocol RL2. The distinguishing feature is the use of a central time stamp authority when validating the temporal order of two stamps. A more interesting distinction regards the differing use of the relative ordering of the stamps. Whereas Protocol RL1 (and likewise Protocol RL3) uses the linking as a means for detecting a rogue time stamp authority, Protocol RL2 (and likewise Protocols RL4 and RL5) uses the linking to allow for a subsequent temporal measurement between two input data. This point is discussed further in Section 4.3.3.

The extension of the linking element can be used to explicitly reference more than one of the previous stamps. The purpose of this technique is to reduce the required number of potential interactions between users (as required specifically for Protocol RL1), as well as increasing the work and potentially the number of conspirators required by an attacker attempting to produce a false stamp.

Protocol RL3 describes the Haber and Stornetta [HS91] variant of Protocol RL1. Its intention is to remove the requirement for users to keep all of their time stamps (in anticipation of their participation in future challenges). A *challenger* can now

Protocol RL1 Haber-Stornetta Linking [HS91].

$\boxed{\text{Stamping}}$

Note: As indicated by Haber and Stornetta, the time stamp authority T need not perform any record keeping nor be trusted since T is unable to back or forward date stamps. (See Section 4.4.2 for and indication as to why this claim is false.)

Input: $y_r = h(x_r)$ is the hash of document x_r

Output: a_r, ID_{r+1}

```
1: User $u$ sends $y_r = h(x_r)$, for document $x_r$ and $ID_r = ID_u$ where $ID_u$ is the
   unique identification for user $u$, to the time stamp authority T.
2: T computes the certificate $a_r$ for this $r$th submission, namely $a_r = sig_T(C_r)$,
   where
```

$$\begin{aligned} C_r &= (r, t_r, ID_r, y_r; L_r) \\ L_r &= (t_{r-1}, ID_{r-1}, y_{r-1}, H(L_{r-1})) \end{aligned}$$

```
   and $H$ is a collision resistant hash function, and $t_r$ is the absolute time of the
   submission. $L_r$ is referred to as the *linking information* and contains the respective
   information pertaining to the submission from the previous round.
3: Upon receiving the next request for a stamp from user $v$, T sends time stamp
   $(a_r, ID_{r+1} = ID_v)$ to $u$ who verifies that the signature has been computed prop-
   erly and saves the time stamp for future use.
```

$\boxed{\text{Verification}}$

Input: (a_r, ID_{r+1})

Output: indication of whether the absolute time t_r associated with y_r in a_r is trustworthy

```
1: User $v$ obtains $(a_r, ID_{r+1})$ from $u$.
2: $v$ verifies the mathematical correctness of the signature $a_r$.
3: To verify that there hasn't been a collusion with T (i.e., T did not use a fake time
   $t_r$), $v$ contacts $ID_{r+1}$ and obtains $(a_{r+1}, ID_{r+2})$ where
```

$$a_{r+1} = sig_T(r+1, t_{r+1}, ID_{r+1}, y_{r+1}; L_{r+1})$$

```
   and checks that $L_{r+1}$ contains both $y_r$ and $H(L_r)$.
4: Optionally, $v$ may also check $ID_{r+2}$'s stamp or verify previous stamps using $ID_{r-1}$
   (as it is included in $L_r$).
```

Protocol RL2 Recursive Hash Linking [PF96].

$\boxed{\text{Stamping}}$

Input: $y_r = h(x_r)$ is the hash of document x_r

Output: $\{a_r, C_r, L_r, t_r, r\}$

1: User u submits $y_r = h(x_r)$ to a time stamping authority T who computes the time stamp $a_r = sig_T(C_r)$, where $C_r = (r, t_r, y_r, L_r)$, where t_r is the time of submission of y_r and the linking element is computed as

$$\begin{aligned} L_1 &= IV \\ L_r &= h(a_{r-1}, L_{r-1}), r \geq 2, \end{aligned} \tag{2.6}$$

where IV refers to an initial value.

2: $\{a_r, C_r, L_r, t_r, r\}$ are returned to u by T and maintained in a database by T.

$\boxed{\text{Verification}}$

Input: $\{a_i, C_i, L_i, x_i, t_i, i\}, \{a_j, C_j, L_j, x_j, t_j, j\}$

Output: A determination of the temporal ordering of x_i and x_j.

1: User v computes $y_i = h(x_i)$ and $y_j = h(x_j)$ and validates the signatures on the T-signed a_i and a_j.

2: v requests $s_{ij} = \{$set of stamps from round i to $j\}$ from T.

3: v continues the recursive hash computation (as in (2.7) above) from L_i, using a_i and s_{ij} to see if L_j is the result. If so, v concludes that a_i contributed to the computation of a_j and was therefore stamped prior to a_j. If not, v repeats the process starting at L_j to determine if a_j contributed to the computation of a_i and was therefore stamped prior to a_i.

Protocol RL3 Haber-Stornetta Extended Linking [HS91].

```
[Stamping]
Input: $y_r = h(x_r)$ is the hash of document $x_r$
Output: $a_r, (ID_{r+1}, \ldots, ID_{r+k})$
1: Similar request and construction to Protocol RL1 except that the linking information is now
       $L_r = [(t_{r-k}, ID_{r-k}, y_{r-k}, H(L_{r-k})), \ldots, (t_{r-1}, ID_{r-1}, y_{r-1}, H(L_{r-1}))]$
2: Upon receiving the next $k$ requests for a stamp from user $v$, T sends $(a_r, (ID_{r+1}, \ldots, ID_{r+k}))$ to $u$ who verifies the time stamp $a_r$ by verifying the mathematical correctness of the signature over $C_r$ and verifies that the information signed is consistent with the information that was submitted.
[Verification]
Input: $(a_r, (ID_{r+1}, \ldots, ID_{r+k}))$
Output: indication of whether the absolute time $t_r$ associated with $y_r$ in $a_r$ is trustworthy
1: User $v$ obtains $(a_r, (ID_{r+1}, \ldots, ID_{r+k}))$ from $u$.
2: $v$ verifies the validity of the time stamp $a_r$ by validating the mathematical correctness of the signature applied by $T$.
3: To verify that there hasn't been a collusion with T, $v$ contacts any of the next $k$ clients, $ID_{r+i}$, $i = 1, \ldots, k$ and ensures that the time stamp information for $a_r$ is included in the linking information of the time stamp for these clients.
4: Optionally, $v$ may also verify the correct inclusion of previous time stamping material in $L_r$ by consulting with any of the $k$ clients included in $u$'s own time stamp.
```

check any of the previous or next k clients, ID_{r+i}, $i = 1, \ldots, k$. Inserting a document presumably requires finding k simultaneous collisions for the hash H. In other words, referring to Figure 2.4, notice that an attempt to backdate a stamp immediately prior to a_{r-k} would require finding collisions for the inputs to the computations of the linking elements $L_{r-k}, \ldots, L_r$ (since each contains reference to the k most previous stamps). A noticeable drawback is that the size of L_r (for Protocol RL3) is quite large. Figure 2.4 displays the relationship between a particular stamp and its k "children".

Pinto and Freitas [PF96] use some of the techniques of Section 2.3 and apply them to Protocol RL3 in order to reduce the size of the linking element. The time stamp for a single document y returned by T is a_r as in Protocol RL3 where now we have

$$C_r = (r, t_r, y_r, L_r) \tag{2.7}$$

so that the only difference from both Protocol RL1 and Protocol RL3 is that L_r is different.

This first method is Protocol RL4. There are obvious problems with storage here as well. Note that all the a_i must be stored by T (allowing computation of new linking elements) and the linking elements stored by each user are prohibitively large. This scheme has the advantage that disputes can be resolved between two users, without the cooperation of T or other users.

An alternative in which the storage for each user is reduced to a fixed size for each stamp is Protocol RL5. Verification involves the intervention of T (to determine if $a_i \in A_j$ since only a (one-way) hash of the linking element is available to users). This solution provides for a constant size for the linking element, independent of the number of time stamps produced. There are several concerns with this scheme. As above, all the a_i must be stored by T. Beyond this, the server is required to recompute the L_i based on *all* previously input time stamps. Lastly, disputes require verification of all stamps from a_1 to the disputed stamp. (Though they could make this simpler by having L_i include a fixed number of stamps rather than an unbounded number as in Protocol RL3.)

A suggestion made by Pinto and Freitas [PF96] for controlling $|s_{ij}|$ is to insert intermediate stamps $a_I < a_{II}$ (see Definition 4.17 in Section 4.3.3) whose relative

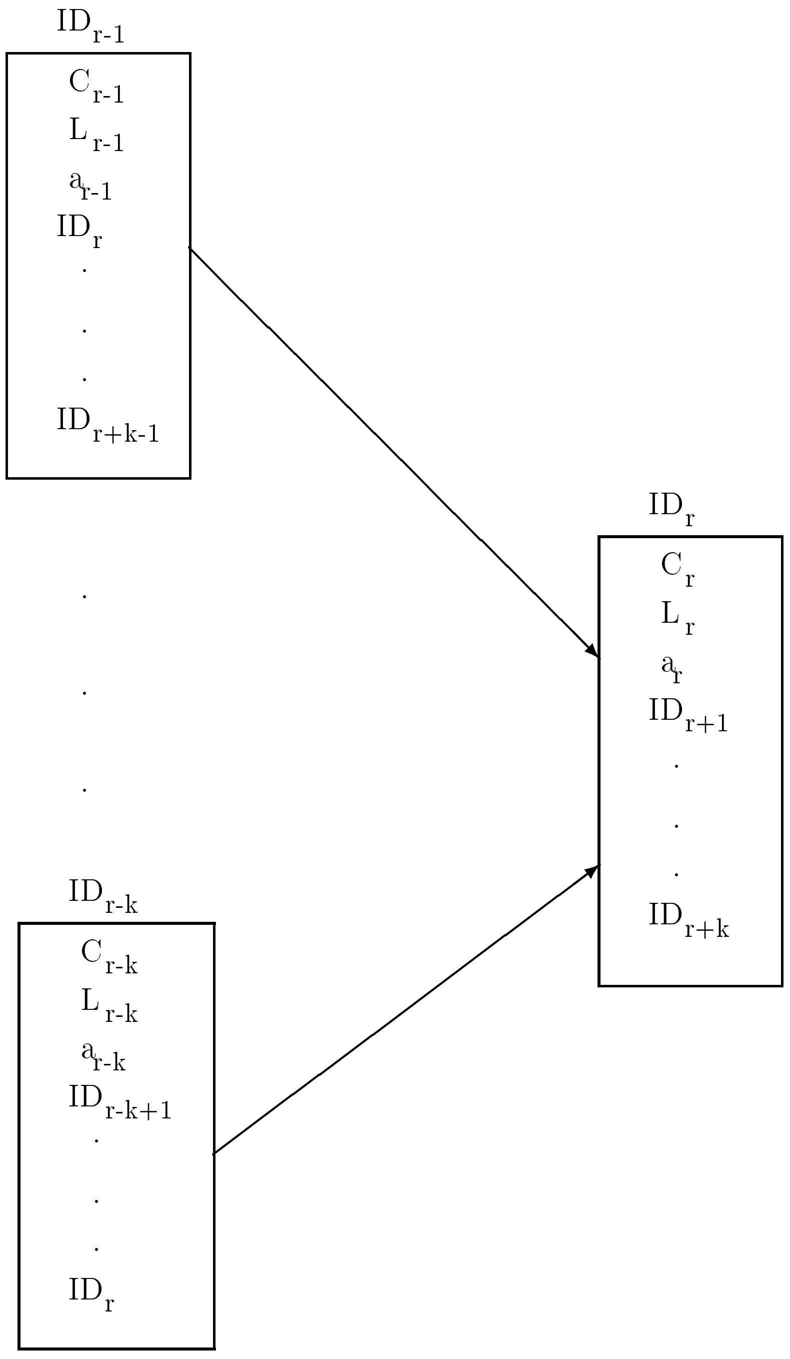

Figure 2.4: Chain of time stamps in Protocol RL3. L_r is explicitly computed as a function of the k most immediately previous stamps, i.e., those stamps held by users $ID_{r-k}, \ldots, ID_{r-1}$; the arrows are used to indicate this dependence. User ID_r (owner of stamp a_r) maintains the IDs of the next k stamp owners, i.e., $ID_{r+1}, \ldots, ID_{r+k}$.

Protocol RL4 Cumulative Extended Linking [PF96].

$\boxed{\text{Stamping}}$

Input: $y_r = h(x_r)$ is the hash of document x_r
Output: $\{a_r, C_r, L_r, t_r, r\}$

```
1: User $u$ submits $y_r = h(x_r)$ to a time stamping authority $T$ who computes the time
   stamp $a_r = sig_T(C_r)$ (for $C_r$ from (2.7)) where the linking element is computed
   as
       $L_r = (a_1, a_2, \ldots, a_{r-1}).$
2: $\{a_r, C_r, L_r, t_r, r\}$ is returned to $u$ by T.
```

$\boxed{\text{Verification}}$

Input: $\{a_i, C_i, L_i, x_i, t_i, i\}, \{a_j, C_j, L_j, x_j, t_j, j\}$
Output: A determination of the temporal ordering of x_i and x_j.

```
1: User $v$ computes $y_i = h(x_i)$ and $y_j = h(x_j)$ and validates the signatures on the
   T-signed $a_i$ and $a_j$.
2: User $v$ determines whether $a_i \in L_j$ or $a_j \in L_i$.
3: The former implies that $a_i$ was constructed before $L_j$ was produced, hence, $x_i$
   was stamped before $x_j$. The latter implies the opposite conclusion.
```

Protocol RL5 Cumulative Hash Extended Linking [PF96].

$\boxed{\text{Stamping}}$

Input: $y_r = h(x_r)$ is the hash of document x_r
Output: $\{a_r, C_r, L_r, t_r, r\}$

```
1: Same as Protocol RL4 except that the linking element is computed as
       $A_r = (a_1, a_2, \ldots, a_{r-1})$
       $L_r = h(A_r).$
```

$\boxed{\text{Verification}}$

Input: $\{a_i, C_i, L_i, x_i, t_i, i\}, \{a_j, C_j, L_j, x_j, t_j, j\}$
Output: A determination of the temporal ordering of x_i and x_j.

```
1: User $v$ computes $y_i = h(x_i)$ and $y_j = h(x_j)$ and validates the signatures on the
   T-signed $a_i$ and $a_j$.
2: $v$ requests $A_i$ and $A_j$ from T.
3: $v$ determines whether $L_i = h(A_i)$ and $L_j = h(A_j)$.
4: $v$ determines whether $a_i \in A_j$ or $a_j \in A_i$.
5: The former implies that $a_i$ was constructed before $L_j$ was produced, hence, $x_i$
   was stamped before $x_j$. The latter implies the opposite conclusion.
```

order is known *a priori* so that

$$(a_i < a_I) \wedge (a_{II} < a_j) \rightarrow a_i < a_j.$$

In other words, not all intermediate stamps between a_i and a_j would be required for validation. Rather, only the stamps from a_i to a_I and from a_{II} to a_j would be required. The stamps from a_I to a_{II} would not be required. Intermediate stamps are discussed further in Section 4.3.3.

2.6 Concluding Remarks

The techniques described in this chapter demonstrate how a time can be authentically associated with a string of bits. Although these bits may have associated semantics, this meaning is irrelevant to the entity performing the stamping. In other words, though data with a specific form might be submitted to the time stamper, this data is not interpreted prior to the application of time. In Chapter 5 for example, the time stamping of digital signatures is examined. Interpreting the semantics of the input can be handled by a digital notary, and is discussed in Section 5.4.

This chapter presented a review of the state-of-the-art in the literature with respect to time stamping. (Work related to the linking of time stamps has recently been presented by Buldas *et al.* [BLLV98].) Critical analysis of the schemes presented in this chapter is presented in Chapter 3. Generalizations and extensions are presented in Chapter 4. Although there are many applications in which a time stamp might be useful, e.g., patent submissions, electronic commerce, chapters 5 and 6 deal specifically with the temporal authentication of certificate-based digital signatures.

Chapter 3

Critical Analysis of Previous Work

In this chapter, we critique the time stamping protocols from Chapter 2. Beyond classifying the schemes as accomplished by the ordering into different sections (i.e., distinguishing those schemes providing absolute and relative time stamps), various other analyses can be performed. In Section 3.2, we examine the methods of group hashing reviewed in Section 2.3. Several new properties are defined and comparisons are made between the various techniques. In Section 3.3 and Section 3.4, the respective absolute and relative time stamping schemes of Section 2.4 and Section 2.5 are examined.

3.1 Critique Metrics

In this section, we briefly review some of the properties that are used in the remaining sections of this chapter for the critical analysis of the time stamping protocols from Chapter 2. More specifically, we consider the traditional measures of storage, communication and computational complexity.

We also note that the participation of the trusted authority is an important concern. In some cases, an authority may be required for only a one-time setup. In other cases, the authority may participate a fixed number of times (over the life of the protocol) or even an indeterminate number of times. The role of the authority may also be restricted to being off-line (where real-time participation in the protocol is not required) or on-line (where real-time participation is required). The variance of these roles will have an affect on the efficiency of the protocol, as well as contributing to the complexity and therefore, potential for disputes.

Storage

The amount of storage required for a particular protocol can typically be measured relative to various roles. For example, each user may only be concerned with the amount of storage required at their own site. Should the protocol also involve the use of a centralized entity, the amount of storage maintained by this entity is important as well. Beyond individual entities, the amount of storage over the entire system can also provide a measure of efficiency. For example, each user may have a reasonable amount of local storage but if each store the same information, the storage over the system as a whole may be considered unreasonable.

More precisely, suppose that a centralized version of some protocol required w bits of storage at a central site. In a distributed version in which the storage is uniformly distributed, we would expect that, on average, each of the m distributed entities participating in the protocol would require at least $\frac{w}{m}$ bits of storage.

One may also introduce temporal quantifications that determine the length of time that the storage of particular data must be maintained. For example, are users allowed to throw away unwanted stamps, or must they necessarily be stored indefinitely for the proper functioning of the time stamping protocol?

Beyond the quantitative storage measures, we also have some qualitative measures. For example, of the information stored, which must be authentically stored and/or have its privacy maintained. Such storage should be clearly identified and minimized as it creates additional overhead, e.g., costs.

Communication Complexity

It is important to measure the number of interactions and amount of entity involvement required for each step of the time stamping protocol. A quantitative measure of the number of bits communicated is not sufficient. One must also be wary of *how* users are called to participate in the protocols. For example, does the Stamping Protocol (see Section 2.1.1) require that users who may not be submitting a document for a particular round, are still required to participate in the protocol. As well, are users required to participate in the verification of the time stamps of other users? It would be advantageous for users to only participate in protocols when the nature of the protocol is directly related to a specific time stamp of theirs.

Computational Complexity

Computational complexity concerns a measure of the cost of producing, challenging, or verifying a time stamp. A reasonable measure might be in terms of the number of primitive operations required to complete a particular protocol. These could be, for example, a measure of the number of applications of a hash function (or internal stages of the hash, e.g., more time required for the hash of longer messages), signature algorithm and verification algorithms (see Section 2.1.2). Such an analysis allows one to compare the efficiency of those schemes which are not instantiated with identical hash functions or signature schemes.

Architectural Complexity

Related to the implementation and maintenance of the schemes is their architectural complexity. This has an impact on the communication complexity during the running of the protocols as well as on the initial development costs and potential introduction of security flaws for the more complex solutions. Also of interest are the potential for denial of services as well as simply bottlenecks that may result from reliance on a central server, i.e., a single point of failure.

3.2 Critique of Group Hashing

The group hashing techniques reviewed in Section 2.3 are used to allow for so-called document membership tests. Given a document, it is necessary to be able to verify whether or not the document in question contributed to the construction of the stamp for a particular round. This is done by first determining whether the document contributed to the group hash computation. Subsequent testing is used to recover the time associated with the group hash result.

The motivation for a granularity whereby multiple documents are used to produce a single stamp is twofold:

1. to allow for a decentralized stamp computation;
2. to decrease the amount of storage by using a single value as representative of an entire round of documents.

The first point results from the fact that since there is no secret key involved in the computation of the stamp (at least for the schemes reviewed in Section 2.3), any user can compute the stamp given the document submissions from all participants. Indeed, the intention for the creators of some of the schemes (e.g., Benaloh and de Mare [BdM93]) was to remove the requirement of trusting a central authority for the computation of the round result. Note that for the second item above, the amount of storage is decreased only if the size of the stamp is smaller than the sum of the size of the document submissions. Ideally, the stamp is equivalent in size to a given submission, or at least independent of the size or number of submissions. As evidenced in Section 2.3, achieving this property may affect the amount of storage required for each user. This property is discussed further in Section 3.2.2.

Centralized versus Decentralized Group Hashing

For a decentralized computation, there is an additional amount of communication required for participants. This is especially true for those schemes in which an ordering of the submissions is required, i.e., there needs to be consensus (either unilateral or multilateral) on the ordering of the inputs to the group hashing function. Thus, for Protocols GH4 and GH5 (where no such ordering is required), there appears to be the advantage of a more efficient stamp creation protocol (at least relative to those schemes in which an ordering is required). On the other hand, use of Protocol GH1, Protocol GH2 and Protocol GH3 require additional communication in a decentralized protocol (to reach a consensus on the order of inputs to the group hash function) since the order of the inputs to their group hash functions, does matter.

A centralized variation in which a trusted entity T computes the group hash result for the round, would be similar to the application of multiple instances of Protocol AB1 except that storage at T or at an alternative source is lessened by a factor of m (should the size of the resultant stamp be proportional to the size of a submission), where m documents contribute to the computation for a given round. The reason for this is that T would perform an authentication over the group hash result rather than a different application over each user's submission.

Authenticating the group hash result

Group hashing provides no message or temporal authentication on its own. Subsequent authenticity provisions are therefore very important. In a decentralized version, it is not clear whose "authentic" version of the resultant stamp for a given round, would "succeed" in the event of a dispute. In a centralized variation, the location and maintainer of the stamp's authenticity is equally as important. The group hash computations reviewed in Section 2.3 are not keyed by any secret parameter. Hence, anyone can compute what appears to be a valid time stamp, corresponding to a document of their choice. Therefore, the maintenance of the authenticity of the resultant stamp is of the utmost importance. This point is discussed further in Section 4.2.1.

3.2.1 Formalizing Group Hashing

In this subsection, we formalize the notion of group hashing, and discuss some of the requirements for the components of a group hash protocol.

Definition 3.1 A *group hash scheme* $\mathcal{G}$ is a seven-tuple $(\mathcal{Y}, \mathcal{A}, \mathcal{B}, G, F, V, W)$, where the following conditions are satisfied:

1. $\mathcal{Y}$ is a finite set of possible *messages*;

2. $\mathcal{A}$ is a finite set of *group hashes*;

3. $\mathcal{B}$ is a finite set of *membership values*;

4. Let $G : \mathcal{Y} \times \cdots \times \mathcal{Y} \to \mathcal{A}$ be a *group hash function*, and $F : \mathcal{Y} \times \cdots \times \mathcal{Y} \to \mathcal{B}$ be a *membership production function*. For a *verification test function* $V : \mathcal{A} \times \mathcal{Y} \times \mathcal{B} \to \{\text{true, false}\}$, and *verification function* $W : \mathcal{Y} \times \mathcal{B} \to \mathcal{A}$, and for every $y_i \in \mathcal{Y}$, $a = G(y_1, \ldots, y_m)$, and $member_{y_i} = F(y_1, \ldots, y_m)$

$$V(a, y_i, member_{y_i}) = \begin{cases} \text{true} & \text{if } W(y_i, member_{y_i}) = a \\ \text{false} & \text{if } W(y_i, member_{y_i}) \neq a \end{cases}$$

■

Referring to Definition 3.1, there are two fundamental operations related to the production of a group hash given a set of data $Y = \{y_1, \ldots, y_m\}$. The first is the *compression* of the data which involves the production of the value a used to represent the set Y:

$$a = G(Y) = G(y_1, \ldots, y_m).$$

For Protocol GH4, $a = {x_0}^{y_1 y_2 \cdots y_m} \bmod n$ for publically known x_0 and suitable composite n. The second operation is the computation of

$$member_{y_i} = F(y_1, \ldots, y_{i-1}, y_{i+1}, \ldots, y_m) \tag{3.1}$$

which, if it exists, is additional information necessary for document membership verification. For Protocol GH4, $member_{y_i} = {x_0}^{y_1 \cdots y_{i-1} y_{i+1} \cdots y_m} \bmod n$. As indicated, $member_{y_i}$ is a function of data other than the user's own y_i.

Verification of an items "membership" in a group hash value, involves computing $a^{(i)} = W(y_i, member_{y_i})$ and determining whether $a = a^{(i)}$. For Protocol GH4, $a^{(i)} = (member_{y_i})^{y_i} \bmod n$. Verification occurs at a subsequent time when the owner u of some y' wants to demonstrate that $y' = y_i$ for some $i \in \{1, \ldots, m\}$. This is done indirectly (as opposed to directly which would check for membership of y' in Y) using the representation a for Y and the membership verification function V. We have

$$V(a, y', member_{y'}) = \begin{cases} \text{true} & \text{if } y' \in Y \\ \text{false} & \text{otherwise.} \end{cases}$$

This verification is performed as above whereby $a^{(')} = W(y', member_{y'})$ is computed and tested for equivalence to a. Recall Protocol GH1 where $a = h(y_1, \ldots, y_m)$ and $member_{y_i} = (y_1, \ldots, y_{i-1}, y_{i+1}, \ldots, y_m)$. Verification involves computing $a^{(i)} = h(y_1, \ldots, y_{i-1}, y_i, y_{i+1}, \ldots, y_m)$ and determining if $a = a^{(i)}$.

G must be collision-resistant so that one cannot find an alternate set $Y' \neq Y$ such that $G(Y) = G(Y')$. If this were not so, one might, for example, for a set $Y = \{y_1, \ldots, y_i, \ldots, y_m\}$, find a set $Y' = \{y_1, \ldots, y', \ldots, y_m\}$ such that $G(Y) = G(Y')$. Interestingly, the order of the input to G need not be important. For example, although $G(\{y_1, y_2\}) = G(\{y_2, y_1\})$ for a commutative G, no real collision has occurred since no $y' \notin Y$ has been found that succeeds verification. As well, for a given group hash a, and $y' \notin Y$, it should be difficult to determine any $member_{y'}$ such that $V(a, y', member_{y'})$ is true. In other words, it should be computationally infeasible to compute $a' = W(y', member_{y'})$ such that $a^{(')} = a$ if $y' \notin Y$. Thus, not only is $V()$ one-way in some sense, but $member_{y'}$ should be computationally infeasible to obtain for a $y' \notin Y$, with respect to a given group hash value a, i.e., at least F should be collision-resistant.

3.2.2 Storage Analysis

In this subsection, comparisons are made between the storage requirements for each of the group hashing techniques described in Section 2.3. Consider first the bit size $|a|$ of the group hash value a. Two categories can be identified based on its size:

1. $|a|$ is dependent on the number of submissions. For example, $|a| = m \cdot |y_i|$ in the most trivial scheme where $a = Y$.

2. $|a|$ is fixed. There are two sub-cases here. In the first, $|a|$ is independent of the size of or number of inputs to G. This is true for Protocol GH4, where $|a|$ is dependent upon a chosen security parameter and as well for Protocol GH5 where $|a| = cN$, for constant c where the restriction is that $m \leq N$. However, one must be careful to note that $|Y| = m$ is also dependent (i.e., bounded, though not tightly) on this security parameter. Secondly, we have $|a| = |h()|$

where $|h()|$ is the size of the output of the hash used (as in Protocols GH1, GH2 and GH3).

For Protocol GH4, though the stamp size is fixed, it is noticeably larger than the size of the submissions. For example, in comparison to Protocols GH1, GH2 and GH3 where $|h()|$ will be approximately 160 bits in practice, k may be 1024 or even 2048 bits. For Protocol GH5, even with two compressions by factors of d and m respectively, a "large" resultant round value is still obtained. Recall that the submission of a user is expanded to an l-bit y_i where

$$l = \overbrace{(Net)}^{r} \underbrace{(\lg N)}_{d}.$$

Despite the fact that each y_i is reduced by a factor of d, and there is an m-to-1 compression from these m r-bit b_i to a, we still have $|a| = r$. Now $r = Net$ where N is an upper bound on the number of messages that can be submitted. Referring to Table 2.1, for $l = 1.4M$, we have $|a| = 140K$ which would only provide compression (i.e., less storage efficiency than simply storing the original document submissions) if at least 875 160-bit inputs were available.

The same concerns with regard to size follow for $|member_{y_i}|$. For Protocols GH1 and GH2, $|member_{y_i}| = O(m \cdot |h()|)$ while $|member_{y_i}| = O((\lg m) \cdot |h()|)$ for Protocol GH3. For Protocols GH4 and GH5, $|member_{y_i}| = k$ and $|member_{y_i}| = 0$, respectively, where $k = |n| = \lg n$ is the length in bits of the security parameter (modulus). These results are summarized in Table 3.1.

In Table 3.2 the results from Table 3.1 are instantiated with $m = 16$, $m = 256$, and $m = 1024$ respective documents. Noteworthy increases include the linear increase in $|member_{y_i}|$ for Protocols GH1 and GH2 compared to the logarithmic increase with Protocol GH3. Each has a constant size for the group hash value a as does Protocol GH4. Protocols GH4 and GH5 have a constant size for $|member_{y_i}|$ where the latter protocol it is in fact 0.

In Table 3.3, the efficiency of the storage over the system as a whole is measured, relative to a control scheme in which each user maintains their own y_i as well as its storage being authentically maintained at a central repository. Protocols GH3 and

	Size in bits	
Hash Protocol	hash result a	verification info. $member_{y_i}$
GH1: Cumulative	$\|h()\|$	$(m-1) \cdot \|h()\|$
GH2: Recursive	$\|h()\|$	$(m-1) \cdot \|h()\|$
GH3: Tree	$\|h()\|$	$(\lg m) \cdot \|h()\|$
GH4: Exponentiation	k	k
GH5: Bit	cN	0

Table 3.1: Comparison of Storage for Group Hashing Techniques. There are m data submitted for group hashing, each of size $|h()|$, where $|h()|$ is the number of output bits from the particular hash function used. $k = |n| = \lg n$ is the length in bits of the security parameter (modulus). N is the implementation-dependent upper bound on m and c is a constant ranging from 135 to 200 for practical purposes. See Table 2.1 for more details regarding the parameters of Protocol GH5.

	$m = 2^4 = 16$		$m = 2^8 = 256$		$m = 2^{10} = 1024$	
Hash Protocol	$\|a\|$	$\|member_{y_i}\|$	$\|a\|$	$\|member_{y_i}\|$	$\|a\|$	$\|member_{y_i}\|$
GH1: Cumulative	160	2.4K	160	41K	160	164K
GH2: Recursive	160	2.4K	160	41K	160	164K
GH3: Tree	160	640	160	1.3K	160	1.6K
GH4: Exponentiation	1K	1K	1K	1K	1K	1K
GH5: Bit	2.2K	0	34.8K	0	139.2K	0

Table 3.2: Specification of Table 3.1 Results. $|a|$ refers to the size (in bits) of the resultant group hash value computed with input $(y_1, \ldots, y_m)$ where each $y_i = h(x_i)$ is a 160 bit hash over a document x_i of arbitrary length. $|member_{y_i}|$ refers to the size (in bits) of data that each user must maintain (not including storage of their own y_i or x_i) to allow later demonstration that y_i did indeed contribute to the construction of a. For Protocol GH4 a 1024 bit composite integer is assumed. For Protocol GH5, we assume that $N = m$, $t = 50$ where $|a| = r = Net$. (See Table 2.1 for further details regarding the parameters for Protocol GH5).

	$m = 2^4 = 16$		$m = 2^8 = 256$		$m = 2^{10} = 1024$	
Hash Protocol	$actual$	$\frac{actual}{control}$	$actual$	$\frac{actual}{control}$	$actual$	$\frac{actual}{control}$
GH1: Cumulative	41.1K	8.03	10.5M	128	167.7M	512
GH2: Recursive	42K	8.03	10.5M	128	167.7M	512
GH3: Tree	13K	2.53	0.4M	4.50	1.8M	5.50
GH4: Exponentiation	20K	3.90	0.3M	3.71	1.2M	3.70
GH5: Bit	4.8K	0.92	75.8K	0.92	0.3M	0.92

Table 3.3: Overall System Storage Efficiency for Group Hashing Techniques. The overall system storage for each technique is computed as $actual = m \cdot |h()| + m \cdot |member_{y_i}| + |a|$ where the first term accounts for the storage by each user of their own $y_i = h(x_i)$, the second accounts for the storage by each user of their $member_{y_i}$ while the third accounts for the storage of the resultant group hash by a central authority. The values for each term are taken from the corresponding examples in Table 3.2. $\frac{actual}{control}$ refers to the ratio of the protocol's overall system storage to the control scheme in which a copy each user's y_i is maintained by themselves as well as a central authority; the storage for this technique is $control = 2m \cdot |h()|$.

GH4 are the most efficient of the schemes in which the size of the group hash value a is constant (with respect to changes in $|m|$). Protocol GH4 is the most efficient based on the metric described in Table 3.3 though as evidenced by Table 3.2, the size of a increases linearly with the size of m.

3.2.3 Incremental Group Hashing

In this subsection, we define the concept of a group hash as being *efficiently incremental* or *efficiently decremental*, and examine the schemes of Section 2.3 with respect to these properties.

Bellare, Goldreich and Goldwasser [BGG94] introduced the concept of *incremental cryptography.* The basic idea is to allow repeated function computations to be computed efficiently in the case that the input has only changed slightly. Particularly for hashing, given the hash of a message and a subsequent modification of the message, the time required to update the hash (to produce a new hash over the modified message) should be "proportional" to the amount that the message has been modified.

We can also define such efficient incrementality for group hashing. The "message" modified in this case is the set of data input to the group hash. More specifically, if the original group hash was applied to the set $Y = \{y_1, \ldots, y_m\}$ then the modified set to which a group hash result would be required would be either $Y' = \{y_1, \ldots, y_m, y_{m+1}\}$ for the appendage of an item y_{m+1} or $Y' = \{y_1, \ldots, y_{i-1}, y_{i+1}, \ldots, y_m\}$ for the removal of an item. Although not specifically applicable to the production of a time stamp, group hashing with an incremental property is useful for the group hash of revocation information, related to the time stamping and non-repudiation of digital signatures (see Section 5.2.3).

Definition 3.2 A group hashing function G is *efficiently incremental* if for an l-bit element $y_{m+1} \in \mathcal{Y}$, where $f(l)$ time is required to compute $G(y_{m+1})$ for some function f, then at most $af(l) + b$ time is required to compute $G(y_1, \ldots, y_m, y_{m+1})$ given $G(y_1, \ldots, y_m)$, for constants a and b (independent of l and m).

Intuitively, G is *efficiently incremental* if the amount of work required to recompute a group hash value a is "linearly proportional" to the size of the additional data element y_{m+1}. For Protocol GH1, recomputation of the entire hash is required, and is hence not efficiently incremental. Note that given $a = h(y_1, \ldots, y_m)$, incrementing with y_{m+1} may require computing $a' = h(y_1, \ldots, y_m, y_{m+1})$, i.e., computation of the hash over an input size that is $m - 1$ times larger than y_{m+1}.[1] For Protocol GH2, a hash computation proportional in size to the new element is computed. In other words, $a = h(\cdots h(h(y_1, y_2), y_3), \ldots, y_m)$ can be incremented with y_{m+1} by computing $a' = h(a, y_{m+1})$. Hence, Protocol GH2 is efficiently incremental. Protocol GH3 is not efficiently incremental since $O(\lg m)$ hash computations are required to recompute a (the root of the tree). Given the group hash value a, Protocol GH4 requires computing $a' = a^{y_{m+1}} \bmod n$ and is therefore efficiently incremental. Although Protocol GH5, requires only a single application of G, it is applied to data that is extended to an $l - bit$ value after submission of the original hash by the user. This $l - bit$ value is dependent on an upper bound on the number m of submissions so that Protocol GH5 is not efficiently incremental. These results are summarized in Table 3.4.

[1]For hash functions which use an iterated compression function and do not appending padding (see Menezes *et al.* [MvOV97, Section 9.3]), the computation of a' *will* be efficiently incremental.

Regarding the incrementality of group hash techniques, an important concern relates to the requirement for the owners of the initial $y_1, \ldots, y_m$ to update their $member_{y_i}$. For each of the group hash protocols except Protocol GH5 (in which there is no $member_{y_i}$ for users), updates are required. This requirement necessitates additional communication with users (likely performed by a central entity) to reflect the change to a.

Definition 3.3 A group hashing function G is *efficiently decremental* if for an l-bit element $y_i \in \mathcal{Y}$, where $f(l)$ time is required to compute $G(y_i)$ for some function f, then at most $af(l) + b$ time is required to compute $G(y_1, \ldots, y_{i-1}, y_{i+1}, \ldots, y_m)$ given $G(y_1, \ldots, y_m)$, for constants a and b (independent of l and m).

Intuitively, G is efficiently decremental if the amount of work required to recompute the group hash value a is "linearly proportional" to the size of the removed data element y_i. Decrements differ slightly from increments in that deletions from $\{y_1, \ldots, y_m\}$ are not restricted to occur only at the end, i.e., any y_i may be removed. Given the removal of y_i, notice that the recomputation of the function $G(y_1, \ldots, y_{i-1}, y_{i+1}, \ldots, y_m)$ given $\{y_1, \ldots, y_{i-1}, y_{i+1}, \ldots, y_m\}$ is not an option unless the input data are available. However, their availability would defeat the point of using of a compressed representation.

Protocols GH1 and GH3 are not efficiently decremental for the same reason that they are not efficiently incremental. Protocol GH2 is not efficiently decremental since it requires knowledge of the inverse of $h()$ for even the efficient removal of y_m. For Protocol GH4, although it is efficiently incremental, it is not efficiently decremental since this would require knowledge of the inverse of the element to be removed; knowledge of this fact would allow one to factor the modulus. Finally, Protocol GH5 is not efficiently decremental since for any 1's in the element y_i to be removed, knowledge of whether any other y_j, $j \neq i$, has a 1 in that particular position is required for the authenticator to be properly updated. The requirements for changes to $member_{y_i}$ are the same as for the addition of a new element. These results are summarized in Table 3.4.

Depending on the intended application, Protocols GH2 and GH4 are superior with

	Additions		Deletions	
Hash Protocol	eff. incr.?	static $member_{y_i}$?	eff. decr.?	static $member_{y_i}$?
GH1: Cumulative	no	no	no	no
GH2: Recursive	yes	no	no	no
GH3: Tree	no	no	no	no
GH4: Exponentiation	yes	no	no	no
GH5: Bit	no	yes	no	yes

Table 3.4: Comparison of Updates for Group Hashing Techniques. 'eff. incr.?' is yes if the protocol is efficiently incremental (see Definition 3.2). Likewise for 'eff. decr.?' (see Definition 3.3). 'static $member_{y_i}$' is no if updates are required to $member_{y_i}$ in response to an addition or deletion of a data element from the group hash computation.

regard to the amount of time required to increment a group hash value given the appendage of an element to the original hash input. However, notice that Protocol GH1 may also be efficiently incremental, depending on the particular group hash function used. On the other hand, though not efficiently incremental, Protocol GH5 does not require the updating of $member_{y_i}$ subsequent to either the addition or removal of an element from the original hash input. Protocol GH5 would therefore be advantageous in situations where computation time is not a major concern, but communication time is.

3.3 Critique of Absolute Time Stamping

In this section, we critique the absolute time stamping protocols reviewed in Section 2.4. Protocol AB1 is a model of simplicity. Each user stores an amount of information proportional to the number of stamps submitted. No record-keeping is required by the time stamp authority (T). The communication involves only a single message pass by both the user and T. Document submission and stamp computation involve only a nominal number of applications of hash functions and digital signatures.

The main motivation for the remaining schemes is the requirement of *total trust* in T for Protocol AB1. Indeed, Protocol AB2 directly addresses this concern by building on top of Protocol AB1. The fundamental concern is that the issuance of a false time stamp can be devastating to a scheme. Suppose, for example, that for the provision of non-repudiation, users might have their signatures time stamped (as examined in Section 5.3). The ability to alter such a time (either in collusion with T or subsequent to a compromise of T's private signature key) allows one to backdate a signature applied with user u's private signature key, at a time when u's key may be compromised, to a time when u's key was not compromised. Hence, u is apparently made responsible for a message that he may not have signed. Such an example illustrates both the importance and fragility of the association between time and cryptography.

The schemes of Section 2.4.2 differ in that they do not require a trusted, centralized time stamping service to produce time stamps. Rather, they attempt to decentralize either the stamp computation or the storage of the resultant stamp. Protocols AB3 and AB4 distribute the storage of the resultant stamp. The former is impractical with regards to communication and storage, and because it requires each user to store each and every "submitted" stamp. This appears not to leave room for error in the case that stamps are incorrectly recorded for either malicious or unintentional reasons. Even further, this seems to imply a static membership since new users would not possess the stamps to documents from older submissions. This makes the verification (and potential adjudication) of a stamp's correctness a difficult, if not impossible task. As well, the amount of storage is excessive over the system as a whole since the same stamp will be stored in multiple locations.

In Protocol AB4, the delegation of authority is unclear. After all, who ensures the correct publication of the stamp? Considering that the authentication of this storage is important in case of disputes, it is critical that such a responsibility be delegated with clear goals in mind. As well, this scheme does not appear to allow for the option of verifying the signature applied by the submitter, i.e., is it left up to the publisher of the newspaper to confirm this? Though similar to Protocol AB1 in terms of the submission from the user (e.g. with respect to the time and space complexities), the "stamp" for the message appears to make the verification or adjudication of its validity difficult. After all, the source of and responsibility for maintaining the authenticity of the stamp is unclear. As well, the motivation and purpose for the inclusion of pub

	Centralized Protocols		Decentralized Protocols		
Properties	AB1	AB2	AB3	AB4	AB5
Certificate Obtained?	Y	Y	N	N	Y
Participation Req'd?	N/N	N/N	Y/Y	N/Y	Y/N
User Storage	1	$1 \cdot k$	$n \cdot 1$	1	$1 \cdot k$

Table 3.5: Comparison of Absolute Time Stamping Techniques. A certificate is obtained if the document submitter receives a signed response. The requirement of participation refers to the external participation of other user entities (i.e., users other than the submitter of the document for stamp production): (cooperation for Stamping Protocol?)/(cooperation for Verification Protocol?). User storage assumes that for n users, user u_i submits a document to be time stamped at time interval i. The storage computation consists of the multiplication (number of rounds in which a stamp is stored by u_i)$\cdot$(size of stamp in each round), where the size of a stamp from Protocol AB1 is defined as the unit size of a stamp and $k \leq n$.

is not well motivated. It is not clear what conclusions we can draw from its use. A document submitted at time t can easily associate an old pub value to it, for example. (The provision of an implicit absolute time is discussed in Section 3.3.1.)

Motivated by the same concern of trust in a centralized authority, Protocol AB5 distributes the stamp computation as opposed to its storage. The storage for each user is increased by a factor of k in comparison to Protocol AB1. In practice, this distribution will also increase the time required to obtain the stamp. Using k fellow users as opposed to one trusted center may introduce some problems. For example, what if some of the k users refuse to, or simply can't participate in the stamping of a message. It is unclear how this could be handled in a secure manner to allow for the proper results to be obtained from subsequent verification or adjudications. As well, the simple act of communicating with k other entities to receive a single time stamp is very costly.

Table 3.5 provides a summary of some of these concerns. The greatest distinction is that Protocol AB1 and Protocol AB2 makes use of a trusted, centralized entity for producing the time stamp, while stamps are constructed by the users themselves in the remaining schemes. Participation is required for Protocol AB3 in both the stamping and verification stages. In the former, each stamp submission is broadcast to all users, requiring their maintenance of its storage. In the latter, verification of the submission received by other users may be required in case of disputes. As mentioned earlier, it is not clear how disputes would be handled in Protocol AB4. Verification or a dispute regarding a particular stamp involves the participation of other users (e.g., users that possess copies of the widespread publication) though it is not clear whether these users are trusted entities or not. As well, although the question of cooperation for the stamping protocol is answered "No" here, there is the open question of who is *required* to maintain the eventual storage of the widespread publication. It appears though, that some user cooperation might be required. For Protocol AB5, it is unclear how the time of stamping should be verified since k, potentially different, times are included in a single time stamp.

With regards to the architectural complexity for the absolute schemes, each appears to require no more than the participating users themselves (i.e., potential time stamp requesters) and a single time stamp authority (if required at all). However, schemes in which the stamps are stored via a widespread publication may be susceptible to a large architectural complexity, depending on how much overhead is required for the publication, storage and verification of the stamps. The use of a widespread publication for the purpose of authenticating the stamps is discussed further in Section 4.2.1.

Overall, for each of the schemes highlighted in Table 3.5, Protocols AB3, AB4 and AB5 appear to be the least suitable for any practical implementation. Although Protocols AB4 and AB5 provide interesting, distributed alternatives for respectively storing and constructing time stamps, their protocol descriptions are ambiguous enough to cause concern with the consistency of future verifications of resultant time stamps.

3.3.1 On the Use of Implicit Time

Intuitively, an *implicit absolute time* is data from which a specific time can be uniquely and efficiently computed. More specifically, we have the following.

Definition 3.4 An *implicit absolute time* $impTime \in \mathcal{I}$ is the output of a function $Z : \mathcal{D} \times \mathcal{T} \rightarrow \mathcal{I}$, where $(d, t) \in \mathcal{D} \times \mathcal{T}$ is a data, time pair where d is uniquely associated with only one time t. The association between d and t is publically known, and trustworthy. ■

As an example, $d \in \mathcal{D}$ might represent the contents of a particular local newspaper at time t, where $impTime \in \mathcal{I}$ might represent the listing of the current weather for the day, as printed at time t. Implicit time was used (as opposed to an explicit time) in Protocols AB2 and AB4. More specifically, they used a particular type of implicit time known as unpredictable information. (Not all unpredictable information is an implicit time, e.g., consider the the use of hashes linking data together as recommended by Item 7 for Protocol AB2. Until this information is provided with an absolute time, it is unpredictable, but certainly does not allow a method for computing the explicit time from it.) *Unpredictable information* is data created at time t, having the property that it's entire contents could not have been predicted before time t. More specifically, we have the following.

Definition 3.5 *Unpredictable information* is an l-bit data $uI \in \mathcal{UI}$, created at time t, such that it is computationally infeasible to correctly determine or predict all l bits of uI prior to time t (the time at which all l bits are known).

The motivation for using unpredictable information is to prevent *forward dating*. For example, at time t, a document can be time stamped (e.g., by a malicious time stamper) with an explicit time $t' > t$ but not with unpredictable information (from which the time t' would be uniquely determined) since, by definition, this information can not be determined at time t.

However, there appears to be little advantage to using unpredictable information to prevent forward dating in this manner. Note that in Protocol AB4, if an explicit time were used as opposed to pub, it makes little sense to forward stamp the data with a time $t' > t$ where t is the current time. This attempt would be detected by anyone who verifies the posting of the stamp, and in the case of a newspaper, the date on the newspaper would alert future verifiers to the discrepancy with the date of the posted stamp.

3.4 Critique of Relative Time Stamping

The linking of data (resulting either from a single document submission or a group hashing) for the purpose of temporally ordering the data was reviewed in Section 2.5. In this section, we elaborate on some of the more relevant properties.

Additional communication is required (more-so for decentralized protocols) when compared with schemes in which only an absolute time is provided since the process of linking is not memoryless but dependent on result(s) from previous round(s), i.e., the authenticity of a stamp is measured "relative" to the stamps produced during other rounds, and hence requires access to previous stamps. This dependence has the effect of an increased interaction for either the production or verification of time stamps (e.g. recall the verification subprotocol of Protocol RL1) or necessitates some form of secondary, authentic storage (which requires an extra communication to obtain this value during stamp creation or verification).

Remark 3.1 *A relevant and practical concern with the application of relative temporal authentication is the notion of interoperability. Note that the temporal "links" produced within a particular group of users or by a particular central authority, are part of a closed system. Given a second group of users or time stamping authority, it may be difficult to obtain a relative measure among the time stamps produced within the different domains. See Definition 4.17 of Section 4.3.3 for further discussion.*

The contents of the linking elements for the protocols from Section 2.5 are shown in Table 3.6. Participation required during the validation of the temporal authenticity of the stamps is shown in the 'Validation' column of Table 3.6. The participation during validation is not dependent upon the particular linking element used, but rather, the protocol description and where the linking information is stored.

For Protocol RL1, verification of a time stamp requires an ability to access an indeterminate number of stamps produced before and/or after the stamp in question. This would require that the storage of all stamps be maintained by a particular user. This does not allow for the case when a user might not want to maintain the storage of a particular stamp any longer (in the case that there is no secondary storage),

Protocol	Linking Element L_r	Validation
RL1: Haber-Stornetta	$(t_{r-1}, ID_{r-1}, y_{r-1}, H(L_{r-1}))$	G
RL2: Recursive Hash	$h(a_{r-1}, L_{r-1})$	T
RL3: Extended Haber-Stornetta	$[(t_{r-k}, ID_{r-k}, y_{r-k}, H(L_{r-k})), \ldots, (t_{r-1}, ID_{r-1}, y_{r-1}, H(L_{r-1}))]$	G
RL4: Cumulative	$(a_1, a_2, \ldots, a_{r-1})$	U
RL5: Cumulative Hash	$h(a_1, a_2, \ldots, a_{r-1})$	T

Table 3.6: Linking Elements for Relative Time Stamping Protocols. 'Validation' indicates the participation required for the validation of a time stamp. 'G' refers to group validation in which users other than the stamp owner are required for validating the stamp's temporal authenticity. 'T' refers to validation in which participation from the trusted time stamp authority is required. 'U' refers to user validation in which the stamp owner has sufficient linking information to allow a self-contained verification temporal authenticity by others.

nor is it robust against the simple loss of a stamp. Protocols RL2 and Protocol RL5 deal with this problem by having a centralized authority maintain the storage of the stamps. Protocol RL3 (likewise Protocol RL5) extends the explicit reach of the linking element so that only 1 of k links are required to verify the authenticity of a stamp. Protocol RL4 places the storage of all stamps in possession of each stamp requester, thereby allowing stamp validation to be performed without the participation of any other users or trusted entities.

Distinctions can also be made with respect to the goal of the use of linking for each scheme. Protocols RL1 and RL3 use linking as a means for preventing the time stamp authority (T) from either backdating or forward dating time stamp requests. In Section 4.4, we present an attack to this provision which allows T to indeed backdate stamps for these particular protocols. Protocols RL2, RL4 and RL5 provided a relative ordering for the purpose of determining the position of two time stamps at some later time. These techniques are discussed further in subsections 4.3.3 and 4.2.1.

Chapter 4

A Framework for Temporal Authentication

In this chapter, we examine time stamping from the viewpoint of a time stamping protocol providing authentication. Just as a digital signature can provide message authentication and a key agreement protocol can provide key authentication, a time stamping protocol provides *temporal authentication.* A framework for the provision of temporal authentication is constructed from the time stamping protocols reviewed in Chapter 2. This framework provides precision for the informal definitions, concepts and protocols introduced and reviewed in Chapter 2. Generalizations are motivated with the presentation of two protocol failures and demonstrated by a hybrid time stamping protocol proposal.

Chapter Outline

In Section 4.1, we define a *time stamping scheme* (see Definition 4.1) and discuss the provision of *temporal authentication* with this scheme. Section 4.2 presents Protocol TS1, providing one alternative for implementing a time stamping scheme. Various options for authenticating the time stamp are also discussed. In Section 4.3, the provision of absolute, relative and hybrid temporal authentication are examined. In Section 4.4, the importance of the distinction between absolute and relative time stamps and the proper verification of their temporal authenticity is demonstrated by identifying protocol failures with the Benaloh-de Mare (Protocol GH4) and Haber-Stornetta (see Protocol RL1) protocols. In Section 4.5, we critique a hybrid time stamping protocol (see Protocol HY1) that most closely follows our framework (though it does so with solutions that are non-cryptographic). Subsequently, we modify this hybrid protocol giving Protocol HY2, which more closely follows our framework and addresses concerns raised regarding Protocol HY1.

4.1 Temporal Authentication

In this section, we introduce the notion of the *temporal authentication* of digital data as accomplished by time stamping the data. We begin by defining a *time stamping scheme*, which together with Protocol TS1 (see Section 4.2), add some precision to the stamping protocol of Definition 2.6.

Definition 4.1 The $seven-tuple$ $(\mathcal{M}, \mathcal{S}, \mathcal{G}, \mathcal{SS}, \mathcal{T}, \mathcal{K}, \mathcal{P})$ is a *time stamping scheme* $\mathcal{TS}$ where the following conditions are satisfied:

1. $\mathcal{M}$ is a finite set of possible *messages*;
2. $\mathcal{S}$ is a finite set of possible *time stamps*;
3. $\mathcal{G}$ is a group hash scheme (see Definition 3.1);
4. $\mathcal{SS}$ is a signature scheme (see Definition 2.11);
5. $\mathcal{T}$ is a finite set of times;
6. $\mathcal{K}$, the keyspace, is a finite set of possible keys;
7. For each $K \in \mathcal{K}$, corresponding to a *time stamp provider* $P \in \mathcal{P}$, for the *temporal authenticator (time stamping) function* $sig_P \in \mathcal{SA}$ (see Definition 2.11), where $sig_P : (\mathcal{M} \cup \mathcal{A}) \times \mathcal{T} \rightarrow \mathcal{S}$ (where $\mathcal{A}$ is the set of group hash values, see Definition 3.1) and *time stamp verification function* $ver_P \in \mathcal{VA}$ (see Definition 2.11) where $ver_P : \mathcal{S} \times \mathcal{T} \times (\mathcal{M} \cup \mathcal{A}) \rightarrow \{\text{true, false}\}$, the following equation

is satisfied for every message or group hash $z \in (\mathcal{M} \cup \mathcal{A})$, time $t \in \mathcal{T}$ and *time stamp* $s \in \mathcal{S}$:

$$ver_P(s, t, z) = \begin{cases} \text{true} & \text{if } s = sig_P(z, t) \\ \text{false} & \text{if } s \neq sig_P(z, t) \end{cases}$$

■

Definition 4.1 more clearly defines the authentication of a (cryptographic) time stamp, as informally defined with Definition 2.2. The authentication is provided by the signature of the time stamp provider. To aid in the presentation of how a time stamping scheme provides temporal authentication, we first review the notion of authenticity.

4.1.1 Authenticating Data

The term "authentication" is an overused and often abused term in cryptography. In this subsection, we provide an intuitive understanding of what it means to authenticate data by reviewing several requirements that would typically be used in a scheme with certificate-based digital signatures – a scheme providing message authentication.

Claiming that something is *authentic* implies that it is "fully trustworthy as according with fact" [Mer98]. The object in question is "actually and exactly what is claimed" and "not false or [an] imitation" [Mer98]. In cryptography, the objects that are purported to be authentic can include message or key data, a digital signature, or the identity of an individual. We introduce here the concept of *temporal authentication* which deals with the authentication of time data as provided by a time stamping scheme (see Definition 4.1).

One can think of the "authentication of data" as the legitimization of the data using a set of mathematical functions with corresponding requirements and assumptions with regard to both the functions and the provider of the authentication. The functions serve to specify the properties that a particular form of authentication provides. The requirements and assumptions allow a verifying party to measure their trust or confidence in the authentication of the data through a verification procedure. As an example, recall the message authentication of data m, as reviewed in Section 1.1.

The message authentication of m can be achieved by the production of a signature $c = sig_u(m)$, purportedly by the user u, as defined by the functions of Definition 2.11. The message authentication of m is intended to corroborate that the *source* of c is indeed u. This corroboration can be achieved by the binding of u's name to the verification key ver_u used to verify the mathematical correctness of the signature c. This binding is performed by a *trusted* certification authority (CA), where the subsequent verification of this certification (by signature verifying parties) may be achieved through the *a priori* possession of the CA's verification key. Therefore, for this example, trust is achieved by the delegation of the trustworthy certification of user's verification key to a trusted CA that provides verifiable certification of user's verification keys. For users that trust this particular CA and its practices, the message authenticity of signatures received from users certified by this CA, can hence verify the mathematical correctness of the signature and ensure that the public key used to perform the verification, is bound to u by a trusted CA. (A more complete signature verification procedure is given as Protocol DS1 in Section 5.3.2.)

Remark 4.1 *(Trust) The term trust can be very difficult to define, e.g., as for "trusted entities" or "trusted data." For our purposes, we assume that a trusted entity is an entity that honestly and correctly executes functions for which it is recognized as intended to execute. The entity honestly determines the correctness of any input data or any requesting entity with regard to the publically verifiable requirements corresponding to the function for which the trusted entity will be executing. Trusted data is data produced by a trusted entity(s).*

4.1.2 Temporal Authentication

Temporal authentication intuitively combines message authentication with the notion of timeliness of messages (see Definition 4.2). The term *temporal* refers to something "of or relating to the sequence of time or to a particular time" [Mer98]. The temporal authentication of the message y ensures that in addition to producing an authentic representation for y, this representation is ordered amongst all other temporally authenticated data (i.e., "[related] to the sequence of time") and/or associated with a specific time (i.e., "[related] to a particular time"). We refer to the former representation as *relative temporal authentication* (see Definition 4.13) and the latter as *absolute temporal authentication* (see Definition 4.8).

A temporal association ensures that the resulting representation for y is both authentic and timely. The temporal authentication of y can be accomplished by the production of a time stamp $s \in \mathcal{S}$ by a trusted time stamp provider, which is a function of both the message y (which itself may be the function of some document) and a time $t \in \mathcal{T}$ from which the temporal position of y can later be inferred. This can be accomplished with the time stamping scheme of Definition 4.1. The time stamp is *temporally authentic* if it

1. is verifiably produced by a trusted provider(s) and

2. includes a trustworthy time.

The provision of message authentication (as described above) relied on a trustworthy certification by a CA. The same is true for the property of temporal authentication.

Definition 4.2 The *temporal authentication* of data y provides corroborative evidence regarding a time of existence of y in the form of a *time stamp* s and can be achieved by a time stamping scheme (see Definition 4.1) using a *trusted* time stamp provider (authority) T.

The role of a time stamp provider(s) is examined further in Section 4.2.1. The provision of time is discussed further in Section 4.3.

Temporal authentication provides an assurance of the temporal ordering (induced by the time stamp construction) of two messages. The ordering of these stamps is a partial ordering. If any two stamps are comparable, the set of stamps form a total order or chain. We refer to this set as a *temporal chain (or temporal order)* since elements of the set are comparable based on their temporal interpretation. Note that we can discuss orderings of stamps versus the ordering of a document/stamp pair. This distinction is relevant in cases where several documents are input to produce a single time stamp, i.e., group hashing as reviewed in Section 2.3. Although the resultant stamps from each round form a total order, it is not necessarily the case that the document-stamp pairs within a given round can be ordered since group hashing does not necessarily provide an ordering of the data.

4.2 Providing Temporal Authentication

The process of temporally authenticating data can be achieved with a time stamping scheme (see Definition 4.1). Each instance or round of a larger time stamping process implements this scheme as a *time stamping protocol*. The time stamping process is illustrated in Figure 4.1. One possible implementation of the time stamping scheme as a time stamping protocol is presented as Protocol TS1. Throughout, we make use of the following definitions. Unless otherwise noted, the 'event' associated with time t_i refers to the time of stamping.

Definition 4.3 Let '$\prec$' represent the 'earlier than' relation where $t_i \prec t_j$ if the event associated with the time t_i occurred earlier than the event associated with t_j. Let '$\preceq$' be the 'earlier than' relation in which $t_i = t_j$ may be true.

Definition 4.4 Let '$\succ$' represent the 'later than' relation where $t_i \succ t_j$ if the event associated with the time t_i occurred later than the event associated with t_j. Let '$\succeq$' be the 'later than' relation in which $t_i = t_j$ may be true.

This ordering of the times is used to define a time stamp process as follows.

Definition 4.5 A *time stamping process* is a set of functions $\{f_1, f_2, \ldots\}$ defined by

$$f_i(z) = sig_P(z, t_i)$$

such that $\forall i \geq 1$, $t_{i+1} \succ t_i$, where z, t_i and sig_P are defined in Definition 4.1. A trusted time stamp provider executes f_i 'earlier than' f_{i+1} for all $i \geq 1$. (Further requirements regarding the application of time are discussed in Section 4.3.) ■

An important assumption with regard to Protocol TS1 is that the time stamp provider is trusted to honestly produce time stamps with a correct time and maintains

Round 1.

1. User u_1 submits data y_1 for stamping to T.
2. T returns the time t_1 and time stamp s_1 to u_1.

Round 2.

1. User u_2 submits data y_2 for stamping to T.
2. T returns the time t_2 and time stamp s_2 to u_2.

$\vdots \qquad\qquad\qquad \vdots \qquad\qquad\qquad \vdots$

Figure 4.1: Global View of a Time Stamping Process. Each round identifies an instance of this process as might be performed by the general time stamping Protocol TS1.

reasonable protection of the signing key used for the production of time stamps. Alternatively though, the function of the time stamp provider can be distributed using either threshold or proactive signatures (see brief discussion in Section 6.2) so that a number of time stamp providers contribute to the production of a single stamp and compromise of a single provider's signing key does not allow the production of forged time stamps. Alternative options in the case that corroborative evidence is required, in addition to the temporal authentication provided by a single time stamp provider, are discussed in Section 4.2.1.

Figure 4.2 gives a conceptual representation of the functions used in the time stamping Protocol TS1. The length of a round (in which more than one document is submitted to be time stamped) may be determined either by fixing an upper bound of the number of messages that will be jointly stamped in the round or on the amount of time that is allowed to elapse before a stamp is output. A group hash is used in the case that more than one document is input during a particular round. In any event, only one time stamp is output for a particular round. The provision of time for the time stamp is discussed in Section 4.3. The authentication and storage of the stamp are discussed in Section 4.2.1.

Protocol TS1 General Time Stamp Protocol.

Description: This protocol gives the abstract steps required for the production and verification of a time stamp s from Definition 4.1.

Note: Let P represent the time stamp provider. Each potential verifying party has a copy of P's verfication key *a priori*.

Time Stamp Production

Input: data y or set of data $(y_1, \ldots, y_m)$

Output: time stamp capsule $cap_y = (y, t, s, a, member_y)$

```
1: User $u$ submits the data $y$ to $P$ for time stamping.
2: If a group hashing scheme $\mathcal{G}$ (see Definition 3.1) is being used, then $P$ gathers $m$
   such input, $(y_1, \ldots, y_m)$ and computes the group hash value $a = G(y_1, \ldots, y_m)$.
3: $P$ obtains the time $t$ (either an absolute, relative or hybrid time) as specified in
   Section 4.3.
4: $P$ computes the time stamp $s = sig_P(z, t)$ where
```

$$z = \begin{cases} a & \text{if group hashing is used} \\ y & \text{otherwise} \end{cases} \tag{4.1}$$

```
5: $P$ returns the time stamp capsule $cap_y = (y, t, s, a, member_y)$ to $u$, where $a$ and
   $member_y$ are included only if group hashing is used.
```

Time Stamp Verification

Input: time stamp capsule $cap_y = (y, t, s, a, member_y)$

Output: whether s is a valid time stamp for data y at time t

```
1: Verifier $v$ obtains the time stamp capsule $cap_y$, e.g., user $u$ is the verifier of the
   capsule upon receipt from $P$.
2: If $a$ and $member_y$ are included in the time stamp capsule (so that group hashing
   was purportedly used), then $v$ computes $V(a, y, member_y)$ (see Definition 3.1)
   and continues to step 3 if successful and aborts with 'false' output otherwise.
3: Let $z$ be defined as in (4.1). $v$ determines the truth value of $ver_P(s, t, z)$ by
   determining the mathematical correctness of the signature $s$ using an a priori
   stored copy of $P$'s signature verification key.
```

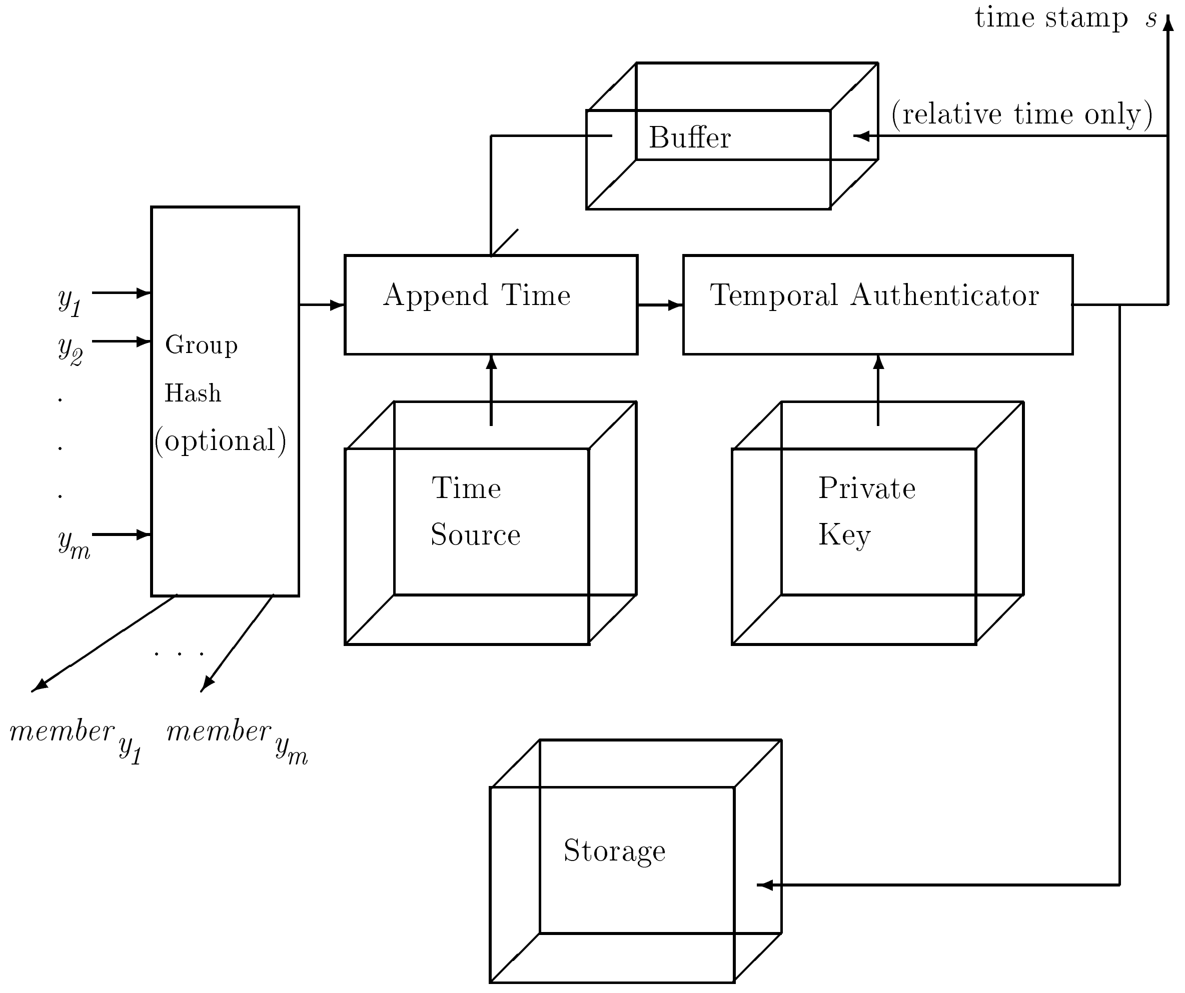

Figure 4.2: Generic structure of a time stamping protocol. Each $y_i = h(x_i)$ is the hash of a document x_i, input to receive a time stamp. Rectangles indicate functions that are performed (typically by the time stamp producer) and corresponed to those described in Sections 4.2 and 4.3. Cubes indicate sources of trusted or authenticated storage. The role of the Buffer and the Time Source are described in Sections 4.3.2, 4.3.3 and 4.3.4. The Temporal Authenticator and Storage are described in Section 4.2.1.

The verification of the stamp assumes trust in the time stamp provider for honestly producing time stamps. However, not all stamp verifications will be successful. We identify a *false stamp* as one in which an unsuccessful verification results.

Definition 4.6 A *false stamp* is a time stamp for which the verification of the stamp's temporal authenticity has failed. A stamp s for a document y is a *valid stamp* if s is not a false stamp.

Notice that this definition of a false stamp differs from a fraudulent stamp produced by a dishonest time stamp provider, e.g., if the provider includes an improper time.

4.2.1 Alternatives for Authenticating the Stamp

In situations where long-term trust in a single time stamp provider (authority) T is not desireable, additional corroborative evidence may be provided for the time stamp s. In this subsection, we examine several options or enhancements to Protocol TS1 for authenticating or legitimizing the resultant time stamp, with particular emphasis on techniques that corroborate a time stamp that would be provided by a single time stamp authority.

Message Authentication

Several uses of a digital signature were provided by the stamping protocols reviewed in Chapter 2, including the following:

1. the digital signature of a trusted authority, e.g., Protocol AB1;
2. a decentralized protocol requiring the digital signatures of other entities, e.g., Protocol AB5.

The digital signature of a trusted authority can consist of a centralized protocol or a distributed version, e.g., a threshold or proactive scheme (see brief description in Section 6.2). Protocol AB1 is an example of a centralized protocol that requires complete trust in the provision of temporal authentication by the single time stamp authority. On the other hand, Protocol AB2 additionally provides corroborative evidence by the inclusion of independent time stamps from a number of so-called temporal data authorities. The provision of a time stamp in which several varying times are included, as in Protocol AB2 and Protocol AB5 has the effect of weakening the granularity of time provided. In other words, given that for k absolute times provided in a time stamp, where $t_1 \preceq \ldots \preceq t_k$, one may only be able to determine that a data item was time stamped after time t_1 and before time t_k. A large gap between these two times can reduce a scheme's practicality.

Alternatively, there may be no message authentication provided for the $(data, time)$ pair. For example, in Protocol AB4, although the submitting user signs the $(data, time)$ pair, no real temporal authentication is provided since a lone user should not be trusted for providing a valid time. In Protocol AB4, subsequent storage techniques (see below) are used to provide authentication for the data.

Storage

Storage of a time stamp (as well as any other information required for the verification of the stamp) refers to the maintenance of its existence and integrity for the purpose of future verification(s). The storage of the stamp serves many purposes, including the following.

1. *Functionality.* The storage of the time stamp, if stored only by the time stamp submitter, is required for subsequent verification to be performed at all.

2. *Redundancy.* The storage of the time stamp may also be used as alternative means for demonstrating the existence of the time stamp. It can be used as corroborative evidence in the case of a dispute regarding the status of a user's version. In this sense, its mere existence at a secondary storage facility reduces the trust required in a central time stamp authority in the case that a temporal ordering is provided by the storage (see next point). See Protocol HY2 (of Section 4.5) for a scheme that uses a storage facility for redundant storage of time stamps.

3. *Relative Ordering.* A centralized storage of time stamps may provide for an incidental relative ordering (see Section 4.3.3) of the stamps in the case that

newly received stamps are appended to the end of storage upon receipt (and as well, are received in the same order that they are time stamped).

The integrity of the storage is required for each of the points indicated above. The provision of this integrity depends on the how the stamp is stored. This provision can affect one's trust in the time stamp and can also affect the efficiency of the stamp's verification. Alternatives for storage of the stamp include:

1. *Storage by Owner.* In this case, only the submitting user maintains a copy of the resultant time stamp, e.g., Protocol AB1.

2. *Centralized Storage.* Centralized storage may involve a storage facility maintained by a trusted entity (possibly but not necessarily the time stamp authority). Redundancy may be provided by distributing the storage among several trusted entities. See Protocol HY2 (of Section 4.5) for an example usage of centralized storage.

3. *Decentralized Storage.* The decentralized storage of time stamps, involves the distribution of the time stamps among users. Protocol AB3 distributed the storage of the time stamp among users. Protocol RL1 distributed information allowing the verification of time stamps among users. Additional cooperation may be required here for the verification of the time stamp since users not directly responsible for the production of the time stamp may need to be contacted.

4. *Widespread Publication.* A widespread publication involves a large scale distribution of the stamp. If performed only periodically (so that the information from the widespread publication alone is not sufficient to validate the stamp), alternative means (as described above) must be used for storage of necessary information. See Protocol HY1 (of Section 4.5) for an example usage of a widespread publication.

The most interesting option for "authenticating" storage involves the use of a widespread publication of the information. Originally suggested by Merkle [Mer80] (often referred to as a "Merkle channel") for the authentication of public keys and later by Bayer, Haber and Stornetta [BHS93] for the authentication of time stamps, a widespread publication can be thought of as a decentralized storage. For example, recall Protocol AB4 where time stamps are published daily in the local newspaper. More correctly, suppose that the submissions of users were published. The publication date accompanying the submissions provides an absolute time stamp. Authentication of the submissions is provided by the fact that the widespread dissemination of the information has bound their submission with the associated time of publication. Linking can be combined with group hashing so that resultant information need only be published periodically. This technique is used by Protocol HY1 in Section 4.5 where some additional concerns with a widespread publication are presented.

4.3 Associating a Time with Data

Section 2.1.1 introduced the protocols and functions used to provide a (cryptographic) time stamp s for data y by the authentic association of a 'time' t with y. Three varieties of 'time' were also introduced. A typical purpose for obtaining a time stamp is to allow a determination of 'when' y existed as compared to some other 'time' (possibly also contained within another time stamp); a so-called *temporal measurement* (see Definition 2.9). In this section, we present several requirements regarding the production of a time for a time stamp, and examine how each type of time provides these requirements.

4.3.1 Applying a Consistent Time

The production of a time stamp s is an instance of a larger, ongoing time stamping process (see Definition 4.5). Each such instance is referred to as a *round*, and the provision of the time stamp during each round is performed by a *time stamping protocol* (see Definition 2.6). To allow for meaningful temporal measurements,[1] each assignment of a time by a time stamping protocol should be consistent for all

[1]A temporal measurement is a comparison between two times, the result of which determines which time (or more specifically, the event associated with the particular time) was earlier.

protocol instances. In this subsection, we present several requirements regarding the application of a time in a time stamping protocol, that allow for such meaningful measurements.

To motivate and clarify this consistency requirement, consider the production of the time stamps $s_1 = sig_T(y_1, t_1)$ and $s_2 = sig_T(y_2, t_2)$ for respective data y_1 and y_2 with times t_1 and t_2, using Protocol AB1 of Section 2.4.1. A natural question might be: "Which of y_1 or y_2 were time stamped first?" However, this question is relevant only if t_1 and t_2 indicate the times at which y_1 and y_2 were time stamped. In other words, suppose that the time stamp authority T time stamps data with the 'time of receipt' of the data. Therefore, t_1 and t_2 would represent the respective times at which the data y_1 and y_2 were received by T. Suppose, without loss of generality, that y_1 was received prior to y_2 so that $t_1 \prec t_2$. Yet suppose that y_1 happened to be time stamped by T later than y_2, even though it was received earlier. Therefore, there would be an inconsistency with the answer provided for the question above, and the times associated with the data since although the time associated with y_1 is earlier than the time associated with y_2, y_1 as stamped later than y_2.

Consider also the following example where the stamps s_1 and s_2 are defined as above. Suppose that y_1 happened to be *received* at time t_1, while y_2 happened to be *stamped* at time t_2, where as above, $t_1 \prec t_2$. Beyond being inconsistent with the provision of times in the time stamps, the answer to the question "Which of y_1 or y_2 were time stamped first?" would be indeterminable since the time of stamping of y_1 is not known; t_1 indicates only the time of receipt of y_1.

The lack of precision regarding the application of a time by T in the above two examples was purposeful. Its intent was to illustrate the requirement for precision so that time stamps produced by T should be applied with times that obey a consistent and unambiguous rule for all rounds in which a time stamp is produced. A variety of such rules, regarding the 'meaning' associated with the time of stamping include the following:

1. each input to be time stamped is assigned a time upon its receipt, to be included in the resultant time stamp by the time stamp provider, or

1. The time applied to each data or group of data should be consistent and its meaning unambiguous for all rounds in which a time stamp is produced.
2. The time applied to each data or group of data should be monotonically increasing with the number of data input to be time stamped (see Definition 4.5).

Table 4.1: Requirements for the Association of a Time in a Time Stamping Protocol.

2. each input received is ordered in a queue where each item is assigned a time, to be included in the resultant time stamp, upon its removal from the queue.

A time stamp provider must ensure that a single, unambiguous, consistent rule is followed for each time stamp production. Additionally, the provision of time through a time stamp must allow for subsequent determination of the order in which the time stamps were provided. In other words, if data y_i was submitted no later than data y_j, then for the respective times t_i and t_j associated with the data through the respective time stamps s_i and s_j, then one must be able to determine that $t_i \preceq t_j$. The requirements for the association of a time in a time stamping protocol are summarized in Table 4.1.

4.3.2 Providing Absolute Time

Definition 2.3 provides a definition of an absolute time stamp as the result of the authentic association of an absolute, universal time with data. In this subsection, we more precisely define an absolute time, and examine how such a time is provided for a time stamp.

An absolute time is more intuitive than a relative time (see Section 4.3.3 below). It is the de facto form of time that is recognizable by many people on this earth. For example, the time t where

$$t = \text{Mon Aug 28 15:43:32 EDT 1998}$$

is an example of an absolute time. Indicated are the day of the week:'Mon'; the month of the year:'Aug'; the date within that month:'28'; the time of day:'15:43:32';

the current ‘time zone’:‘EDT’; and the year:‘1998’. This time has meaning relative to other times. For example, t is one hour later than the absolute time $t^* =$ Mon Aug 28 14:43:32 EDT 1998, identifying the time, and any events associated with t as ‘later than’ events associated with t^*, i.e., $t \succ t^*$.

Although there is no (known) truly “absolute” time, for digital applications, one can be constructed. For example, this time might be represented, similar to the above representation, by the current year (number of times the earth has revolved around the moon since the birth of Christ), the month within that year and day within that month (following the Julian calendar) and the time of day (specified by Universal Coordinated Time (UCT)). However, there can be alternative definitions [DS93, Section 3]. A consistent, clear definition of how the time is provided and format for describing this time is required in any case [DS93]. Throughout, we assume that a standarized time value is recognized by all time stamp authorities. We refer to this time as a *universal time*.

Definition 4.7 An *absolute time* is a time t from which a universal time t' can be uniquely and efficiently determined. The absolute time t is *explicit* if t' can be computed from t using only arithmetic operations, while t is *implicit* if additionally, external time data is required to compute t' from t.

Examples of an explicit absolute time include the time $t =$ Mon Aug 28 15:43:32 EDT 1998 (which is also one possibility for a universal time) as shown above or alternatively, the time t_1 which represents the number of seconds that have elapsed since a certain some ‘base’ time, i.e., $t_1 = 1$ with base time of t, would represent the universal time $t' =$ Mon Aug 28 15:43:33 EDT 1998. Examples of an implicit time were given for Protocol AB2 in Section 2.4.1.

An absolute time stamping protocol is *memoryless* since the time stamp construction computations during one round are not dependent on any function of the time stamps from previous rounds. Rather, an absolute time stamp requires a source for its time independent of occasions of time stamp production. This source of the absolute time can be a clock that is either internal or external to the time stamp provider. An internal clock is maintained by the time stamp provider, i.e., its integrity is maintained locally by T. More than one internal clock may be used whereby some function of the times (e.g., their average so long as the difference between their times is not too great) is used in the time stamping operation.

Remark 4.2 *(Similarities between cryptographic keys and clocks.) A clock is similar to a cryptographic key whose authenticity must be maintained, but not its privacy (e.g., a public verification key). Both are susceptible to attacks in which the current state or value is modified, e.g., by forwarding or reversing the time on the clock or altering the value of the key. Synonymous to a key that is renewed to limit the exposure subsequent to an attack, clocks may be periodically synchronized, to limit the effects of "clock skew" but as well to allow the detection of potential, unauthorized alterations of the time (where an unauthorized alteration would involve, for example, the modification of the time).*

Definition 4.8 A time stamping protocol that provides temporal authentication with an absolute time provides *absolute temporal authentication.*

Measuring Absolute Time

Definition 4.9 An *absolute temporal measurement* is a temporal measurement (see Definition 2.9) for which the order of absolute times t and t' is determined. In other words, whether $t \prec t'$, $t = t'$ or $t \succ t'$. ■

Unlike a relative measurement, the times in an absolute measurement need not be cryptographically associated with a time stamp. Since absolute times are so ubiquitous, other times may be used in an absolute measurement, including:

1. the current time, i.e., time when the comparison is being made;
2. the absolute time(s) from some other stamp(s); or
3. some other event or object for which an absolute time is supplied, e.g., the deadline for submission of a conference paper might be compared with the time stamp on the particular version of that paper.

Given a rule for a particular absolute time stamping protocol, stating the time, relative to the time of submission of a document, should be associated with a given submission, we define the malicious back or forward stamping of a document as follows.

Definition 4.10 A document y has been *absolutely back stamped* if an absolute temporal measure of the time stamp s for y infers that the time t associated with y is earlier than the time expected, based on the rule of the time stamping protocol.

Definition 4.11 A document y has been *absolutely forward stamped* if a temporal measure of the time stamp for y infers that the time associated with y is greater than the time expected, based on the rule of the time stamping protocol.

For example, suppose that the rule for a particular time stamping protocol states that data are time stamped with the time indicating their time of receipt by the time stamp provider. Suppose that data y was received at time t. y would be absolutely back stamped if a stamp s was constructed for y, using a time $t' \prec t$. Similarly, if a time $t'' \succ t$ were used as the time of stamping, then y would be absolutely forward stamped. Such malicious action by a rogue T is difficult to prevent. However, some possibilities for limiting the extent of a back or forward stamp are examined in Section 5.1.3.

4.3.3 Providing Relative Time

Definition 2.4 provides a definition of a relative time stamp as the result of the authentic association of a relative time with data. In this subsection, we define a relative time more precisely, and examine how such a time is provided for a time stamp.

Definition 4.12 A *relative time* is a value r_i such that given at least one other relative time value r_j, it can be determined which of the following is true (see Definitions 4.3 and 4.4):

1. $r_i \prec r_j$,

2. $r_i = r_j$, or

3. $r_i \succ r_j$.

■

Notice that an absolute time is also a relative time since given two absolute times t_i and t_j, one can determine which of the above conditions is true. The key distinction is that a relative time need not allow one to compute a unique universal time from it.

Unlike an absolute time (where the time increases independent of data submitted to be time stamped), the provision of a relative time is dependent upon the stamping of previous data. Therefore, notice that from Definition 4.12, a single relative time may have no temporal meaning on its own.

Definition 4.13 A time stamping protocol that provides temporal authentication with relative time provides *relative temporal authentication.*

Remark 4.3 *(Different uses for a relative time.) A relative can serve many functions. For the provision of relative temporal authentication, two such purposes of a relative time have been used.*

1. *(Relative Measure.) In protocols such as Protocol RL2 (of Section 2.5) a relative time was used to allow the ordering of two time stamps to be determined.*

2. *(Corroborating Evidence.) In protocols such as Protocol HY1 (of Section 4.5), an ordering of time stamps is used to provide corrobating evidence for a time stamp by using a relative time (linking) with a periodic "authentication" of stamps. This corroborating evidence serves to reduce the amount of trust required in the principal time stamp provider.*

The "time" in a relative time stamp could be represented by a variety of values, including:

1. *Linking information.* Explicit dependency on one or more previous stamps.

2. *Counter.* A monotonically increasing positive integer.

Both linking information and a counter were used by all of the schemes reviewed in Section 2.5. In the following, we expand on the varieties of a relative time.

Linking Information

A relative time using linking information can be implemented using a buffer to store previous stamp information. This (variable or fixed sized) buffer can be initialized with a publically verifiable value. For Protocol RL1 a buffer of size 1 was used so that each stamp was authentically bound to only 1 previous stamp. Protocol RL3 used a buffer of size k so that each stamp was authentically bound to only k previous stamps. Denoting the current stamp as s_i, the buffer contains some subset S' of some function of the previous stamps $S = \{s_1, \ldots, s_{i-1}\}$. The appendage of a relative time from this buffer makes use of a subset $S'' \subseteq S'$.

Linking can be described in general as computation of the recurrence relation

$$l_r = link(prev, a_r)$$

given some initial value, where l_r represents the relative time for the current round, a_r represents a partial result of the current round (where a_r likely has no timeliness provisions) and $prev$ represents the time stamp result(s) from previous round(s). a_r might be a single document (or some function thereof) or the representative result of a group hashing of m documents. The goal of executing the $link()$ function is to relatively order data.

Remark 4.4 *The $link()$ function must be collision-resistant else one might be able to produce an alternative $prev'$ from a previous round in an attempt to maliciously demonstrate precedence over the current round, e.g., so that $link(prev, a_r) = link(prev', a_r)$.*

Counter

Alternatively or in conjunction with the linking of stamps, T may use a counter to order time stamps where the counter is a positive integer, initialized to zero and incremented by 1 for each time stamp produced. Because of the simplicity of maintaining a counter and the dependence on each stamping of a document, this time source is typically internal to T.

Incidental Relative Orderings with Counters Alternative to the explicit provision of relative time via linking or a counter, other stages in the production of a time stamp can incidently provide a relative time (meaning that their main purpose is not to provide a time, though such a relative measure, as described below, is recoverable).

For example, recall the group hashing techniques reviewed in Section 2.5. The schemes of Section 2.3.1 and Section 2.3.2 provide a relative ordering of the input data. Consider the computation $a_r = h(y_1, \ldots, y_m)$ of Protocol GH1. Given y_i and y_j, and their respective $member_{y_i}$ and $member_{y_j}$, the relative positioning of y_i and y_j can be determined during the recomputation of a_r. This relative ordering would typically be implemented similar to a counter whereby one would store $(i, y_i, member_{y_i})$ to facilitate proper recomputation of the group hash; i represents the counter value. Notice though that this relative ordering (provided by the group hashing), exists only for documents within the same round.

Protocols GH4 and GH5 do not provide such an *incidental ordering*. However, a relative ordering can certainly be explicitly provided during the production of the group hash. For example, in Protocol GH4, rather than broadcasting y_i, users might explicitly order their submissions (as required for the protocols from the previous paragraph) and input $h(i, y_i)$ to the group hash.

However, this precedence relationship should not be interpreted as an indication of which y_i has time precedence over another, unless this is explicitly indicated. As described, the group hash protocols gather submissions with no apparant regard to their "order" of submission. However, consider the situation in which documents are submitted to a central time stamp authority who after the reciept of m submissions, performs a group hash operation before stamping the result. By inputting the documents to the group hash in the order they were received, an incidental relative ordering is achieved.

Besides group hashing, an incidental ordering can also be provided during the storage of time stamps. For example, if stamps are appended to the current storage the last stamp appended would be the stamp with the latest "time" associated with it. As with the group hashing above, this ordering is based on a counter maintained within the file, pointing to the memory location(s) for each stamp.

Measuring Relative Time

Recovery of a relative time alone does not yield any information regarding "when" a relative time stamp was constructed. Rather, a relative time stamp contains a "time" which allows the *temporal position* of the stamp to be determined against other items for which a relative time is also provided.

Definition 4.14 A *relative temporal measurement* is a temporal measurement for which the order of relative times t and t' is determined . In other words, a determination of whether $t \prec t'$, $t = t'$ or $t \succ t'$. For linked times, one can determine for example that $t \succ t'$ by determining if for a set of intermediate times $(t_1, \ldots, t_k)$ with respective data $(a_1, \ldots, a_k)$ for stamps $(s_1, \ldots, s_k)$,

$$t = link(t_k, a_k), t_k = link(t_{k-1}, a_{k-1}), \ldots, t_1 = link(t', a_j).$$

For a counter based relative time, one can determine, for example, that $t \succ t'$ if $t - t' > 0$. ■

Definition 4.15 A document y has been *relatively back stamped* if a temporal measurement infers that y was stamped before y' when in fact, y was stamped after y'.

Definition 4.16 A document y has been *relatively forward stamped* if a temporal measurement infers that y was stamped after y' when in fact, y was stamped before y'.

Consider the unsuitability of these definitions for absolute time stamps (and hence the distinctions with Definitions 4.10 and 4.11). Suppose that a protocol provides a time stamp s for data y where $s = stamp(y)$. Suppose further that s is an absolute time stamp for which the time t, of submission of y, is associated with y. By Definitions 4.15 and 4.16, it would not be back stamping to assign a time $t' \prec t$ to the stamp s for y, so long as t' is greater than the time of the most recently issued stamp. However, this could cause great confusion. Consider the example of having conference submissions time stamped. Suppose that the deadline for submissions is Tuesday morning and that the latest submission so far was received on Monday morning. For the application of an absolute time stamp, the above definition allows a time from Monday afternoon to be applied to a paper that may be submitted after the Tuesday morning deadline.

Relative Markers. One problem with measuring the distance between linked stamps is that there may be a long chain between the times associated with s and s', requiring a potentially large number of computations to be performed for verification. As well, if the storage of the stamps is distributed (as in Protocol RL1) then a large amount of communication is required. Large chains can be dealt with by using relative markers.

Definition 4.17 *Relative markers* are distinguished stamps within a temporal chain that serve a special purpose. Two such markers are

1. *Intermediate Stamps.* Intermediate stamps are used by Pinto and Freitas [PF96] to shorten the chain of comparison between two stamps that contain many chain links between them.

2. *Cross-Stamps.* Cross-stamps which link two different time stamping "domains", allow for the relative comparison of two stamps that may have been produced by different time stamping authorities.

■

The use of *intermediate stamps* by Pinto and Freitas [PF96] was reviewed at the end of Section 2.5. Their usefulness can be motivated with the following example. Consider the verification of stamps a_i and a_j in Protocol RL2. Presuming that a_i was indeed produced prior to a_j, in step 2 of the verification of Protocol RL2, the verifier v obtains $a_{ij} = (a_{i+1}, a_{i+2}, \ldots, a_{j-1})$ from T and in step 3, computes

$$L'_{k+1} = h(a_k, L_k),\ i \leq k \leq (j-1)$$

and ensures that $L_{k+1} = L'_{k+1}$ for each value of k. This recursive operation (relative temporal measurement) demonstrates that a_i was used in the computation of a_j and

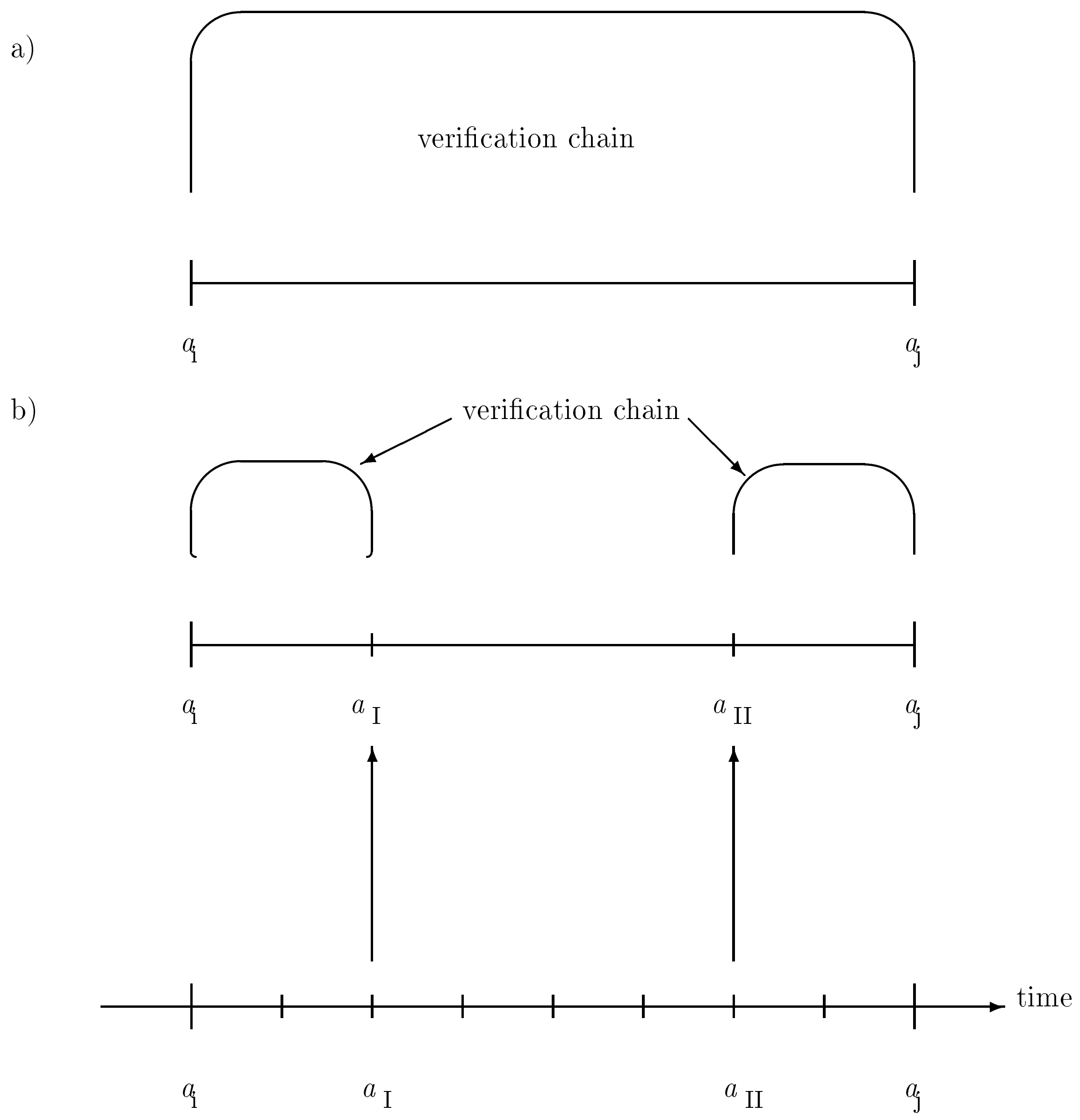

Figure 4.3: Improved Efficiency with Intermediate Stamps. a_i and a_j are time stamps (with respective times t_i and t_j) produced for users u_i and u_j respectively. a_I and a_{II} are trusted time stamps (with respective times t_I and t_{II}) produced solely by T. The temporal order $t_I \prec t_{II}$ is known and trusted to be true. (a) The entire sequence of stamps is required for determining that $t_i \prec t_j$. (b) Given that $t_I \prec t_{II}$, one need only show that $t_i \prec t_I$ and $t_{II} \prec t_j$. The length of the verifying chain is reduced by the number of stamps between a_I and a_{II}.

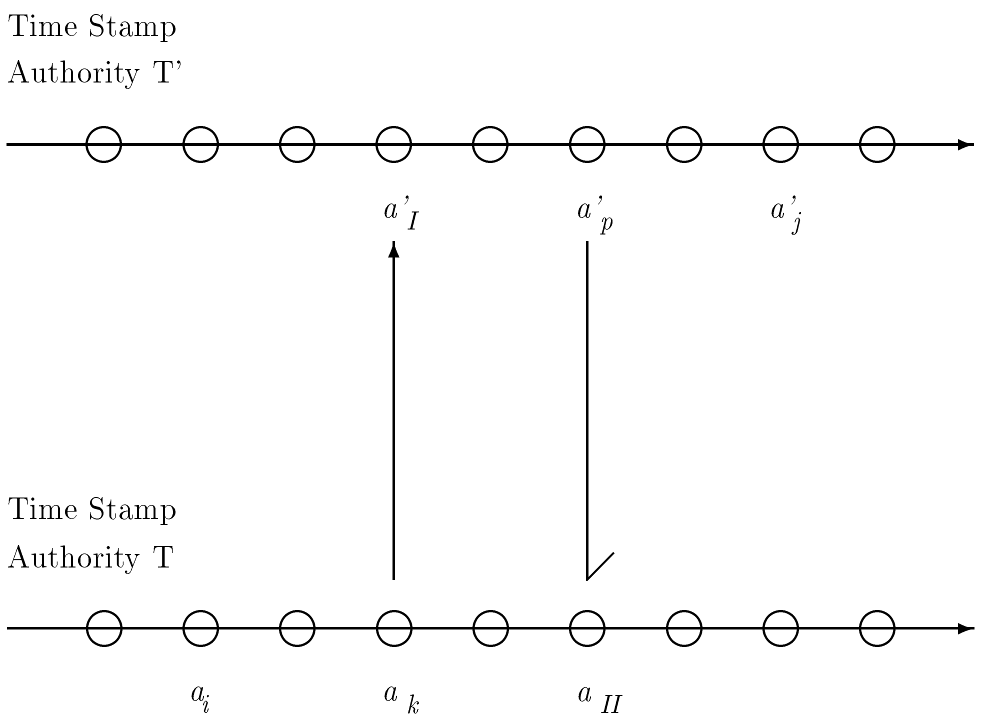

Figure 4.4: Cross-Stamps Allowing for Relative Stamp Interoperability. a_i and a'_j are user stamps created by respective time stamp authorities T and T'. a'_I is a cross-stamp produced by T' as a function of both the previous stamp in the upper chain as well as a_k. a_{II} is a cross-stamp produced by T as a function of both the previous stamp in the lower chain as well as a'_p.

hence, time stamped before a_j. However, it requires that $j - i$ stamps be obtained from T and at least $j - i$ hash computations be performed. As noted at the end of Section 2.5 and illustrated in Figure 4.3, intermediate stamps a_I and a_{II} allow the length of the verification process to be reduced by an amount proportional to the number of stamps between a_I and a_{II}. Intermediate stamps are also used by Haber and Stornetta (see Protocol HY1 of Section 4.5) and authenticated via a widespread publication in the New York Times.

Cross-stamps allow for interoperability between two temporal chains. Consider the user stamps a_i and a'_j from Figure 4.4 produced respectively by time stamp authorities T and T'. On their own, the temporal chains to which a_i and a'_j belong, have no comparable ordering. Therefore, without additional provisions, one would not be able to determine the positioning of stamps produced by different time stamp authorities. To overcome this problem, one can use cross-stamps as in Figure 4.4. a_k is a regular stamp produced for some user by T. T' obtains a_k from T and creates the cross stamp a'_I which connects the stamps prior to a_k on the lower chain to the stamps after a'_I on the upper chain. To complete the process so that the stamps prior to a'_I on the upper chain are comparable to stamps on the lower chain, T obtains a stamp from T' in a similar manner (in this case, T obtains a'_p to construct a_{II}).

4.3.4 Providing Hybrid Time

Definition 2.5 provides a definition of a hybrid time stamp as the result of the authentic association of both an absolute and relative time with data. In this subsection, we examine how such a time is provided within a time stamp. Protocols RL1 and RL3 were reviewed in Section 2.5 and Protocol HY1 is reviewed in Section 4.5. In addition to the provision of a relative temporal ordering, these schemes also provide for the recovery of an absolute time during stamp verification. Hence they provide a hybrid time stamp.

Definition 4.18 A time stamping protocol that provides temporal authentication with both an absolute and relative time provides *hybrid temporal authentication.*

The intent of using a hybrid time is to combine the advantages of both an absolute and relative time. However, it also incorporates some of the disadvantages. The choice between an absolute versus a relative time can depend on several factors. For example, the provision of an absolute time requires a trusted clock (see [LB92]) for the provider of the absolute time stamp and possibly for the verifiers of the absolute time contained within the stamp, should the correctness of the time stamp upon its initial receipt, need to be checked; also required by verifiers so as to allow a determination of what the current time is, during temporal measures of time stamps. On the other hand, the application of only relative times allows only for comparisons between like-stamped data, i.e., data for which a relative time stamp has also been provided. The choice between the more suitable time to provide will depend on the intended application for which the time stamps are being provided as well as the resources of the time stamp provider and verifiers.

Besides providing a recoverable temporal ordering of data, relative time or linking can also serve to reduce the trust required in the time stamp provider (see Section 4.2.1 and Section 4.3.3). Therefore, a natural hybrid scheme is one in which the linking is used as more of an integrity measure than a temporal measure (as in Protocols RL1, RL3 and upcoming with Protocol HY1).

A hybrid scheme can also provide both an absolute and hybrid stamp, whereas verification need only perform an absolute *or* relative measure. This may be practical in the case that a particular time stamp has both an absolute and relative time, yet the verifier, who does not have access to a trusted clock for validation of the absolute time, chooses not to determine the validity of the absolute time. In this case, the verifier must ensure that the time he has chosen to measure was provided for the purpose of temporally authenticating the data.

4.4 Importance of Proper Temporal Measurements

In this section, we present two protocol failures in schemes previously proposed in the literature. These failures occur as a result of misunderstanding the verification of a time stamp. These protocol failures were presented at the 1998 Symposium on Network and Distributed System Security [Jus98].

In Section 4.4.1 we discuss how the inclusion of an absolute time in the production of a time stamp for Protocol GH4 is not recoverable nor verifiable during stamp verification thereby not permitting a successful temporal measurement. Section 4.4.2 presents a collusion attack on Protocol RL1. We demonstrate how the use of only an absolute temporal measure (in a scheme that is intended to provide both absolute and relative temporal authentication) and unreasonable assumptions regarding trust in the time stamp authority, allows the relative backdating of documents.

The particular attack against Protocol RL1 is presented relative to the model in which the original scheme was given. The model assumes that the time stamping authority (T) need not be trusted. In Section 4.4.2, we show that a dishonest T can subvert the scheme unless certain precautions are taken. One should not presume however that such an attack is only successful against Protocol RL1. Indeed, one should be careful when designing similar linking schemes and specifically must consider the necessity of properly authenticating the resultant stamp (see Section 4.2.1). In environments where it is not unreasonable to trust T, such an assumption, and therefore the attack and precautions, may be unnecessary.

4.4.1 Protocol Failure: Inability to Measure an Absolute Time

In Section 2.3.3, Protocol GH4 was reviewed. In addition to the computation of a group hash, it was suggested that an absolute time may be included in the stamp construction, thereby allowing one to also provide absolute temporal authentication for the group hash.

The protocol allows the resultant time stamp for a round to be computed in such a manner that an on-line, centralized entity is not required for the stamp computation. We demonstrate here that the constructed time stamp does not allow for the recovery of an absolute time during its verification, even though such information is suggested for optional inclusion during stamp construction. In a sense, the time is "lost" during the stamp construction. More specifically, we state the following.

Proposition 4.1 *For Protocol GH4, one can neither recover nor validate (the absolute time) x during stamp verification.*

Proof:

Consider the verification of a time stamp by user v. User u_i would demonstrate that that the document y_i contributed to the round in question by giving $\{y_i, member_{y_i}\}$ to v. User v would compute $(member_{y_i})^{y_i} \bmod n$ and determine its equivalence to a_r modulo n. Although an absolute time may have been included in the stamp computation (i.e., by setting x to be the current date), no such absolute time is uniquely recoverable during time stamp verification. In other words, there is more than one candidate x such that $x^y \equiv a_r \bmod n$. Therefore, there is no reason for any recovered x to be trusted as the time of stamping of a_r. Hence, the inclusion of the current date as described in the stamp construction, serves no purpose since it is not recoverable during stamp verification. ■

One way to provide for the absolute temporal authentication of the stamp for Protocol GH4 is to authenticate a_r along with t (the time of stamping). The time t will be verified for its correctness rather than recovered from a_r. Use of a hash by computing $a'_r = h(a_r, t)$ and storing a'_r and t allows for increased storage efficiency. However, there may be additional overhead in case a decentralized protocol is used, for users to agree upon a time t. The provision of a relative temporal measure would allow users to demonstrate a time precedence ordering for documents submitted in distinct rounds. Techniques for providing absolute and relative times were respectively discussed in Sections 4.3.2 and 4.3.3. Authentication of storage of the stamps was discussed in Section 4.2.1.

4.4.2 Protocol Failure: An Improper Relative Measurement

In what follows, we demonstrate how the use of only an absolute temporal measure (in a scheme that is intended to provide hybrid temporal authentication) allows the relative back dating of documents in Protocol RL1. As well, some unreasonable assumptions with regard to the lack of requiring any trust in a central entity are also discussed.

Recall that Protocol RL1 is actually a hybrid scheme, and not just a relative stamping scheme since absolute times t_i are included in each time stamp, in addition to the linking of the stamps. Indeed, the verification process determines the position of only a single document rather than the relative positioning of a number of challenged documents. The linking of the resulting absolute time stamps is used as a means to prevent T from back or forward dating stamps. We therefore assume that if a_{i-1}, a_i and a_{i+1} are stamps that are consecutively linked in a temporal chain and the respective absolute times associated with each are t_{i-1}, t_i and t_{i+1}, then $t_{i-1} \prec t_i \prec t_{i+1}$. We make this assumption for the successful running of the protocol, i.e., we assume that any challenger that moves along the chain will check that the times will follow the same temporal order as the stamps to which they are associated.

Meaning of the Attack

A *fake chain attack* is recognized by Haber and Stornetta [HS91], where it is claimed that

> the only possible spoof is to prepare a fake chain of time-stamps, long enough to exhaust the most suspicious challenger that one anticipates.

Since each time stamp requires a signature by the time stamp authority (T), this attack would presumably require collaboration with T. This attack might appear not that difficult to implement except that for assigning fake stamps, a number of additional collaborators would be required. After all, a suspicious challenger might only be convinced of the legitimacy of a chain if a large number of distinct participants are contacted for verification.

In Lemma 4.1, a new attack is presented whereby one can collude with T and *partially insert* a single false stamp into a valid chain of stamps. In this way, only a small fake chain need be produced, that can be "fused" into the valid chain (though only one end of the fake chain is connected to the valid chain). This fake chain is the lower chain in Figure 4.5. The attack demonstrates that an untrusted, centralized T with no record-keeping is not sufficient for providing the claimed level of security. We can summarize the requirements and results of the attack as follows:

Attack Requirements

1. *Collusion with T.* The absolute backstamping (backdating) (cf. Definition 4.10) of a document requires the participation of the time stamp authority T. Protocol RL1 claimed that T need not perform any record-keeping, nor be trustworthy. The linking alone was claimed to prevent even T from backstamping a document.

2. *Additional collusion or advanced knowledge of attack.* The backstamping of a document requires either the participation of another user with a time stamp

produced near the time of the desired backstamping or the anticipation of a subsequent attack by the attacker having a previously time stamped document existing near the time of the desired backstamp.

3. *Subsequent maintenance by T.* As discussed below, the attack can require that T perform subsequent maintenance in order to "disguise" the existence of a backstamped document.

Attack Results and Limitations

1. *Absolute but not relative backstamping of a document.* As explained below, the attack does not allow one to relatively backstamp a document. This is not that constrictive since the verification procedure (see Protocol RL1) only performs an absolute temporal measure (see Section 4.3.2); the relative ordering is provided only to reduce the trust required in T and is not used in the recovery of a relative ordering during verification.

2. *Attack detection with enhanced verification.* The attack can be thwarted with a more vigorous verification protocol.

The absolute backstamping in Protocol RL1 is successful since the verification protocol performs a temporal comparison based only on the absolute times associated with the time stamps. However, it is not successful for relatively backstamping a document. In other words, T cannot backstamp to show the time precedence of one stamp over a previously, legitimately constructed stamp that would be verified by a relative measure. There is no relationship between stamps that are solely on the upper or lower chains here (see Figure 4.5 and Lemma 4.1). For example there does not exist a relative temporal relationship between the stamps a_{j+1} and a_i as they are not linked together (via a series of directed links).

Let $f : P \to P$ be a function where P is the set of all possible time stamp capsules. For example, from Figure 4.5, we have $p_1 \in P$ belonging to u_1, where $p_1 = \{C_1, L_1, a_1, ID_2\}$. Then $p_2 = f(p_1) = \{C_2, L_2, a_2, ID_3\}$. In other words, on input p_1, f produces the time stamp capsule for a time stamp whose linking information is

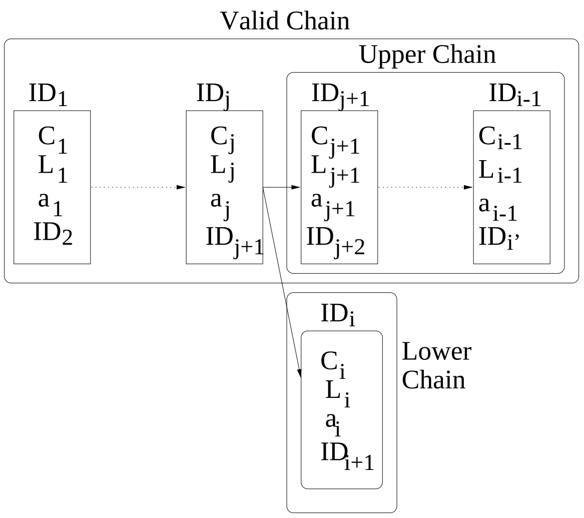

Figure 4.5: Multiple Chains in Protocol RL1. Each of the smaller rectangles represents the time stamp capsules for a user. The valid chain is an example of what might be produced from a normal running of Protocol RL1. The lower chain is produced by T in collusion with ID_i, in order to backstamp a document y_i. $ID_{i'}$ indicates that there is more than one possibility for the placement of the next stamp, depending on how the attack is mounted.

computed explicitly as a function of the stamp contained in p_1. As long as H (where H is used to compute L_2; see Protocol RL1) is collision resistant, the assumption is that f is a one-to-one function.

The claim of Haber and Stornetta [HS91] was that T need not be trusted since an attack would *require* finding a collision for H. From our attack, we can in fact state the following.

Lemma 4.1 *The collision-resistance of H is not sufficient so as to permit the truth of both statements in Protocol RL1:*

1. *T need not be trusted; and*
2. *T cannot back or forward stamp data.*

Proof:

For the proof, we demonstrate that T can indeed back stamp data for user u_i. The attack proceeds with user u_i colluding with T to backstamp a document y_i (with corresponding stamp a_i). (Readers may wish to re-read Protocol RL1 of Section 2.5 at this point to regain familiarity with the protocol.) Referring to Figure 4.5, we see how the resultant T-signed time stamp a_i is expected to appear immediately after a_{i-1}, i.e., normal running of the protocol assumes that additions take place at the end of the valid chain. However, as in the figure, a_i is placed by T, immediately after a_j. What advantage does this give u_i? Suppose that a_j, a_{j+1} and a_{i-1} contained the respective times t_j, t_{j+1} and t_{i-1} where $t_j \prec t_{j+1} \prec t_{i-1}$. If the stamp a_i (corresponding to document y_i) were placed in its correct place (i.e., after a_{i-1}), T would associate a time $t_i \succ t_{i-1}$ with it. By placing it immediately after a_j, T can assign any time t_i to y_i (in stamp a_i) such that $t_i \succ t_j$. Since $t_j \prec t_{i-1}$, T has absolutely backstamped y_i for u_i by assigning it a time earlier than the current time. Notice however that T has not relatively backstamped y_i here since a relative measurement does not show precedence of a_i (the time stamp for y_i) over any stamp in the upper chain.

Subsequent to the linking of y_i in this new chain, all future legitimate stamp requests can either be added by T in the lower chain (i.e., after y_i whereby $ID_{i'}$ in the upper chain could simply be assigned ID_i; see Figure 4.5) wherein only the lower chain would be continuing, or alternately added to first the lower then the upper chain (whereby $ID_{i'}$ in the upper chain would be assigned ID_{i+2}; see Figure 4.5) so that both chains are continuing. The latter technique ensures that challenges can proceed in the forward direction for documents contained in the upper chain (though how a challenger would even know when a chain is supposed to end when moving forward must be considered). However, it does require that the stamps for two documents will have the same round number associated with them (i.e., the same r) which may lead to a detection of the fraudulently produced stamp if the verification procedure were enhanced. This is under the assumption that consecutive stamps must have consecutive round numbers associated with them.

Now that T has produced a backstamped a_i (for submission y_i), suppose v were to verify the time stamp a_i following the procedure outlined in Protocol RL1. If v proceeds forward from a_i, no faults are discovered since documents are subsequently, legitimately stamped after a_i in the lower chain. However, if v proceeds backwards from a_i, note that the owner of a_j, namely u_j was previously given ID_{j+1} by T, However, v expects u_j to have been given ID_i. Hence, v would discover the possibility that something is wrong with a_i. Yet there are still some options to enhance the attack to overcome this apparent obstacle:

1. T and u_i can also collude with u_j, requiring u_j to store ID_i as well ID_{j+1}. If prompted from a verifier proceeding on the upper chain, u_j can reveal ID_{j+1} while from the lower chain, reveal ID_i; the requirement of u_j knowing which chain the verifier is proceeding on is discussed in the section on 'Attack Detection' below. This additional requirement for the attack requires a single additional collusion which is still much less work in comparison to the fake chain attack reviewed earlier.

2. Have $ID_j = ID_i$. This can be accomplished by having u_i periodically stamp (possibly meaningless) documents. For example, referring to Figure 4.5, an additional collusion would still be required with the user identified by ID_j, but if this user happens to be the attacker u_i, then no additional collusion is required.

The second option is clearly more favourable since no additional colluding partners are required. In either case, T has succeeded in absolutely back stamping y_i for user u_i. ■

Notice that if T is dishonest, then f is not necessarily a function at all; it is a relation. For example, from Figure 4.5 we have that $f(p_j) = p_{j+1}$ as well as $f(p_j) = p_i$. Therefore, rather than forming a total order, the set P of time stamp capsules forms a partial order. Let each stamp be a vertex in a directed graph with an edge from stamp a_i to stamp a_j if a_i was stamped before a_j and one can follow a directed chain from a_i to a_j (or vice-versa). Rather than exclusively forming a single chain, a tree is obtained, directed from the root. We have a tree (and hence no cycles) since each vertex has no more than a single incoming edge (dictated by the collision resistance of H), but can have more than one outgoing edge (allowing for the creation of multiple paths). Each path from root to leaf is a potentially valid chain which represents a total ordering on its own.

Attack Detection?

In the following, we consider different possibilities that might be suggested as a means for detecting the lower chain (see Figure 4.5) produced from the attack described in Lemma 4.1. We define a *valid state* as one in which only a single temporal chain has been produced by T (i.e., the valid chain in Figure 4.5). Attack detection is the discovery of a state that is not valid. For each of the cases discussed here, detection of the attack is possible only if certain preventive measures are taken and explicitly required during stamp creation and stamp verification (in addition to those given by Protocol RL1). Suggestions for attack prevention are given below.

In item 1 given above, for the next stamp in the "chain" after a_j, u_j stores ID_{j+1} and ID_i where $ID_{j+1} \neq ID_i$. ID_{j+1} refers to the upper chain of stamps while ID_i refers to the lower chain. In item 2, this "fork" in the temporal chain is advanced ahead one link. In other words, u_j stores $ID_{j+1} = ID_i$ pointing to each of the next stamps. On the other hand, u_i, possessing a stamp in both the upper and lower chains, holds $ID_{j+2} \neq ID_{i+1}$ referring respectively to the continuation of u_i's previously constructed stamp on the upper chain and backstamped document on the lower chain.

In the the first detection possibility, we examine possible scenarios in which a verifier v is traversing along the chains shown in Figure 4.5, traversing both from (backwards or forwards in the direction of time) and to (backwards or forwards in the direction of time) the stamps relevant to the attack; namely the stamps a_j, a_{j+1} and a_i and as well, those stamps legitimately added after a_{i-1} or a_i subsequent to the attack. The second detection possibility notes how a relative measurement included in the verification procedure might discover the attack. The third detection possibility discusses the limits of how far back a document might be time stamped.

Detection possibility #1. For either Item 1 or Item 2 above, proceeding backward or forward during a verification of the temporal chain starting *from* the stamp a_j (in the first case) or from the stamps both submitted and stored by u_i, a_i or a_{j+1} (in the second case), causes no suspicion on the part of the verifier v. This is because moving backwards from any of these stamps continues along the previously constructed valid chain and hence relies on the correctness of the protocol itself since a verifier will now encounter only legitimately produced stamps. Moving forward from a_{j+1} or a_i leads respectively forward on the upper or lower chains and does not cause suspicion in the case that stamps have been legitimately added to the upper and lower chains during the production of stamps produced subsequent to the attack. A similar situation occurs for a verifier proceeding through a verification to a_j, a_{j+1} or a_i from earlier stamps. For example, considering the case in which u_j (the owner of a_j) possesses both ID_{j+1} and ID_i indicating the newly created fork in the chain resulting from the attack, it does not matter which chain the verifier is sent on, e.g., v is sent on the upper chain if given ID_{j+1} by u_j.

However, consider item 1 (item 2 is analogous) and suppose that stamp a_{j+k} is currently being challenged where $(j+k) < (i-1)$ and thus the stamp appears in the upper chain (see Figure 4.5). Working backwards to a_{j+1}, the challenger will eventually obtain ID_j from L_{j+1} (which is stored by u_{j+1}) and hence asks u_j for his time stamp capsule, namely $\{a_j, ID'\}$. Notice that if the challenger were proceeding on the upper chain then he would expect $ID' = ID_{j+1}$ whereas on the lower chain he would expect $ID' = ID_i$. Notice also that u_j has no way of knowing which chain the challenger is proceeding on. However, consider that u_j may possess many stamps (all presumably along the same chain from the challenger's point of view). The challenger will have to inform u_j about which stamp he wishes to challenge. This may include information which identifies which chain he may be proceeding with his challenge on. (The protocol description given by Haber and Stornetta [HS91] is not specific enough to determine the exact steps taken during such a challenge.) As well, since the entire capsule (i.e., $\{a_j, ID'\}$) was not signed by T, there is no reason that any integrity should be expected to be associated with it by any challenger.

Detection possibility #2. A second possible suggestion for detection relates to the observation that subsequent to the partial insertion of the false (lower) chain (see Figure 4.5), maintenance of both the upper and lower chains requires that some stamps will share the same corresponding identification (round) number. However, the same identification number will only be shared by stamps that appear on different chains. Stamps will have unique identification numbers relative to the chain that they are on. Therefore, unless two such documents are compared for their relative positioning, such number repetition is not detected during a stamp verification. Performing a relative measure (see Section 4.3.3 as well as 'Attack Prevention' below) may detect this attack.

Detection possibility #3. A third method for possible detection involves the following observation regarding the limits to how far back in time a document might be time stamped. Depending on how far back a_i is partially inserted into the valid chain (to produce the lower chain – see Figure 4.5) the amount of time between the time recorded in the stamp for a_i and the stamp following a_i in the lower chain may be "uncomfortably large." Note that legitimately produced stamps following a_i will be stamped with a time that is at least as late as the (actual) current time. Whereas a_i (since it is being backstamped) will be stamped with a time that is earlier (possibly much earlier) than the (actual) current time.

However, it is difficult to determine how this might be interpreted by a challenger. Should Protocol RL1 require that an upper bound be placed on the amount of time that might elapse between the construction of two consecutive stamps? Prior to knowledge of this attack, such an additional constraint was unmotivated. Given knowledge of this attack, it may still be difficult to enforce.

Implications of detecting multiple chains. If it happened that multiple chains were detected and this evidence is given to an adjudicator, then this essentially brings some suspicion on T. At this point, T may claim that his private signature key must have been compromised. Either he can refute having created one temporal chain, or the other or even both. Note that this loss of key scenario is not the same as if we were to have a single chain for which T refutes some or all of the stamps that he produced. In such a case, the adjudicator may have a choice to believe or not believe T and to not accept or to accept the temporal chain. However, in the case given above, the adjudicator does not have this luxury. Even if he choses not to believe T, how can he tell which chain is correct? We note again here that such malicious action by T may be unlikely, though not impossible. Below, we discuss some additional measures that might be taken to limit the extent of the attack. One must also bear in mind that the relevance of this attack lies mainly in its exposure that assuming that T need not be trusted may be unreasonable.

Preventing the Attack on Protocol RL1

In the following, some enhancements to both the stamp production and verification as described by Protocol RL1 are presented. Two ways to prevent the aforementioned attack are

1. authentic storage of the stamps by a trusted authority and/or
2. treat the protocol as only a relative scheme.

These points were respectively discussed in Sections 4.2.1 and 4.3.3.

It is important to realize that the use of authenticated storage does not simply provide for an extra level of security (should one claim that T need not be trusted) since we have shown the scheme to be insecure without it. With this provision of authentic storage, producing an alternative chain is made more difficult if the one true chain is authentically verifiable. As well, notice that providing for such authentication of the time stamps is not simply an extra feature that can be added to the protocol since its addition produces an entirely new protocol – i.e., the interactions required for the verification protocol (the main feature of the scheme) appear to be unnecessary in this case. The authentication and storage of time stamps was discussed in more detail in Section 4.2.1.

For treating the protocol as only a relative scheme, there are some drawbacks. First is a loss of fine granularity. It can no longer be determined exactly when a document was time stamped, but rather only when it was time stamped relative to when other documents were time stamped. The provision of relative time was discussed in Section 4.3.3.

4.5 Hybrid Implementations

In this section, we critique one of the more widely used time stamping implementations. Protocol HY1 is currently used by Surety Technologies and is described by Haber and Stornetta [HS97, Section 2.4] and Trowbridge [Tro95]. This protocol is of interest since it closely follows the framework described in Section 4.2 (though it does so with a solution that is non-cryptographic).

In anticipation of longer requirements for the temporal authentication of user's data, Protocol HY1 uses a 288-bit hash function consisting of the concatentation of an MD5 [Riv92] and SHA-1 [FIP95] hash. In other words,

$$y = h(x) = MD5(x), SHA\text{-}1(x),$$

where ',' denotes concatenation. The advantage of this technique is discussed further in Section 5.5.

Referring to Protocol HY1, 'Verification II' is used in the case that the response for the original verification from T is in dispute, or requires further corroborative evidence. This is not all that unreasonable since no message authentication is provided for any of the communications performed by T. We can summarize several concerns with Protocol HY1:

1. *Authenticity of the Time Stamp.* (This same concern was noted for Protocol SM1 of Section 2.2.) Even after the execution of 'Verification I', the verifying user has little evidence assuring him of the correctness of the time stamp. This is a result of the lack of message authentication provided for the time stamp. Indeed, discrepancies between various copies of CD-ROMs require resolution using Verification II (discussed further in the next point).

2. *Adjudication.* A realistic concern relates to how the 'storage' used in Protocol HY1 will be treated in the event of disputes. In other words, what makes one copy of newspaper more trustworthy than another. This might require trusted archival of the newspaper at several sites, requiring participation of the archivists in case of a dispute. Indeed, the verification and adjudication protocols seem very difficult to automate.

Protocol HY1 A Hybrid Time Stamp using Widespread Storage [HS97].

```
Stamping
Input: ${y_1, \ldots, y_m}$
Output: $a_r$, $member_{y_i}$ from Protocol GH3 (see Section 2.3.2) and widespread storage
    of $a_r$
 1: User $u_i$ submits $y_i = h(x_i)$ to $T$.
 2: $T$ collects the submissions after 1 second (hence the rounds are 1 second in length)
    and uses Protocol GH3 to compute $a_r$.
 3: A so-called super-hash value (SHV) is computed for this $r$th round, using $a_r$ and
    the previous round's SHV:
        $SHV_r = h(SHV_{r-1}, a_r)$
 4: $SHV_r$ is recorded in a Universal Validation Record (UVR) along with the current
    time $t$ , i.e., as '$t$:$SHV_r$, $a_r$'.[a]
 5: $SHV_r$, $member_{y_i}$ and the time $t$ are returned to $u_i$.
Periodic Publication
 1: The UVR is periodically distributed (on a CD-ROM) to registered users.
 2: Each Sunday, the most recent SHV is published in the NY-Times.
Verification I
Input: $y_i$, $member_{y_i}$, $t$
Output: whether $y_i$ was time stamped at time $t$
 1: User $v$, verifying the purported time $t$ of stamping of $y_i$, submits $y_i$, $member_{y_i}$
    and $t$ to $T$.
 2: $T$ uses $y_i$ and $member_{y_i}$ to recompute $a_r$, uses $t$ to locate $SHV_{r-1}$ (in the UVR),
    computes $SHV'_r = h(SHV_{r-1}, a_r)$ and determines whether it is equal to $SHV_r$
    as recorded in the UVR.
 3: $T$ returns a success or failure response to $u_i$.
Verification II
Input: UVR and trusted copies of the NY Times
Output: Corroboration of the correctness of the UVR
 1: The SHV recorded in the NY Times is treated as a trusted intermediate stamp
    (see Definition 4.17(1)) as similarly described for Protocol RL2.
 2: Let $SHV_j$ represent the first SHV following $SHV_i$, published in the NY-Times.
    To verify the correctness of a particular $SHV_i$ in the UVR, the verifier $v$ computes
        $SHV_k = h(SHV_{k-1}, a_k),\ i \leq k \leq j$
    ensuring that for $i \leq k \leq (j-1)$, the values of $SHV_k$ match those given in the
    UVR and that the value of $SHV_j$ matches the value printed in the NY-Times.
```

[a]The description of Haber and Stornetta [HS97, Section 2.4] is not specific on this point whereas Trowbridge [Tro95] indicates the storage in the UVR as 't:SHV_r'. However, for 'Verification II', knowledge of a_r for each entry is required.

3. *Interoperability.* The ability to compare the relative order of time stamps produced by different time stamp authorities can be partially handled by the use of cross-stamps (see Definition 4.17(2)), and as well by the recording of an absolute time in the UVR.

As an alternative, we propose Protocol HY2, which modifies Protocol HY1 to deal with some of the concerns provided above by providing

1. *Message Authentication.* The stamp is signed by a time stamp authority (T) and verified upon return to the submitter of the data to be time stamped.

2. *Authenticated Storage.* The stamp is recorded and maintained by a storage authority (S) who is independent of T.

and following the framework described in Section 4.2. Rather than using linking for a relative ordering, a counter is used.

With regard to trust in authorities, we take the view of protecting against attacks *to* the authorities rather than attacks *by* the authorities. For example, notice that T and S could collude by having T periodically advance the round counter r ahead by one position, thereby allowing subseqent backdating. Indeed, T could even collude with S to backdate, simply by repeating an r which has already been used, so long as the detection of this is not too likely. Rather, Protocol HY2 protects against malicious attacks that might occur against *either* T or S. Notice that if T's private key is compromised, the backdating of documents would require storage of stamps with an incorrect form by S, e.g., with an old round number r. Likewise, a compromise of S would require collusion with T to produce a T-signed signature.

Notice that the use of a storage authority allows for the provision of corroborative evidence regarding the time stamp s (see Section 4.2.1). The advantage is that the submitting user u is unaware of the provision of storage since it is done off-line with regard to the communication between u and T. Regarding the choice of group hash in Protocol HY2, referring to Table 3.2, Protocol GH3 is the most favourable for limiting the size of a_r and has the second smallest size for $member_{y_i}$ when less than 2^7 data are group hashed.

Protocol HY2 A Hybrid Time Stamp Proposal.

```
Stamping
Input: $\{y_1, \ldots, y_m\}$
Output: $a_r$, $member_{y_i}$ from Protocol GH3
1: User $u_i$ submits $y_i = h(x_i)$ and a time granularity request to $T$.
2: $T$ collects the submissions and queues each based on the granularity requested by
   each user (see Remark 4.5).
3: At the end of each appropriate time interval, $T$ uses Protocol GH3 to compute
   $a_r$ for the appropriate queues.
4: $T$ computes $s = sig_T(r, a_r, t)$ for absolute time $t$ and returns
       $r, member_{y_i}, s, t$   to $u$
              $r, a_r, s, t$   to storage authority $S$.
5: $u_i$ computes $a'_r$ using Protocol GH3 with $y_i$ and $member_{y_i}$ as input, and verifies
   that $s$ is indeed a signature over $(r, a'_r, t)$.
6: $u_i$ stores $(r, t, member_{y_i}, y_i, x_i, s)$.
7: $S$ verifies that $s$ is a signature over the received $(r, a_r, t)$; verifies that $t$ is within
   $t' \pm \delta$ of the current time for small $\delta$ and ensures that $r$ is 1 greater than the last
   received from $T$.
8: $S$ stores $(r, t, a_r, s)$.
Verification
Input: $(r, t, member_{y_i}, y_i, x_i, s)$
Output: whether $y_i$ was time stamped at time $t$
1: User $v$, verifying the purported time $t$ of stamping of $y_i$, computes $y'_i = h(x_i)$ and
   ensures that $y'_i = y_i$.
2: $v$ computes $a'_r$ using Protocol GH3 with input $y_i$ and $member_{y_i}$ and verifies that
   $s$ is a signature over $(r, a'_r, t)$.
3: (Optional.) $v$ contacts $S$ and verifies the existence of the entry $(r, a_r, t, s)$ stored
   by $S$. (This step is also useful in the case that $T$'s private signature key is
   compromised.)
```

Remark 4.5 *(Variable Time Granularity.) The granularity of a group hash operation is determined by the length of the round, which may be fixed or variable, and depend on either the length of time or number of submissions received. A variable time based granularity might work as follows. Note that not all users (submitting a request for a time stamp) obtain the same granularity. Some users require a time stamp to the nearest second, others to the nearest minute or hour. The granularities required by each user can be requested by the submitting users, whereby the time stamp provider can subsequently place each request in separate bins, e.g., a 'second-bin', a 'minute-bin' and an 'hour-bin'. At the end of each second, the second-queue would be processed for group hashing followed by time stamping. Likewise, every minute and hour respectively for the minute- and hour-queues.*

Chapter 5

Time Stamping Digital Signatures

A time stamping protocol provides for the temporal authentication of digital data. The input to a time stamping protocol is viewed as no more than a string of bits. In some cases, input data possessing particular properties allows for additional conclusions to be drawn regarding the data both during the production and verification of a time stamp. One such form of data is that which has an authentic lifetime (i.e., validity period) associated with it.

In this chapter, we focus on the time stamping (temporal authentication) of digital signatures. A public key used to validate a signature is typically, in practice, contained in a certificate; the lifetime of the certificate can be (as we assume in this chapter) constrained by a finite validity period. By time stamping a digital signature, subsequent verification can determine if the signature was produced when the corresponding certificate was valid. Alternatively, this signature validity imposed by the finite validity period of the corresponding certificate can also be verified during time stamp production thereby preventing the time stamping (and hence acceptance) of signatures produced subsequent to a certificate's expiry. This "notarization" of digital signatures determines the correctness of the input and decides whether or not to notarize based on this determination. The time stamping or notarization of signatures also introduces the extension of a signature's message authentication beyond the expected lifetime of the original digital signature algorithm. This chapter examines each of these concerns, providing general techniques allowing a system to obtain clear and consistent verifications of a digital signature.

Chapter Outline

In Section 5.1, several definitions regarding the association of a finite validity period with data are presented. The relevance of these concepts is discussed in relation to remaining sections in this chapter. In Section 5.2, the construction, distribution, maintenance and trust in public-key certificates is reviewed. In Section 5.3, we examine the effect that time has on the status of a digital signature validation. Section 5.3.1 motivates and presents requirements that allow for consistent verifications of a digitial signature over time. In particular, Definition 5.13 presents the components of a *certificate-based signature scheme with time stamping.* The time stamping and long-term storage of a signature and its corresponding verification certificate's status are identified as key requirements for consistent signature verifications. Section 5.3.2 fulfills these requirements in more detail with the presentation of the *signature verification* Protocol DS1. Practical concerns regarding the process of certificate revocation are highlighted by Figure 5.10. The role of a signature dispute adjudicator is presented as a signature verification by a trusted third party. In Section 5.4, we present a *digital signature notary* as a trusted third party that establishes the truth of various statements regarding the status of a digital signature and its corresponding verification certificate at various points in time. This *notarization* of digital signatures is presented as Protocol NT2. In Section 5.5, we review the concept of digital signature renewal. The subtle problems offered by a time stamping solution are reviewed and the application of a notarization solution is presented.

5.1 Data With Inherent Time

Beyond the *existence* of data at a particular time as may be identified by the time stamping of the data, a data item may be defined by a finite *validity period* or *lifetime*. *Creation* and *expiry dates* delimit the lifetime of the data.

Definition 5.1 The *lifetime* of data $y \in \mathcal{Y}$ is the output returned by the function $lifeTime : \mathcal{Y} \rightarrow (\mathcal{T} \cup -\infty) \times (\mathcal{T} \cup \infty)$ where $\mathcal{T}$ represents a finite set of possible *times*. For each data $y \in \mathcal{Y}$, $lifeTime(y)[1]$ (the first element of the ordered pair) represents the *creation date* (see Definition 5.2) for y whereas $lifeTime(y)[2]$ (the second element of the ordered pair) represents the *expiry date* (see Definition 5.4) for y. Undefined creation and expiry dates are represented by $-\infty$ and ∞ respectively.

As an example of a data's lifetime, a public key certificate's lifetime is parameterized by a creation date cd and expiry date ed (see Section 5.2.1). The lifetime of user u's certificate is thus denoted as $lifeTime(cert_u) = (cd, ed)$. In this section, we discuss the usefulness of the concept of a data lifetime with particular emphasis on the lifetime of a digital signature and its corresponding public key certificate.

Definition 5.2 A *creation date* c_y associated with a data item y, is a verifiable or recoverable date cryptographically bound to y indicating the start of a validity period for that data. Data that is distinguished (see Definition 5.3) can have a creation date uniquely associated with it.

A time stamp provides authentically verifiable recognition of the existence of some data at a particular point in time. The first time that data is time stamped can define a creation date for the data. However, since the same data can be time stamped numerous times it may be difficult to determine the earliest or unique creation date for the data. A particular time stamp merely implies that the time stamped data was created *no later than* the time of stamping. It may have been created, even time stamped, at an earlier time.

Since the same data can be time stamped many times, if the data is distinguished from other data at the time of application of the creation date, can a unique creation date be associated with this distinguished data. This motivates the following definition.

Definition 5.3 A set $S = \{s_1, \ldots, s_k\}$ of binary data is *distinguished* if for any pair of elements (s_i, s_j), $i \neq j$, either $length(s_i) \neq length(s_j)$ or if $length(s_i) = length(s_j) = n$, then $s_i - s_j \neq 0^n$.

For a set S that is not distinguished, one can create an alternative, distinguished representation for S, namely the set S' where for each element $s \in S$ a corresponding element $s' \in S'$ can be created by either of the following techniques.

1. *Associating a unique identifier with s.* For example, a public key can be assigned a unique serial number (see Section 5.2 for further details regarding certificate construction) representing a unique or distinguished certificate. Although there may exist several different certificates for a particular public key (by inclusion of the same public key in different certificates), each certificate is unique as identified by its serial number.

2. *Removing duplicate elements from S.* An alternative to the association of a unique identifier as performed above would involve an authority that only issued a single certificate for each public key. Subsequent to the initial request and issuance of a certificate for a particular public key, all subsequent requests with the same public key would be rejected.

Definition 5.4 An *expiry date* e_y associated with a data item y, is a verifiable or recoverable date cryptographically bound to y indicating the end of a validity period for that data. Data that is distinguished (see Definition 5.3) can have an expiry date uniquely associated with it.

The date of expiry of a public key is incorporated within its corresponding certificate. A patent also has a specific date of expiry, relative to its date of filing, i.e., currently 20 years after the date of filing.

Definition 5.5 Data $y \in \mathcal{Y}$ is said to be *alive* as of time t if $alive(y, t) = \text{true}$, where $alive : \mathcal{Y} \times \mathcal{T} \rightarrow \{\text{true, false}\}$ is defined by the following:

$$alive(y,t) = \begin{cases} \text{true} & \text{if } lifeTime(y)[1] \preceq t \prec lifeTime(y)[2] \\ \text{false} & \text{otherwise} \end{cases}$$

■

This general concept of "aliveness" can be used for various concepts regarding a data's aliveness status. For the example of a public key certificate $cert_u$ (see Section 5.2.1) belonging to user u, $alive(cert_u, t)$ (written as $expired(cert_u, t)$ in Definition 5.10) is true if and only if $cd \preceq t \prec ed$ where cd and ed are the respective creation and expiry dates of the certificate.

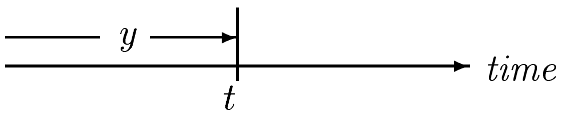

Figure 5.1: A timeline representation of data y in which only the expiry date t is known (or relevant).

Data need not necessarily have either an authentic creation or expiry date. Figure 5.1 gives a pictorial representation for data y that has an expiry date yet no creation date, i.e., $lifeTime(y) = (-\infty, e_y)$. For example, consider the issuance of a club membership, although the creation date may be known (since it is the time of issuance of the membership), it may not be necessary at some subsequent time. The membership card may therefore only require an indication of the expiry date. Figure 5.2 gives a pictorial representation for data y that has a creation date yet no expiry date, i.e., $lifeTime(y) = (c_y, \infty)$. For example, consider the receipt of a univerity degree. Although this degree may have a related creation date (identified by the date of completing the degree requirements), it typically has no such expiry date.

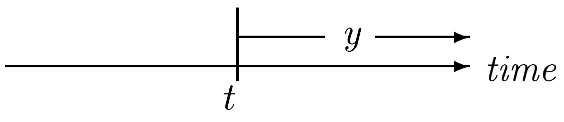

Figure 5.2: A timeline representation of data y in which only the creation date t is known (or relevant).

It is important to note that a lifetime is not necessarily fixed. Although the creation and expiry dates, once set, do not change, data can prematurely expire. For example, although a credit card has an expiry date, abuse or loss of the card subsequent to its issuance may result in a premature removal of the privileges associated with the card, amounting to an early expiry. For an example more relevant to the remainder of this chapter, consider the creation and expiry dates contained within a public key certificate. As a result of a revocation (see Section 5.2.3), the "expiry" of the certificate may occur prior to the originally intended date of expiry.

5.1.1 Relevance to Temporal Authentication

The role of a time stamper is to time stamp raw data for which any semantics associated with the data are considered irrelevant. Yet for data with some associated attributes or auxiliary information (i.e., some meaning associated with the data), additional conclusions can be drawn with regard to the data. In particular, consider data with some cryptographically associated timing information, e.g., a creation and/or expiry date. This timing information may be relevant either during time stamp production or verification as respectively described below.

1. An authority uses the timing information for deciding whether or not to authenticate the submitted data, e.g., apply a signature to the data. In other words, the data y might be authenticated at time t only if $alive(y, t) =$ true. For example, if data is not currently considered to be alive, the time stamp operation might fail. In this way, the time stamper is acting as a *notary* by positively attesting to the current 'liveness' of the data. The role of a notary is discussed in Section 5.4. As an example, a notary might only authenticate a signature if the signature was received at a time when the corresponding certificate of the signing user has not yet expired.

2. The data is authenticated by a trusted authority as before, without any interpretation of timing information (associated with the data) during the provision of message authentication. Data from sources used to provide the auxiliary timing information for the data (e.g., its creation and/or expiry date) are used to draw further conclusions regarding the message authentication of the data during verification or adjudication. For example, if a digital signature is time stamped as opposed to notarized, subsequent verification is required to determine if it was stamped during a time when the public key certificate of the

signer was valid (i.e., had not yet expired nor been revoked).

5.1.2 Extending a Finite Lifetime

The lifetime of data input to a time stamp or notary authority can be undefined and hence assumed infinite in the direction of increasing time, unless it possesses some inherent or alternative timing information. A time stamp often has a finite lifetime in both directions (e.g., based on prudence should the stamp construction be based on complexity theoretic assumptions). The end of a time stamp's lifetime may be

1. parameterized by an expiry date included in the stamp or
2. dictated by some external source, e.g., in the signature verification certificate of the entity who created the stamp

The pair consisting of data y and its corresponding time stamp constitute a document of their own that can be input to a time stamp or notary authority. In this way, the finite lifetime of the original time stamp can be extended by the production of a subsequent time stamp. This procedure can be used as a process of *renewing* a time stamp. With regard to digital signatures, just as successive time stamping serves to renew former time stamps, the time stamping of a digital signature may *extend* the lifetime associated with the digital signature (as defined by the validity period of the public key certificate). This extension and renewal of authentication is discussed in Section 5.5.

5.1.3 Implications for Backward and Forward Stamping

In this subsection, we present some interesting implications (for data possessing inherent timing information) with respect to the back (see Definitions 4.10 and 4.15) or forward stamping (see Definitions 4.11 and 4.16) of time stamps. For cases in which only an expiry date associated with data is relevant (see Figure 5.1), only the prevention of (absolute and relative) forward stamping is typically necessary. Producing a false back stamp for y is considered irrelevant since the goal of an attack would be to stamp y as far forward as possible. This more clearly explains the example given by Benaloh and de Mare [BdM91] stating that a photo of somone holding the Magna Carta does not give evidence that this person was alive in the year 1215 A.D. It gives no evidence because back stamping is not relevant with such a scheme. The scheme is only designed to show the latest date that something has not yet expired (i.e., this person is alive as of this time), and has no relation to the date of creation. As well, there are cases where it is not important when y expires, only when it was created (see Figure 5.2). Thus, only the prevention of backward stamping are typically relevant here. Remark 5.1 summarizes these concerns.

Remark 5.1 *The prevention of backward stamping is typically unnecessary for data whose lifetime is unbounded in the direction of decreasing time. The prevention of forward stamping is typically unnecessary for data whose lifetime is unbounded in the direction of increasing time.*

A finite lifetime also has some interesting implications with regard to the extent to which data can be back or forward stamped. Suppose that a digital signature c has some inherent timing information (i.e., the validity period for producing c dictated by the corresponding verification certificate), defining a creation date t (for the corresponding certificate). Let t' be the current time and the time that might be legitimately associated with c by an absolute time stamp. The absolute back stamping of c might associate a time $t'' \prec t'$ with c. However, it makes little sense for t'' to be less than t (it is generally assumed that subsequent verification or adjudication would deem this as disallowable). Therefore, in cases where the data c possesses some associated timing information, the ability to provide a back stamp for c may be constrained by the associated lifetime for c. Similarly results hold for forward stamping. This point is summarized by Remark 5.2.

Remark 5.2 *The lifetime of data as parameterized by its creation and expiry dates may be used to constrain the range in which data can be forward or back stamped.*

For a document that has no associated timing information, forward stamping cannot be prevented. For the particular example of a digital signature, the submission of the signature to a time stamp authority can merely be postponed an indefinite amount of time. In this case, forward stamping cannot be prevented. This observation was made by Haber and Stornetta [HS91] (where the term used was forward dating), though as indicated with Property 5.2, successful forward stamping can be prevented in the event that the data has an associated expiry date.

5.2 Public Key Certificates – Background

The format of a certificate was mentioned briefly in Section 2.1.2. In Section 5.2.1, we expand on a certificate's construction and contents. In Section 5.2.2 we discuss how certificates are obtained by users (both the certificate owner and verifiers of signed messages from the certificate owner). As well, we recall issues of trust related to certificates, e.g., how does one user trust a certificate received from another user? Section 5.2.3 reviews the purpose and methods of certificate revocation.

5.2.1 Certificate Construction

Certificates are typically constructed by a certification authority (CA). A CA-signed user certificate authentically binds a user's name to a public key.

Definition 5.6 A *(user) certificate* $cert_u$ is defined as a data structure containing at least the following elements,

$$cert_u = \{n_u, I_u, p_u, cd, ed, sig_{CA}(n_u, I_u, p_u, cd, ed)\},$$

where each element in $cert_u$ is identified as follows:

n_u: a unique certificate serial number;

I_u: a distinguished subject name uniquely identifying u among all other users within the name space relevant or controlled by this CA;

p_u: a public key;

(cd, ed): a validity period for the certificate, denoted by the creation date cd and expiry date ed for the certificate;

$sig_{CA}(\cdots)$: the signature of the CA over the items listed above (as well as any other items potentially included in the certificate).

Possible additional entries include alternative names or attributes related to the owner or key. ■

Users may have several certificates, distinguished by different serial numbers. As a specific example, u might have different public keys and hence certificates for encryption or for digital signatures. Unless specified otherwise, we assume that p_u is a signature verification key.

Users can obtain their own certificate(s) from the CA through various techniques. For example, employees might be issued a certificate upon joining a corporation and be physically given a disk containing the CA-signed certificate, which can subsequently can be uploaded by the user to their personal computer. An important detail regarding the issuance of a certificate is the verification of the identity I_u of the certificate requestor and ensuring that this person is indeed the owner of the verification key p_u (where ownership here means that u has knowledge of the signature key s_u corresponding to p_u). Various "levels" of certificates can exist depending on the diligence of the CA to validate this identification.

Validity Period of a Certificate

The use and interpretation regarding a validity period can vary for a certificate. Let us first consider variances in the length of the validity period. For example, in some systems, certificates might have an indefinite expiry date. Alternatively, certificates might only be considered valid for a short period of time (e.g., one hour) after their issuance. Throughout this thesis, we assume that a certificate has a finite validity period (of reasonably long length, i.e., useful for a number of signature productions) defined by its creation and expiry dates.

It is also important to consider the implications regarding a certificate's expiry. Figure 5.3 gives two possible interpretations. Relative to the validity period, one can also consider signing and verification periods. The signing period delimits the time after which the signature producer should no longer produce signatures, nor should

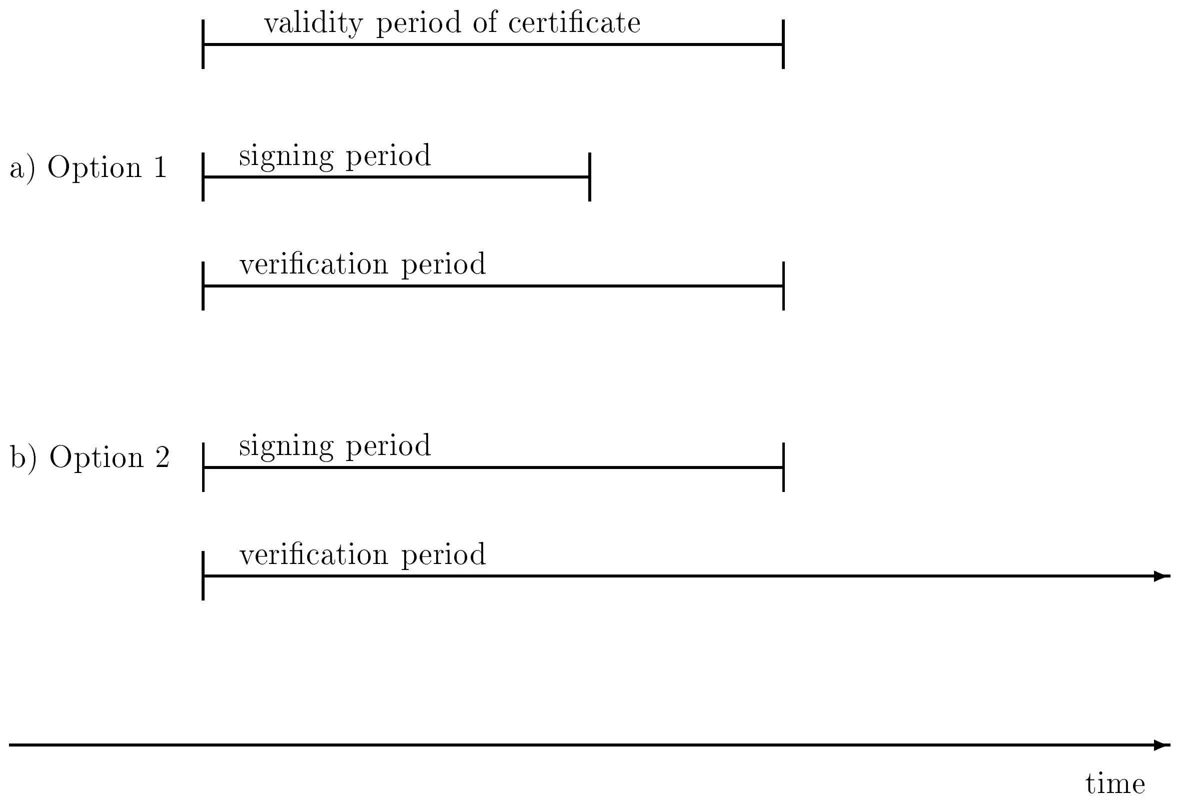

Figure 5.3: Two views of the signing and verification periods for a signature relative to the validity of the verification certificate.

signatures received after this time be verified (the latter applies only to Figure 5.3(b)). The verification period delimits the time after which the verifier of a signature can no longer trust the corresponding certificate and hence should no longer attempt to verify such signatures. Signatures received prior to the end of the certificate's validity may still be verified until the end of the verification period.

For the first option (see Figure 5.3(a)), the signing period is a fraction of the validity period while the period in which signatures can be verified ends with the expiry of the certificate. Verification subsequent to the certificate's expiry would require a renewal of the signature's authenticity (see Section 5.5). For the second option (see Figure 5.3(b)), signatures can be produced and hence verified as valid until the end of the certificate's validity period. The signature can be verified at any time, though only signatures received prior to the signature period expiry (i.e., expiry of the certificate) will be verifiable. These options are discussed further in Section 5.3.1.

Additional complications may result from the fact that the lifetime of certificates is not always fixed. Key compromise and employee dismissal are examples of two situations which might warrant an earlier expiry or *revocation* of a certificate. In Section 5.2.3, implications of this early termination are discussed. In Section 5.3, we discuss the distribution and attainment of trust in certificates.

5.2.2 Certificate Distribution and Trust

Trust in a public key can be obtained by verifying the cryptographic binding between the name and public key contained within the CA-signed certificate. This verification can be performed using a copy of the CA's verification key, obtained, for example, when a user first obtains their certificate from the CA, e.g., through a physical meeting to initiate their relationship with the CA. This verification key may be contained, as we assume in this chapter, in a *self-signed certificate* produced by and containing the verification key of the CA.[1] When one user receives a certificate from another, the user certificate can then be validated using the CA's verification key. User's certificates

[1]The expiry or revocation of this certificate is beyond the scope of this thesis and hence the validity period of such certificates is assumed to be unbounded in the direction of increasing time.

can be distributed amongst themselves by many methods, including:

1. Sent to the recipient by the message originator, most likely accompanying a signed message;
2. Stored in a certificate repository (database) and obtained by using one of the following techniques.
   (a) *Pull.* Message recipients obtain certificates as necessary.
   (b) *Push.* A CA distributes newly created certificates at the time of their creation or at periodic intervals;
3. Cached by recipients, from an initial distribution using either of the methods above.

Multiple Certification Authorities

For reasons of scalability and diversity, it may be impractical for all users' certificates to be managed by a single certification authority (CA). Scalability concerns may result if *all users* have certificates issued from a *single CA*. This places a tremendous burden on the CA for the construction and distribution of the certificates. Diversity concerns result from requiring *all users* to accept the services of a *single CA*. Such services might include, for example, the maintenance of revocation information (see Section 5.2.3). Users with certificates used for high risk transactions might require a CA that performs frequent broadcasts of revocation information. However, this may be unnecessary for certificate owners with more modest certificate requirements. Although a variety of services can be offered by a single CA, a large and ever-increasing number of users can make this task overbearing for a CA.

One solution to single-CA limitations is to use multiple certification authorities. However, this solution introduces some additional complications related to the *trust* a user has in a particular certificate. Consider the first two cases shown in Figure 5.4. In Figure 5.4(a), v is able to validate u's verification certificate (as part of the validation of a u-signed message; see Section 5.3) since v already possesses a copy of the CAs

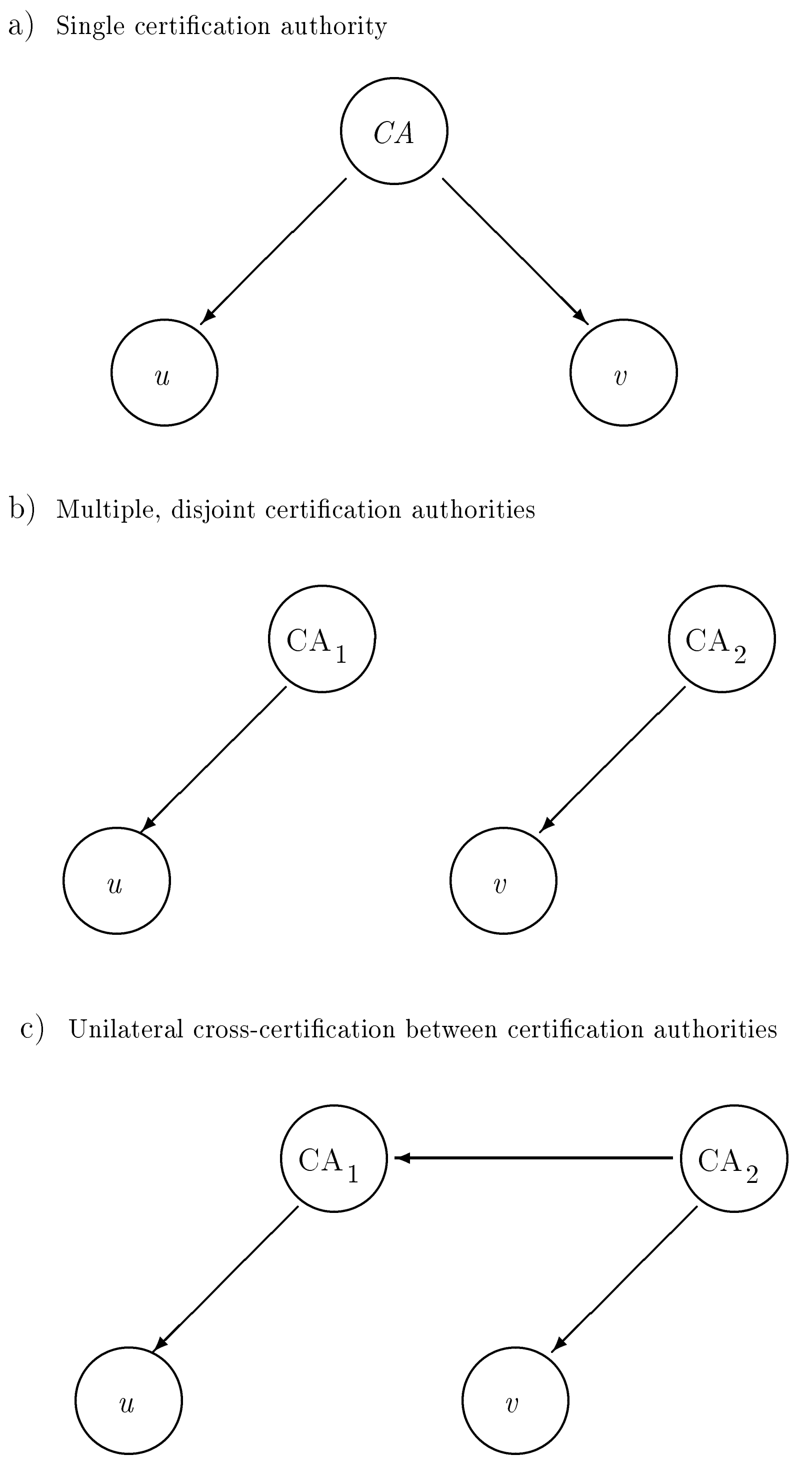
a) Single certification authority
CA
u
v
b) Multiple, disjoint certification authorities
CA$_1$
CA$_2$
u
v
c) Unilateral cross-certification between certification authorities
CA$_1$
CA$_2$
u
v

Figure 5.4: Single, Disjoint and Cross-Certified Certification Authorities (CAs). (a) Users u and v have certificates produced by the same CA. (b) Users u and v have certificates produced respectively by the certification authorities CA_1 and CA_2. (c) CA_2 has cross-certified CA_1, producing the cross-certificate $CA_2\{CA_1\}$.

public key (since the same CA produced v's certificate). For reasons of scalability and diversity indicated above, u and v might have certificates produced by different CAs, as in Figure 5.4(b). In this situation, v is unable to validate u's verification certificate without the public key of CA_1; and v only possesses the public key of CA_2.

To allow v to validate u's certificate (or any other user's certificate produced by a "foreign" CA), v requires an *authentic* copy of CA_1's public key. One possible solution is to obtain it in the same way that CA_2's public key was obtained, e.g., by a physical meeting. However, in the case that there is a large number of CAs, such a solution doesn't offer an efficient means for v to validate signatures whose corresponding certificates are produced by "foreign" CAs. An alternative solution involves treating CA_1's public key as that of a normal user. CA_2 can then certify the public key of CA_1 as well. This solution is described below.

Trust Relationships and Their Certification

Definition 5.7 A *cross-certificate* $CA_i\{CA_j\}$ is a certificate created and signed by the certification authority CA_i, binding the name and public key of certification authority CA_j. ■

In Figure 5.4(c), CA_2 cross-certifies CA_1 allowing v to validate u's certificate by

1. initially possessing a trusted copy of CA_2's verification key,

2. obtaining the cross-certificate $CA_2\{CA_1\}$, either accompanying a signature sent by u or from a directory maintained by CA_2,

3. using CA_2's verification key to validate the cross-certificate $CA_2\{CA_1\}$, thereby allowing v to trust CA_1's verification key,

4. using CA_1's verification key to validate user u's certificate.

v is able to gain trust in both CA_1 and u by respectively validating the CA_2- and CA_1-signing of certificates. Notice that the cross-certificate $CA_2\{CA_1\}$ does not imply the existence of the certificate $CA_1\{CA_2\}$. Therefore, for the example above, $CA_1\{CA_2\}$ is not available to allow u to verify v's certificate, even though $CA_2\{CA_1\}$ exists.

Definition 5.8 Let $CA_{i_k}\{u\}$ represent $cert_u$ (see Definition 5.6) as produced by CA_{i_k}. The collection of certificates,

$$(CA_{i_1}\{CA_{i_2}\}, CA_{i_2}\{CA_{i_3}\}, \ldots, CA_{i_{k-2}}\{CA_{i_{k-1}}\}, CA_{i_{k-1}}\{CA_{i_k}\}, CA_{i_k}\{u\})$$

is called a *certificate chain.* For simplicity, this certificate chain may also be denoted as

$$(CA_{i_1}, CA_{i_2}, CA_{i_3}, \ldots, CA_{i_{k-2}}, CA_{i_{k-1}}, CA_{i_k}, u)$$

■

We say that entity A *trusts* entity B if A can successfully verify the authenticity of B's purported certificate, e.g., with a certificate chain. An entity refers to either a user or a certification authority. See Definition 5.12 for a more formal definition of trust.

Certificate validation can become quite complex. For example, extending the example in Figure 5.4(c), suppose that CA_1 and CA_2 did not *directly* cross-certify each other, but rather, each cross-certified with CA_3. An extra verification would be required by both u and v for the validation of their respective certificates. A complete discussion is beyond the scope of this thesis. See Menezes *et al.* [MvOV97, Section 13.6.2] and Ford *et al.* [FB97, Section 7.2] for further information.

5.2.3 Certificate Revocation

The lifetime of a certificate may be shortened due to a *revocation* of a user's certificate. In the case of a signature verification certificate, it may imply that once a particular certificate has been revoked, signatures produced subsequent to the time of revocation with the corresponding signing key are no longer considered valid and hence no longer accepted. The authority to revoke a certificate should be sufficiently restricted in order to prevent a user from maliciously revoking the certificate of another user (resulting in a denial of service). For example, the user named in the certificate as well as the issuing CA may be allowed to initiate a revocation. For certain environments, an employer may also be able to request the revocation of an employee's corporation-issued certificate. Reasons for revoking a user's certificate include the following:

1. suspected or detected compromise of the signing key (either the user's or CA's private key);

2. change of security requirements in anticipation of or in reponse to a protocol failure, e.g., increasing the CA's key size in anticipation of new cryptanalytic attacks;

3. change of information contained in the certificate, e.g., changing the distinguished name of the owner;

4. revocation of privileges associated with the certificate, e.g., subsequent to dismissal, an employee may have his corporation-issued certificate revoked;

5. change of role within an organization, e.g., moving from one department to another.

Other entities possessing copies of a currently revoked certificate (without knowing that it is currently revoked) must be able to learn of the certificate's shortened lifetime so that signatures produced subsequent to the date of revocation, are not accepted.

Distributing Revocation Information

There are a number of techniques for conveying revocation information to potential signature recipients. A certificate revocation list (CRL) typically contains at least, for each revoked certificate, the certificate serial number and the date of revocation. As well, the name of the issuer (e.g., the certification authority (CA) that originally issued the certificate) and the issue date of the CRL are included. This list is signed by the issuing CA.

Definition 5.9 A *certificate revocation list (CRL)* crl_t is a list created and signed at time t by the certification authority CA_i containing at least the following information:

1. the unique certificate serial number n_u, corresponding to each $cert_u$ (see Definition 5.6) previously issued by CA_i, that is revoked as of time t;

2. the time rev_u of revocation of $cert_u$;

3. the time t of construction of the CRL; and
4. the signature of CA_i over the above information.

More specifically,

$$crl_t = \{(n_{i_1}, rev_{i_1}), \ldots, (n_{i_p}, rev_{i_p}), t, sig_{CA_i}((n_{i_1}, rev_{i_1}), \ldots, (n_{i_p}, rev_{i_p}), t)\}$$

■

CRLs can be distributed using techniques similar to the methods for distributing certificates described in Section 5.2.2, as described below.

Push. For example, using the *push* method, the CA broadcasts the CRL to all "relying parties", i.e., users that will verify certificates. This can be done either periodically or may be *event-based*, e.g., subsequent to each revocation. The periodicity of the distribution depends on the policy of the CA, e.g., every hour versus once a day. Both the periodic and event-based options are susceptible to an attacker intercepting and stopping the delivery of the revocation information. For this reason, the implementation of a periodic distribution may include a field within the current CRL indicating the time of the next update. Relative temporal authentication (see Section 4.3.3) allows one to detect the deletion of revocation information for the event-based option, e.g., by using a sequence number for each CRL. The CRL broadcast may consist of all currently revoked certificates or alternatively, the most recently revoked (see delta-CRL discussion on page 134).

Pull. Alternatively, the *pull* method can be used whereby users request revocation information directly from the CA or more commonly a directory/repository. In case that a user may be anticipating a loss of (e.g., online) access to CRL information for a period of time, it might be prudent for a user to obtain the entire, most recent CRL. As an alternative to a CRL, a user might only query for information regarding a particular certificate. The latter has the disadvantage of increased overhead resulting from multiple pulls, one required for each certificate query. However, there may be decreased storage at the user's site by requesting only a single certificate. If the user caches the results of the individual certificate or CRL queries, one disadvantage is the staleness of the revocation information.

We can expand on the pull technique described above (for individual certificate information) where a real-time method is used (i.e., online certificate status check). Each certificate to be validated involves a query to the CA for the status of the certificate. The response from the CA may consist of either

1. a signed response from the CA indicating the status of the certificate (without returning a copy of the actual certificate itself) or

2. returning the certificate. This can be accomplished by returning the certificate with a statement from the CA indicating its status or make use of extended fields within the certificate to indicate whether the particular certificate is revoked or not, e.g., in the CA's directory of certificates, the original certificate might be replaced by a copy in which the "revocation bit" is set to 1.

Scalability concerns are present here in case of a large number of users under a particular CA. These include bottlenecks occurring from many users requesting information from a single CA directory as well as designating a single point of failure in the case of CA down-time or even worse, a CA whose information is maliciously corrupted.

On one hand, it appears prudent to combine the functionality of certificate distribution with the distribution of revocation information by having the CA return a freshly signed certificate subsequent to a request from a potential signature recipient. In this way, the validity period of the certificate can be limited to a very short time (as indicated in Section 5.2.1). However this limits the usefulness of the signature for applications requiring non-repudiation (see Section 2.1.2) guarantees. Alternatively, returning the status of the certificate allows the CA to perform less computation and return a smaller response. As well, users may prefer the option of only obtaining a status check for a fraction of certificates, i.e., rely on cached versions of the certificate or certificates sent directly from the signature originator.

Freshness of Revocation Information. It is important to note here the freshness of revocation status information in the possession of a certificate relying party. Among

Push by CA		Pull by User			
CRL Broadcast		CRL Request		Individual Cert. Request	
Event-Based	Periodic	Event-Based	Periodic	Certificate	Certificate Status

Figure 5.5: Classification of techniques for distributing/obtaining certificate revocation information.

other complications (see Figure 5.10), even in the case where an on-line certificate status check is used, the certificate owner (and originator of some signature) may be in the process of revoking their certificate when the status information is being obtained by a verifying party. This point is discussed further in Section 5.3.2. The techniques for distributing certificate revocation information are summarized in Figure 5.5.

Reducing CRL Size

One disadvantage of using certificate revocation lists (CRLs) versus individual certificate revocation queries is their size. This has an effect on the bandwidth used to transmit CRL information from the CA to individual users and on the ability of users to accept and store the information at their local machines. In a large community of users (under a particular CA), a signature recipient may only require a small number of certificates and likely only a small portion of the information provided by a CRL. Obtaining individual certificate information is convenient though can be more difficult for scalability for a single CA or directory. In what follows, we examine the distribution of information through CRLs, and expand on three techniques for reducing their size. Information regarding CRL distribution points and delta-CRLs was obtained from Ford *et al.* [FB97] and Menezes *et al.* [MvOV97]. The solution using group hashing (see Section 2.3) was presented by Kocher [Koc98].

CRL Distribution Points. CRL distribution points partition CRLs, limiting each distribution point (itself a smaller CRL) to grow only to a fixed maximum size. Whereas certificate relying parties previously requested revocation information from the corresponding CA that signed the certificate in question, distribution points introduce some indirection:

1. *Certificate redirection.* An additional entry within each user's certificate indicating which distribution points (e.g., CRL, directory entry, or other location) may contain revocation information with regard to the particular certificate.
2. *CRL redirection.* An additional entry within the original CA's CRL indicating the location of distribution points which may contain revocation information regarding certificates that would otherwise have appeared in the CRL.

CRL distribution points may be distinguished by the class of certificates they hold. For example, there may be 10 distribution points, each containing revocation information for the different possible last digits in a certificate's serial number. As well, the distribution points might be distinguished by a "reason code" where certificates that have been revoked as a result of a suspected or detected key compromise might only reside at a particular distribution point.

Delta-CRLs. Delta-CRLs reduce the size of revocation information that must be downloaded at each push or pull of a CRL. A so-called base CRL is issued first. Subsequently, delta-CRLs are issued (e.g., periodically or event-based), which only contain information regarding revoked certificates since the last base CRL was issued. These delta-CRLs contain a pointer to the base CRL. A relative ordering (see Section 4.3.3) allows detection of maliciously deleted delta-CRL transmissions from a CA.

Group Hashing. Group hashing can be used to reduce the size of the information that must be obtained by a signature verifier, while potentially increasing the size of the information accompanying the signature from the originator. Consider the following use of the group hashing Protocol GH3 (see Section 2.3.2) given by Kocher [Koc98].

At time t, a trusted tree issuer (TTI) obtains up-to-date revocation information (e.g., using CRLs) and produces statements indicating the range of certificates revoked by each CA. For example, if CA_1 has 2 revoked certificates with serial numbers 121 and 300, then the TTI would produce the following statements:

$$\text{If} \quad CA_1 \quad \text{and} \; -\infty \leq X < 121 \text{ then } X \text{ is revoked iff } X = -\infty$$

$$\text{If} \quad CA_1 \quad \text{and } 121 \leq X < 300 \text{ then } X \text{ is revoked iff } X = 121$$
$$\text{If} \quad CA_1 \quad \text{and } 300 \leq X < \infty \text{ then } X \text{ is revoked iff } X = 300$$

Such statements are made for the revoked certificates of each CA and used as the input to Protocol GH3. The output a_r is time stamped by the TTI giving $s = sig_{TTI}(a_r, time)$ and made available to all potential signature verifiers just as a CRL would.

When sending a signature, user u would obtain the supporting evidence that best describes his certificate. More specifically, u would obtain $member_{y_i}$ from Protocol GH3 where y_i is the hash of the statement that answers the question regarding the position of u's certificate, i.e., the statement indicating the range into which u's certificate number lies. This information is obtained from the TTI and sent by u, accompanying a signature. A signature verifier determines the correctness of the information by computing a'_r as a function of the user's certificate and accompanying information, and determining whether $a'_r = a_r$.

One advantage of this scheme concerns the size of information required by the verifier, i.e., a signature of a single hash value. Also notice that only a single signature verification allows multiple certificates to be validated for their revocation status. However, as with CRLs, fresh information (in this case, a new tree root) must be obtained when subsequent revocations are performed.

We note here that a tradeoff between size and computational cost for the TTI would involve the use of an efficiently incremental group hash (see Definition 3.2). Though computationally more efficient for the TTI, each of the remaining group hash protocols from Section 2.3 would increase the size of either a_r or $member_{y_i}$. Protocol GH5 has the attractive property of not requiring any additional information (beyond the originator's certificate and statement y_i) to be sent with the signature (since $|member_{y_i}| = 0$).

5.3 Time Stamping Digital Signatures

In this section we motivate and examine the relationship between time and the production, verification and adjudication of digital signatures. More specifically, we identify the importance and relevance of changes in trust regarding a signature that is verified over time. The following are identified as key requirements allowing for repeated, consistent verification of a digital signature:

1. the temporal specification of *when* the message was signed;
2. the maintenance of relevant certificate information evidence from *when* the message was signed, allowing subsequent verifications to be performed using this evidence from the point in time at which the message was originally signed, including
   (a) the long-term storage of certificate revocation information, and
   (b) the long-term storage of cross-certificates.

To the author's best knowledge, this section presents the first complete analysis of these concepts.

Haber, Kaliski and Stornetta [HKS95] include the most recent discussion of how time stamping can be used to support digital signatures. The (signature, message) pair is time stamped by T and widely published (see Section 5.3). Any challenge to the validity of the signature involves comparing the time of stamping by T with the time of reported loss of sig_u by the originating sender u of the message. Pinto and Freitas [PF96] include similar mention of how time stamping isolates a point in time when a message was signed (i.e., the receiver of a message time stamps the message upon receipt). Each identifies the importance of time stamping the signature but does not recognize the importance of Item 2 above.

In Section 5.3.1, the basic digital signature model is reviewed and requirements for the production of digital signatures are identified and examined. In Section 5.3.2, we describe in more detail, the processes involved in the production, verification and adjudication of digital signatures.

5.3.1 Digital Signature Requirements

In this subsection, we examine the process of verifying a digital signature. The main goal is to discover requirements that can be satisfied during signature production so as to make all subsequent verifications, including possible dispute adjudication, as trustworthy, reliable and consistent as possible. More specifically, we examine changes in the status of verification certificates over time, potentially altering the result of what might have once been a successful signature validation. It is the temporal recording of these changes over time, that allow a once successfully verified signature to remain as such at points in the future.

Webster's Dictionary [Mer98] defines a *dispute* as a "a verbal controversy." Rather than resorting to a term such as "cyber-dispute", we recognize the use of voiceless communication, and refer to a dispute simply as some form of a controversy. The important consideration is the cause of the controversy. The basic protocol over which a dispute might occur is depicted in Figure 5.6. The simplicity conveyed by this snapshot of a particular point in time is misleading. The construction and distribution of the certificate precede the signature's transmission. The certificate's expiry or potential revocation typically follow it. Verifications of the signature may occur at numerous points of time subsequent to the signature production. A number of actions may occur from the time the signature was created till the time(s) it is verified. Disputes can occur with respect to the occurrence and time of occurrence of each of these events. Table 5.1 lists the assumptions made, that define the base model in which digital signatures are produced, transmitted and verified.

We consider the potential for dispute regarding the production of a digital signature based on the intended use for that signature.

1. *Short-term requirement.* A signature is received and verified. If successful, some short-term privilege is given to the originator, e.g., access granted to a connection. If unsuccessful, the signature is rejected and no privilege is granted.[2]

2. *Long-term requirement.* The signature is received and verified. If successful, the

[2]This short term requirement is better suited to entity authentication (see Menezes *et al.* [MvOV97, Definition 10.1]) as opposed to temporal authentication.

Multiple Certification Authorities (CAs) There is a set $\mathcal{CA} = \{CA_1, \ldots, CA_k\}$ of certification authorities, each of whom maintain a read-only public database from which users query relevant information.

Cross-Certificates Cross-certificates (see Definition 5.7) may exist between any pair of CAs. These certificates are added to the database of the creating CA and removed subsequent to their expiry.

Revoked Cross-Certificates The revocation of cross-certificates is added periodically to the database maintained by the issuing CA as an *authority revocation list (ARL)*. An ARL is synonymous to a CRL (see Definition 5.9) except that it contains reference to revoked cross-certificates as opposed to revoked user certificates. The ith ARL arl_{t_i}, posted at time t_i contains the serial number and revocation date for all revoked cross-certificates previously issued by the CA, and overwrites the $(i-1)$st ARL $arl_{t_{i-1}}$ previously posted at time t_{i-1}. Certificates which are expired as of time t_i are included on arl_{t_i} if and only if they were not on $arl_{t_{i-1}}$.

User Certificates Each user u has a certificate $CA_i\{u\} = cert_u$ issued by CA_i and is assumed to have a trusted copy of CA_i's self-signed certificate $CA_i\{CA_i\}$, containing the public verification key of CA_i. The expiry or revocation of this self-signed certificate is beyond the scope of this thesis. Where reference to the CA is evident or not required, we refer to the certificate $cert_u$ as opposed to $CA_i\{u\}$. Each user certificate is added to CA_i's database subsequent to its creation and deleted subsequent to its expiry.

Revoked User Certificates The revocation of user's certificates is added periodically to the database maintained by the issuing CA as a CRL (see Definition 5.9). The CRL crl_{t_i}, posted at time t_i contains the serial number and revocation date for all revoked certificates previously issued by CA_i, and overwrites the $(i-1)$st CRL $crl_{t_{i-1}}$ previously posted at time t_{i-1}. Certificates which are expired as of time t_i are included on arl_{t_i} if and only if they were not on $arl_{t_{i-1}}$.

Globally Trusted Time Stamp Authority The is a globally trusted time stamp authority T, where "globally" here means across the entire community of relying parties who must rely upon the time stamp authority's signatures. Each user maintains a trusted copy of T's self-signed certificate $cert_T$, containing the public verification key ver_T for T. The expiry or revocation of this certificate is beyond the scope of this thesis.

Table 5.1: Requirements for the Digital Signature Model. This table contains a list of requirements and assumptions made to allow for the verification of signed messages, assuming the bounding of the signature and verification periods as in Figure 5.3(a). Section 5.3.1 demonstrates some limits of these requirements and several enhancements are presented in Table 5.3.

u		v
message m	$\xrightarrow{c = sig_u(m), cert_u}$	verify $cert_u, c$

Figure 5.6: Generic signature sending from originator u to recipient v.

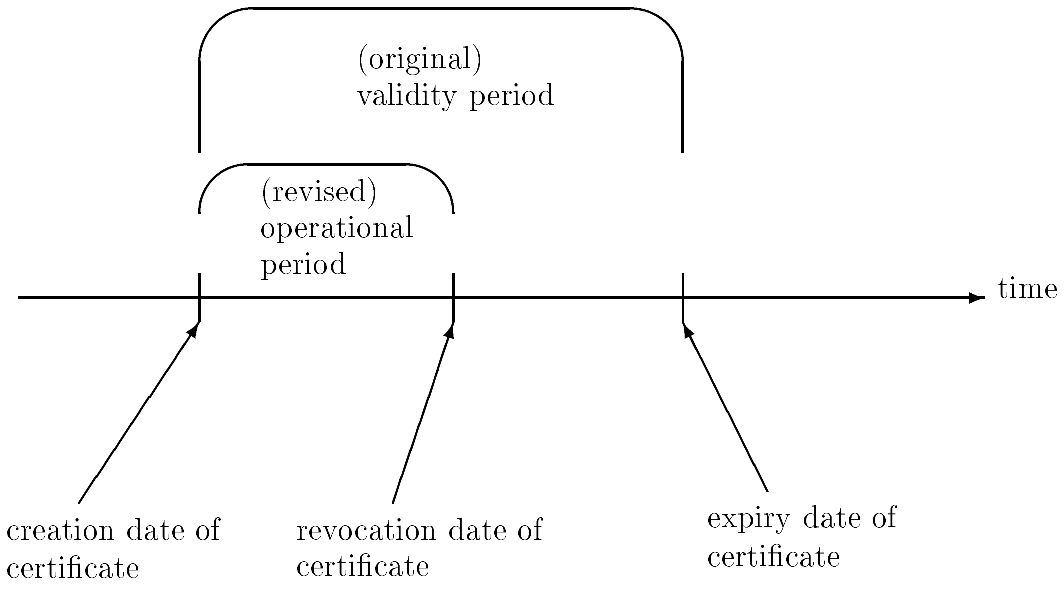

Figure 5.7: Periods of Change in a Certificate's Status.

signature is stored by the recipient as evidence of some form of commitment by the signature originator. If unsuccessful, the signature is rejected.

The distinguishing feature between these uses is that the former requires a single verification at the time of receipt of the signature, while the latter may require subsequent, *consistent* verifications, and hence, the existence of timely evidence for these verifications. In the remainder of this subsection, requirements for the long-term verification of signatures are identified.

Signature Verification

In order to validate a digital signature c, purportedly originating from user u, a verifier v performs the following:

1. *Verify signature correctness.* Verify the mathematical correctness of the sigature c, purportedly for the message m and purportedly produced by user u. This verification is performed using the verification key ver_u contained in $cert_u$, e.g., the signature verification for DSA from Protocol SG1 (see Section 2.1.2).

2. *Determine certificate status.* Determine the status of the signature originator's verification certificate, including both of the following.

   (a) *Validity.* A determination of whether the certificate is currently expired, and therefore not valid, or revoked, and therefore not operational (see Figure 5.7).

   (b) *Trust.* The trust measured from the point of view of v relative to u. In other words, is there a chain of certificates (see Definition 5.8) available to validate (and hence obtain trust in) $cert_u$ (see Definition 5.12)?

Changes Over Time

The status of a certificate can change over time. The verification of a digital signature determines the status of the certificate and hence, of the signature, relative to the first verification of the signature and its corresponding certificate. Verification subsequent to the initial receipt of the signature is likely performed relative to the current time at which the subsequent verification is performed and therefore, relative to the current status of the certificate. The signature may be validated numerous times. Given that the information used to validate the signature may change over time, it is important to recall the state of matters at the time when the signature was produced in order for the signature to be fairly and consistently validated at these later times. (In other words, validated relative to the state of information at the time the signature was produced.)

Remark 5.3 *A signature that is verified against information that is current as of the time of verification permits the possibility that different results may be obtained from validations of the same signature over time.*

Problems can arise from the fact that the status of the signature originator's ceritificate might differ from when a signature was originally verified. We identify here, the importance of changes regarding a certificate's expiry, revocation or trust, relative to the verifying party, for repeated, consistent verifications of a signature.

(1) Certificate Expiry. The expiry (end of the validity period) of a certificate introduces some restraints regarding the production and verification of a signature. More specifically, it may not allow for subsequent trustworthy and consistent verification and hence, may not be suitable for long-term signature requirements unless the signature verification procedure is enhanced. Consider the following definition which returns a measure of a certificate's validity (expiration) status over time.

Definition 5.10 Let $expired : \mathcal{C} \times \mathcal{T} \rightarrow \{\text{true,false}\}$ represent the function such that for the finite set of possible certificates $\mathcal{C}$, where $\mathcal{T}$ is a finite set of possible times,

$$expired(cert_u, t) = \begin{cases} \text{false} & \text{if } cd \preceq t \prec ed \\ \text{true} & \text{otherwise} \end{cases}$$

where $cd = lifeTime(cert_u)[1]$ is the creation date of $cert_u$ and $ed = lifeTime(cert_u)[2]$ is the expiry date of $cert_u \in \mathcal{C}$. Therefore, $expired(cert_u, t)$ is true if $cert_u$ is *valid* at time t (see Figure 5.7). ■

Recall the two interpretations regarding the expiry of a certificate from Figure 5.3 of Section 5.2.1. For Figure 5.3(a), the expiry of the certificate effectively ends the life of any signature requiring verification with an expired certificate. Verification of a signature at time $t \succ ed$ should not be performed with certificate $cert_u$ since $expired(cert_u, t) = true$. Hence, verification or adjudication subsequent to the expiry is made very difficult. For Figure 5.3(b), although verification is permissible subsequent to the certificate's expiry (i.e., even though $expired(cert_u, t) = true$), verification or adjudication cannot ensure when the signature was produced relative to the certificate's expiry. Besides possibly local evidence maintained by a recipient, indicating the time of signature receipt (which would not typically be considered trustworthy in any case), there is in general, no evidence that would indicate to a third party when the signature was actually produced.

(2) Certificate Revocation. Suppose that a signature is produced during the validity period of the corresponding verification certificate. If the certificate is revoked prior to its expiry, it is placed on a CRL. Prior to the expiry of the certificate, the revocation status of the certificate can be obtained from the proper CA's database.

Definition 5.11 Let $revoked : \mathcal{C} \times \mathcal{T} \rightarrow \{\text{true, false}\}$ represent the function such that, for the finite set of possible certificates $\mathcal{C}$, where $\mathcal{T}$ is a finite set of possible times,

$$revoked(cert_u, t) = \begin{cases} \text{true} & \exists\ crl_{t'} \text{ (see Definition 5.9) such that } (n_u, rev_u) \in crl_{t'} \\ & \text{for } rev_u \in \mathcal{T} \text{ where } rev_u \preceq t \\ \text{false} & \text{otherwise} \end{cases}$$

Therefore, $revoked(cert_u, t)$ is false if $cert_u$ is *operational* at time t (see Figure 5.7). ■

Thus, one can determine the revocation status for any time t, by determining if $cert_u$ is contained on a CRL prior to time t. However, one must also be able to determine when a signature was produced, relative to this revocation.

(3) Change in Trust. Suppose that a signature recipient verifies a signature upon its receipt and is able to obtain a chain of certificates (see Definition 5.8) that demonstrate trust in the signature originator's certificate. At some time in the future, the trust between CAs may change so that the signature verifier no longer trusts any signatures currently received from this signature originator. However, if the recipient has not maintained previous cross-certificates (as well as information regarding their possible revocation), there may be no evidence that there was once trust in the original signature. The temporal changes in trust are captured by the following definition.

Definition 5.12 Let $certTrust : \mathcal{C} \times \mathcal{C} \times \mathcal{T} \rightarrow \{\text{true,false}\}$ such that for the *verifying user* $v \in \mathcal{U}$ who initially trusts the certificate $CA_{i_1}\{CA_{i_2}\}$ (it may be that $i_1 = i_2$), and user $u \in \mathcal{U}$ possessing a certificate $cert_u \in \mathcal{C}$, for the finite set of possible times $\mathcal{T}$,

$$certTrust_v(CA_{i_1}\{CA_{i_2}\}, cert_u, t) = \begin{cases} \text{true} & \text{if } \exists \text{ a certificate chain} \\ & (CA_{i_1}, CA_{i_2}, \ldots, CA_{i_k}, u) \\ & \text{(see Definition 5.8)} \\ & \text{that is mathematically correct} \\ & \text{AND} \\ & \text{for each certificate } CA_i\{CA_j\} \\ & expired(CA_i\{CA_j\}, t) = \text{false} \\ & \text{AND} \\ & revoked(CA_i\{CA_j\}, t) = \text{false} \\ \text{false} & \text{otherwise} \end{cases}$$

The certificate $CA_{i_1}\{CA_{i_2}\}$ is referred to as v's *trust anchor.* ■

Providing Temporal Authentication

The finite length of the lifetime of a certificate for each user is defined by its validity period. In this sense, absolute temporal authentication is provided for certificates by the inclusion of the validity period and the message authentication (i.e., signature) of the CA over (amongst other items) both the validity period and the public key. One might think then that signatures have a lifetime imposed on them by the lifetime of the certificate corresponding to the particular signature key used to sign the message, i.e., indicating that the signature was produced during the lifetime (prior to the expiry) of the certificate. However, the lifetime of a certificate does not necessarily imply a lifetime for corresponding signatures beyond the point in time in which the signature is first validated (or more specifically beyond a revocation or expiry of the certificate). Although not providing sufficient evidence for subsequent verifications, information is provided by the validity period contained within the certificate to affect the verification of the signature upon receipt.

Remark 5.4 *The validity period of a public key certificate can be used to impose*

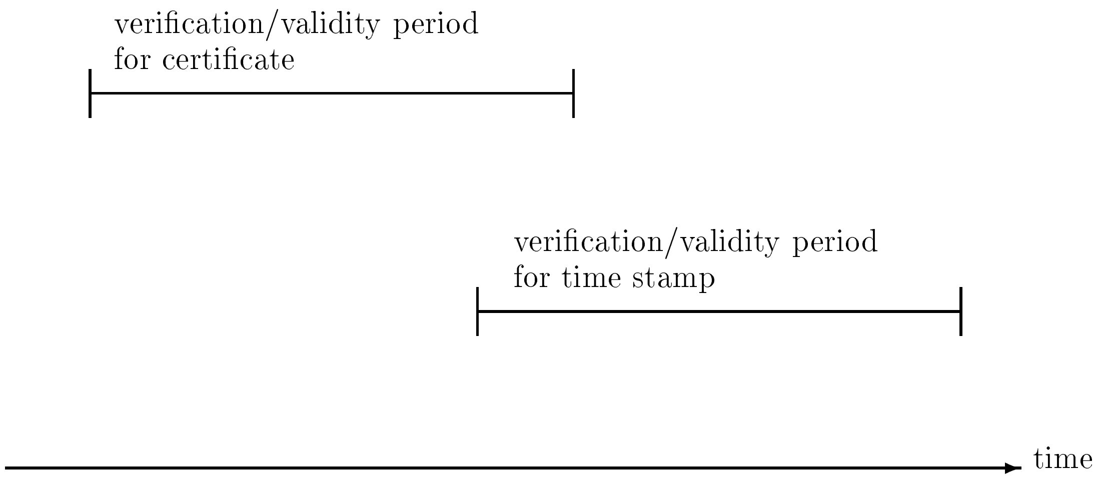

Figure 5.8: Certificate Verification Life Extended with Time Stamp. The validity period of the certificate is the period during which signatures could be verified (see Figure 5.3(a)). Without time stamping, the period beyond the expiry (i.e., end of the validity) of the certificate would not allow for trustworthy verification. The validity period of the time stamp indicates that a signature was time stamped prior to the expiry of the corresponding verification certificate. This signature can be verified so long as the time stamp is trustworthy.

a restriction in a signature verification procedure whereby signatures verified with a currently expired certificate need not be accepted.

The time stamping of a digital signature establishes the existence of the signature at a fixed point in time, thereby allowing subsequent verifications to be performed relative to this time. The time stamping of a signature serves two functions:

1. *Fixes Point in Time.* The time stamping pinpoints the time, allowing one to measure against changes in certificate status (expiry or revocation) or certificate trust (changes in cross-certificates). This solves the limitations of option 2 from Figure 5.3(b) in which verification subsequent to the expiry of the certificate was unable to determine when the signature was produced, i.e., whether it was produced prior to the expiry of the certificate. It also allows for the status of certificates relevant to the signature verification to be measured as of the time of stamping of the signature.

2. *Extends Lifetime.* As indicated by Figure 5.8, the time stamping can extend the lifetime of the original signature (see Section 5.5) beyond a possible finite verification period defined for a certificate (as with option 1 of Figure 5.3(a)). The period of time, after the expiry of the signature verification certificate, that the signature can be validated is then determined by the lifetime of the time stamp, e.g., depends on the expiry of the verification certificate corresponding to the time stamp authority. As for Item 1 above, that also allows determination of the corresponding certificate's status as of the time of stamping of the signature.

In this way the time stamping of the digital signature uses the best qualities from the two options given in Figure 5.3. The original user certificate maintains a finite verification lifetime (as in option 1) which is important for limiting the damage in case of the undetected key compromise of a user's private signature key, for example. Yet time stamped signatures may still be verified with a corresponding certificate even though the certificate may be currently expired, revoked or in which a verifier cannot currently validate the trust. This *temporal verification* of time stamped signatures is captured by the following definition, an enhancement of the signature scheme of Definition 2.11.

Definition 5.13 A *certificate-based signature scheme with time stamping* CSTS is a $six-tuple$ $(\mathcal{SS}, \mathcal{TS}, \mathcal{CA}, \mathcal{U}, \mathcal{V}, \mathcal{C})$, where the following conditions are satisfied:

1. $\mathcal{SS}$ is a signature scheme (see Definition 2.11);
2. $\mathcal{TS}$ is a time stamping scheme (see Definition 4.1);
3. $\mathcal{CA}$ is a finite set of possible certification authorities (CAs);
4. $\mathcal{U}$ is a finite set of possible users within the name space of the certification authority $CA_i \in \mathcal{CA}$;
5. $\mathcal{V}$ is a set of possible users within the name space of the certification authority $CA_j \in \mathcal{CA}$;
6. $\mathcal{C}$ is a finite set of possible certificates;

7. $validSig_v : \mathcal{M} \times \mathcal{Q} \times \mathcal{S} \times \mathcal{T} \times \mathcal{C} \times \mathcal{C} \rightarrow$ {true,false} is a function such that the following equation is satisfied for every message $m \in \mathcal{M}$, signature $c \in \mathcal{Q}$, time stamp $s \in \mathcal{S}$, time $t \in \mathcal{T}$, certificate $cert_u \in \mathcal{C}$ and certificate relying party $v \in \mathcal{V}$ with trust anchor $CA_i\{CA_{i'}\} \in \mathcal{C}$:

$$\begin{array}{l} validSig_v(m, c, s, t, \\ cert_u, CA_i\{CA_{i'}\}) \end{array} = \begin{cases} \text{true} & \text{if } ver_u(m, c) = \text{true (see Definition 2.11)} \\ & \text{AND} \\ & ver_T(s, t, c) = \text{true (see Definition 4.1)} \\ & \text{AND} \\ & expired(cert_u, t) = \text{false} \\ & \text{(see Definition 5.10)} \\ & \text{AND} \\ & revoked(cert_u, t) = \text{false} \\ & \text{(see Definition 5.11)} \\ & \text{AND} \\ & certTrust_v(CA_i\{CA_{i'}\}, cert_u, t) = \text{true} \\ & \text{(see Definition 5.12)} \\ \text{false} & \text{otherwise} \end{cases} \tag{5.1}$$

A signature c is *valid* if $validSig(m, c, s, t, cert_u, v) = \text{true}$. ■

Intuitively, (5.1) states that a signature, time stamped at time t (which may be earlier than the current time), is valid so long as the corresponding certificate was not expired nor revoked, and trustworthy (with respect to the verifier) at time t and that each signature and certificate is mathematically correct. The implementation of each of the functions described by (5.1) are presented as Protocol DS1 in Section 5.3.2.

In the following, we discuss enhancements to the requirements of Table 5.1 permitting the proper evaluations of the functions in (5.1) of Definition 5.13. In particular, we identify the need for the long-term storage of information that allows subsequent determination of a certificate's status from earlier times.

1. the message signed, in precisely the bit representation (canonical form) for which the signature was generated,
2. the originator's verification certificate,
3. the time stamp, computed over the signature,
4. the verification certificate of the time stamp authority T,
5. revocation information related to the originator's and T's certificates,
6. any cross-certificates required to obtain trust in the originator's certificate,
7. revocation information related to all cross-certificates used in the validation of the user's certificate.

Table 5.2: Evidence Required for Signature Verifications. Referring to Definition 5.13, this table lists the information that is required for all verifications of a digital signature.

Providing Long-Term Temporal Storage

Differing slightly in purpose from the storage required for time stamping (see Section 4.2.1), the storage of information necessary for the validation of digital signatures is required for functionality as well as the maintenance of evidence (in case of disputes). We assume that all current certificate information (e.g., certificate itself, revocation information, cross-certificates) is maintained in a database by the certification authority (CA) up to the point when the certificate expires. This is based on the assumption that subsequent to the expiry of the certificate, the signature should not be subsequently verified.

However, with the addition of time stamping, relevant certificate information is required after the time of expiry of the certificate. We refer to the storage of this information as *long-term storage*.[3] Referring to (5.1) of Definition 5.13, to determine the validity of a signature c, at any time, requires the computation of the truth values of each of the listed functions. The material necessary to achieve this is listed in

[3]In Section 5.4, we see how a notary may remove the requirement for the long-term storage.

Cross-Certificates Modifying the requirements of Table 5.1, cross-certificates must be made available subsequent to their expiry in the case that a signature time stamped prior to their expiry, requires an expired certificate for verification of the signature. Therefore, expired cross-certificates remain in the issuing CA's database.

Revoked Cross-Certificates Modifying the requirements of Table 5.1, past ARLs are required. Therefore, rather than storing only the most recent ARL, all ARLs are stored in an ARL database which adds the most recent ARL to the database while not removing past ARLs. Expired cross-certificates are removed from individual ARLs that are posted to the database for reasons of efficiency.

Revoked User-Certificates Similar to revoked cross-certificates (above), a CRL database is used in which current CRLs are added to the database.

Table 5.3: Assumptions Made for Time Stamped Digital Signature Model. This table enhances the requirements listed in Table 5.1 by including requirements made necessary for the long-term, consistent verification of (time stamped) digital signatures.

Table 5.2. This information is required for as long as the verifier requires evidence that the signature originator did indeed sign the signature at a particular time. Table 5.3 presents a list of modified requirements of the previous assumptions from Table 5.1. In Section 5.3.2, it is shown how this information is used during the verification of digital signatures.

5.3.2 Signature Verification and Adjudication

In Section 5.3.1, the verification of digital signatures was reviewed and temporal requirements allowing for the long-term validation of signatures were explicitly presented. In this section, we examine in more detail the steps undertaken by the verifier of a digital signature. We expand the discussion from Section 5.3.1, where now, v verifies a time stamped digital signature. More specifically, for the message m, v determines the correctness of

$$\{m, sig_u(m), sig_T(sig_u(m), t), cert_u{}^*\}$$

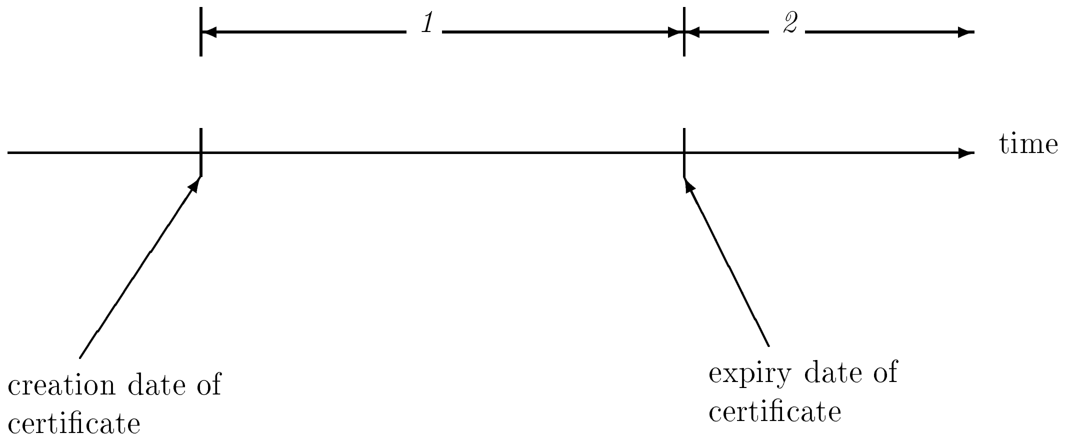

Figure 5.9: Signing periods for a non-revoked certificate. Time period 1 indicates the period of time during which the certificate remains valid. Time period 2 indicates the period of time after which the certificate has expired.

where $sig_u(m)$ is u's purported signature on m, $sig_T()$ represents a purported signing of the contents by the time stamp authority (T) and t represents the purported time of stamping (e.g., possibly the time of submission $sig_u(m)$ to T), and $cert_u$ indicates the purported verification certificate of u. An asterisk (*) indicates an optional item, e.g., $cert_u$ may have been previously cached by v.

A relative time can be provided and used for determining the order of two signed messages. Since we are concerned with certificates whose lifetime is finite and parameterized by a validity period, an absolute time must be used for certificates themselves since the expiry time cannot be indicated with a relative time. Time comparisons between relatively stamped documents was discussed in Section 4.3.3. In this section, since are concerned with the time stamping of documents relative to absolutely stamped certificates, we consider only the absolute stamping of the signatures.

The verification of a signature can be described directly from the functional operations given by (5.1) of Definition 5.13. This signature verification protocol is described as Protocol DS1.

Referring to Figure 5.9, certificates validated during period 1 may be verified using the most current CRL and ARL information obtained from corresponding CA databases. During period 2, any user certificates or cross-certificates used in the

Protocol DS1 Signature Verification Protocol.

Description: This protocol describes the verification by v of a signature purportedly originating from user u. The verification is based on a certificate-based signature scheme with time stamping (see Definition 5.13). Table 5.3 lists protocol assumptions and requirements.

Note: For each computation in which the truth value of a function is determined, if unsuccessful, c is rejected as invalid. If successful, continue, where the requirements for success of each function are specified by (5.1) of Definition 5.13. v accepts c as valid only if all steps are successful.

Input: message m, purported signature $c = sig_u(m)$, purported time stamp $s = sig_T(c,t)$, purported time t of stamping, purported certificate $cert_u$ of signature originator u, and trust anchor $CA_{i_1}\{CA_{i_2}\}$ of verifier v

Output: result of $validSig_v(m, c, s, t, cert_u, CA_{i_1}\{CA_{i_2}\})$ (see Definition 5.13)

1: *(Signature correctness.)* Given the message m and signature $c = sig_u(m)$, verifier v determines the mathematical correctness of c by using the verification key ver_u from $cert_u$ and determining the truth value of $ver_u(m, c)$ (see Definition 2.11).

2: *(Time stamp correctness.)* Given the time stamp s, time t of stamping and signature c, verifier v determines the mathematical correctness of s by using the verification key ver_T from (the stored copy of) $cert_T$ and determining the truth value of $ver_T(s, t, c)$ (see Definition 4.1). If true, the time t is accepted as the time of stamping of c.

3: *(Certificate expiry.)* v ensures that c was time stamped during the validity period of $cert_u$ by determining the truth value of $expired(cert_u, t)$ (see Definition 5.10).

4: *(Certificate revocation.)* v ensures that c was time stamped during the operational period of $cert_u$ by determining the truth value of $revoked(cert_u, t)$ (see Definition 5.11). Revocation information regarding $CA_i\{u\} = cert_u$ is obtained from CA_i's CRL database. The CRL $crl_{t'}$ is obtained where t' is the latest time (recorded for a CRL posting) that is earlier than the date of expiry of $cert_u$. If this indicates the most recent CRL then, depending on the policy of the verifier, the verification may be delayed until the next CRL is posted by CA_i.

5: *(Certificate Trust.)* v determines $certTrust_v(CA_{i_1}\{CA_{i_2}\}, cert_u, t)$'s truth value. As implied by Definition 5.12, v must first determine a certificate chain, starting from $CA_{i_1}\{CA_{i_2}\}$, allowing trust to be obtained in $cert_u$. The construction of this chain is beyond the scope of this document. Once a certificate chain is constructed, each cross-certificate $CA_i\{CA_j\}$ is verified by ensuring that both $expired(CA_i\{CA_j\}, t)$ and $revoked(CA_i\{CA_j\}, t)$ are false. The expiry of each cross-certificate (as of time t) can be determined using the expiry date contained in each certificate. Revocation information regarding each cross-certificate $CA_i\{CA_j\}$ is obtained from CA_i's ARL database. The ARL $arl_{t'}$ is obtained where t' is the latest time (recorded for an ARL posting) that is earlier than the date of expiry of $CA_i\{CA_j\}$. If this indicates the most recent ARL then, depending on the policy of the verifier, the verification may be delayed until the next CRL is posted by CA_i.

verification of a signature will typically not appear on the most recent revocation lists. Hence the requirement for maintaining long-term storage of this information. And although the originating user's certificate (which is fixed for each verifier) should be accompanied with the signature, cross-certificates (which typically vary, depending on the verifier) are stored and made available long-term, by the issuing CA.

Complications Regarding Revocation

Systems which allow revocation of certificates introduce additional challenges. Referring to Figure 5.10, one can recognize the additional complexities that revocation creates. Consider a request for a revocation of $cert_u$ made by user u at time t_1, resulting from a suspected or detected key compromise. A signature forged at time t where $t_1 \prec t \prec t_2$ might wrongly be accepted by a verifier (without knowledge of the revocation) should a CRL have been issued at time t' where $t \prec t' \prec t_5$ and used for verification of the signature as in Protocol DS1. Using event-based as opposed to periodic CRL distribution may allow some speedup in the receipt of revocation details by potential signature verifiers, i.e., by decreasing the time required between t_3 and t_4. In addition, other means for minimizing delays between each of the intervals from Figure 5.10 would be advantageous.

Adjudicating Disputes

Adjudication is a form of verification. A mutually agreed upon, impartial *judge* (adjudicator of a dispute) J is queried by user u (the purported signature originator) or v (a signature verifier) to resolve a dispute regarding

1. the expected commitment resulting from a digitally signed statement, or
2. the validation status of a signature.

The discussion of a commitment from a signature is beyond the scope of this thesis. We present here, the role of a judge in determining the validity of a signature, as defined in Definition 5.13 and determined by Protocol DS1.

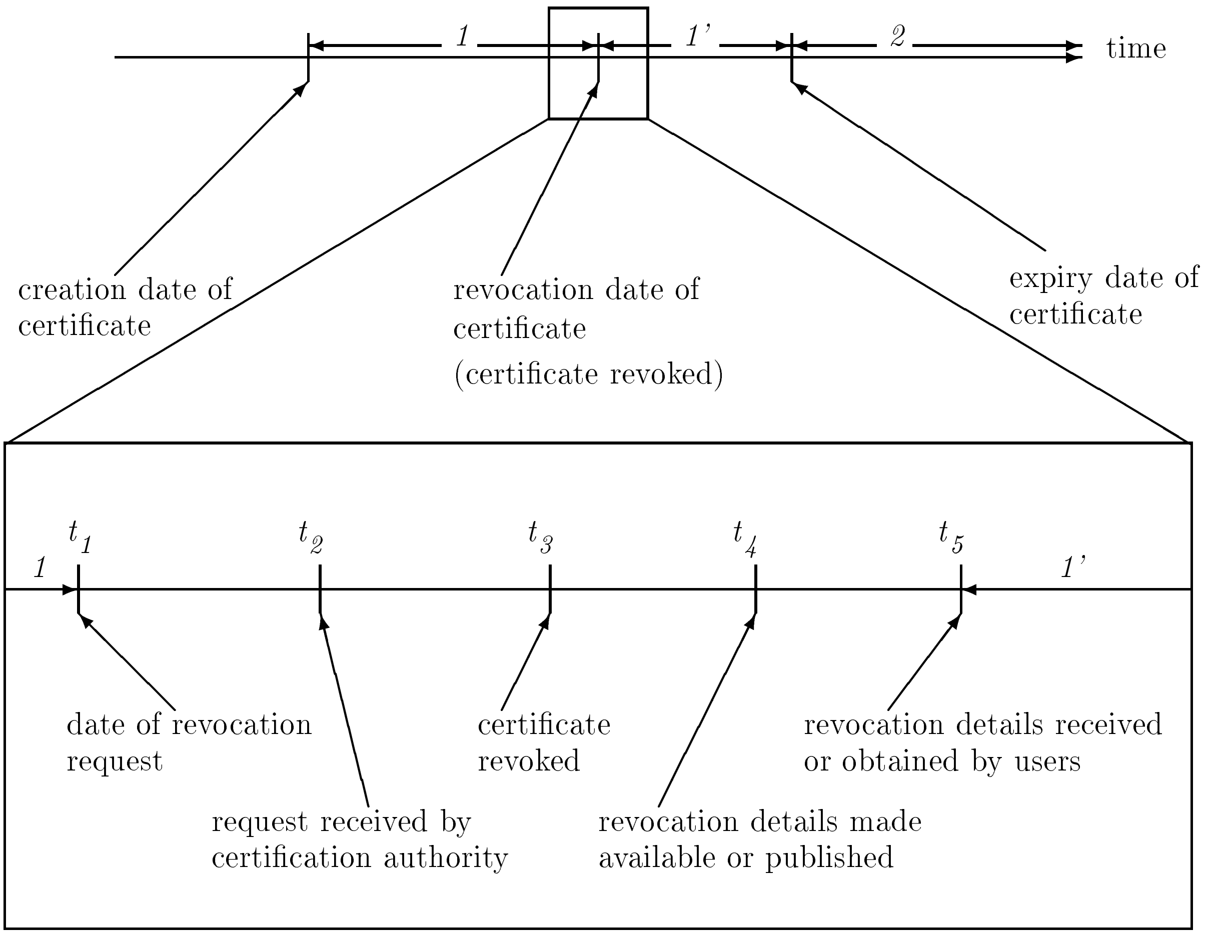
1
1'
2
time
creation date of certificate
revocation date of certificate (certificate revoked)
expiry date of certificate
t_1
t_2
t_3
t_4
t_5
1
1'
date of revocation request
request received by certification authority
certificate revoked
revocation details made available or published
revocation details received or obtained by users

Figure 5.10: Signing periods for a revoked certificate. Time period 1 indicates the period of time prior to the certificate's revocation and expiry. Time period $1'$ indicates the period of time subsequent to the certificate's revocation but prior to its expiry. Time period 2 indicates the period of time after which the certificate has expired. The bottom portion of the figure depicts more detailed steps involved in an actual certificate revocation.

Merkle [Mer80, Mer82] talks at length on the adjudication of signatures by an impartial third party. His discussions deal more with the *correctness* of the particular signature itself, with respect to the signature algorithm used, rather than whether or not the signature was signed during the operational or validity period of the verification certificate. In what follows, we present the requirements necessary for adjudicating such disputes regarding certificate-based digital signatures.

Evidence. The evidence required for an adjudication regarding the status of a signature is similar to what would be required by any verifying party, and includes:

1. *(Verification Evidence)* the purported evidence indicated in Table 5.2;

2. *(Trust Anchor)* In order to determine the validity of the purported originating user's verification certificate, the judge must be able to determine the truthfulness of the predicate $certTrust_J(CA_i\{CA_j\}, cert_u, t)$ (see Definition 5.12) therefore requiring that the judge is given a trust anchor $CA_i\{CA_j\}$ from which this trust can be determined.

Adjudication. The function of the adjudicator is to subsequently perform validations of the signature, given the evidence from u or from v. This is accomplished by computing and outputting the result $validSig_J(m, c, s, t, cert_u, CA_i\{CA_j\})$ (see Definition 5.13) using Protocol DS1 with trust anchor $CA_i\{CA_j\}$.

5.4 Notarizing Digital Signatures

The verification of signatures described in Section 5.3 possess some potentially undesireable features:

1. *Storage.* Subsequent to the expiry of a certificate, various certificate information may need to be stored long-term to allow for the continued verification and possible adjudication of signatures. The period of time might be exceedingly long given the nature of certain signatures, e.g., large monetary contracts or wills.

2. *Verifier Participation.* Signatures are produced once yet may be verified multiple times. This requires multiple requests for information verifying the same signature. This is lessened if cached/stored once obtained but there may be new verifiers as well.

One possible improvement is to have the signature originator provide more corroborating evidence to accompany a signature. This can be achieved, for example, by having the originator obtain the necessary evidence (e.g., most recent CRL) and send it along with the signature. However, this increases the size of the information accompanying a signature and may provide a verifier with unnecessary information, e.g., revocation information for certificates besides that of the originator. As well, this would require a signature originator to search for and obtain the necessary evidence for particular recipients. Even further is the consideration, that like the certificate of a user, this evidence may also become less trustworthy over time.

The time stamping of the signature provides some corroborative evidence and turns out to be necessary in order to allow for consistent signature validations subsequent to the expiry of the verification certificate (cf. Section 5.3.1). As described here, the *notarization* of a signature provides for an enhancement of a time stamp in which trusted third-party corroborative evidence pertaining to the state of the aliveness of the signature originator's verification certificate is also delivered. This notarization provides a trusted attestation of the status of submitted evidence with reference links to stored information, thereby providing corroboration in addition to or in lieu of long-term, stored evidence. Responsibility of verifying pertinent certificate information is shifted from a signature verifier to a notary authority.

Appendix A provides a historical review of the concept of notarization including a review of the (physical) notary public as well as a *digital notary.* In Section 5.4.1, we present a general definition of notarization. In Section 5.4.2, a protocol for the notarization of digital signatures is presented.

5.4.1 Notarization: Trusted Corroboration

In this subsection, we present Definition 5.14 (refined from Menezes *et al.* [MvOV97, page 550]) specifying the function of a general *notary* N and subsequently present Definition 5.15 which specifies the role of a *digital signature notary* DSN. We proceed to identify a list of statements for which a DSN may attest to the truth of. A notarization protocol of Merkle [Mer80, Mer82] is subsequently reviewed.

Definition 5.14 A *notary* N is a trusted third party whose role is to attest to the existence and/or truth of any statement (over which it is trusted on or granted jurisdiction over) at a given point in time thereby imparting authenticity to the statement. The *notarization* of a statement refers to the provision of an authentic attestation by a notary.

In particular, this attestation may be performed as a notarization whereby N returns a signed statement, wherein the statement contains explicit attestations to the truth of the statements presented by the submitting user. We examine this notarization more thoroughly, with particular emphasis on the notarization of purportedly valid digital signatures.

Definition 5.14 provides an open-ended defintion regarding the statements over which a notary can attest to the truth or establish the existence of. Our particular purpose (in this section) is to notarize statements regarding the validity of a digital signature. Referring to (5.1) of Definition 5.13, there are a number of functions, whose truthfulness must be successfully determined in order to have a *valid* signature. As demonstrated by Protocol DS1 (see Section 5.3.2), determining the truthfulness of some of these functions requires obtaining additional information, in addition to the signature itself. For example, for a signature c purportedly constructed by user u and time stamped at time t, a verifier must determine the truth of the statement $revoked(cert_u, t)$. This requires obtaining the proper revocation lists from the appropriate CAs. A DSN can therefore determine and attest to the truth of (see (5.1) from Definition 5.13) various statements, including those listed in Table 5.4.

1. $ver_u(m, c)$: Given a verification key, puportedly belonging to user u, a DSN can determine and attest to the mathematical correctness of the signature $c = sig_u(m)$ for the purported message m;

2. $ver_T(s, t, c)$: Given a verification key, purportedly belonging to the time stamp authority T, a DSN can determine and attest to the mathematical correctness of the signature $s = sig_T(c, t)$ for the purported signature and time, c and t, and if successful, use t as the point in time for determining a certificate's status;

3. $expired(cert_u, t)$: Given a public key certificate $cert_u$, purportedly belonging to user u, and a time t, a DSN can determine and attest to whether or not $cert_u$ was valid at time t;

4. $revoked(cert_u, t)$: Given a public key certificate $cert_u$, purportedly belonging to user u, and a time t, a DSN can determine and attest to whether or not $cert_u$ was operational at time t;

5. $certTrust_v(CA_i\{CA_{i'}\}, cert_u, t)$: Given a public key certificate $cert_u$, a time t and a trusted certificate (trust anchor) $CA_i\{CA_{i'}\}$, a DSN can determine and attest to whether or not, as of time t, there is (or was) a certificate chain (see Definition 5.8) $(CA_i, CA_{i'}, \ldots, u)$.

Table 5.4: Statements (cf. Definition 5.13) for Which a Digital Signature Notary (see Definition 5.15) can Attest to the Truthfulness.

Definition 5.15 A *digital signature notary* DSN is a trusted third party whose role is to attest to the truth of the functions from (5.1) (see Definition 5.13), which may involve establishing the existence of supporting evidence. The *notarization* of these statements refers to the provision of an authentic attestation to the truth of the statements by a DSN.

As an example of notarization, consider Protocol NT1, which is slightly modified from the protocol as presented by Merkle [Mer80, Mer82]. The protocol uses a time stamp authority (T) (what Merkle refers to as a *time-keeper*) whose purpose is to digitally time stamp submitted information, and a CA (what Merkle refers to as a *central authority*) to attest to the validity of a certificate at a given point in time; like an online certificate status check (see Section 5.2.3). More specifically, the CA is used to attest to the fact that $cert_u$ is not currently expired or revoked.

As specified in Protocol NT1, the CA is acting as a DSN by attesting to the truth of the state of u's certificate $cert_u$, i.e., its expiry or revocation. In Section 5.4.2, we present Protocol NT2, in which a DSN determines and attests to the truth values of all functions specified in (5.1) of Definition 5.13.

5.4.2 Notarizing Digital Signatures

The relevance of notarizing digital signatures is evident from the discussion in Section 5.3.1 noting that the verification of a digital signature requires the maintenance of evidence at a given point in time: the time at which the signature was time stamped. The main difference with the time stamping solution of Section 5.3 is that a notarization by a DSN (see Definition 5.15) provides a self-contained package of corroborative evidence, allowing one to validate a digital signature numerous times without requiring the verifier to obtain additional information.

The main purposes of the notarization of digital signatures is to

1. *centralize the validation of a digital signature* so as to limit the amount of corroborative evidence gathering that may be performed by the signature verifier,

2. *reduce the amount and type of information stored* (as described in Section 5.3.1) for subsequent validations of signatures, and

Protocol NT1 Signature Notarization by Verifier [Mer80, Mer82].

Description: The recipient v of signature c uses a CA as a DSN (see Definition 5.15) as part of a larger protocol in which evidence regarding the status of the purported signature originator's verification certificate is collected by v. The end result of the protocol is v's decision as to whether or not to accept or reject a signature based, in part, on the information collected regarding the status of the originator's certificate.

$\boxed{\text{Signature Transmission}}$

Input: message m
Output: signature $y = sig_u(m)$ and purported originator certificate $cert_u$

```
1: For a message $m$, $u$ computes $y = sig_u(m)$ and sends $(y, cert_u)$ to $v$.
```

$\boxed{\text{Time Stamping of Signature}}$

Input: signature $y = sig_u(m)$
Output: time stamp $s = sig_T(y, t)$

```
1: $v$ sends $y$ to $T$.
2: $T$ returns the time stamp $s = sig_T(y, t)$ where $t$ is typically the time of receipt of $y$ by $T$. (This same technique is used by Protocol AB1 in Section 2.4.1.)
```

$\boxed{\text{Notarization of Certificate Revocation \& Expiry Status}}$

Input: u's purported certificate $cert_u$
Output: whether $cert_u$ is currently expired or revoked

```
1: $v$ requests a validity check from the $CA$ with regards to the current status of $u$'s certificate by sending $cert_u$ to the $CA$.
2: Upon receiving $cert_u$ at time $t'$, the $CA$ determines the truth value of $expired(cert_u, t')$ and $revoked(cert_u, t')$ as performed by a signature verifier in steps 3 and 4 of Protocol DS1. If true, then the $CA$ notarizes this successful result by returning the signature $sig_{CA}$("$u$'s certificate $cert_u$ is still valid and operational at time $t'$").
```

$\boxed{\text{Signature Validation}}$

Input: time t of stamping of signature y and notarization of $cert_u$
Output: determination of whether or not $y = sig_u(m)$ is a valid signature

```
1: If $v$ receives a positive response from the CA and $t \prec t'$, then $v$ completes steps 1 and 5 of Protocol DS1 using the time $t$ (obtained above from $T$) as the time of stamping, and if successful, accepts $u$'s signature $y$ over $m$.
```

3. *provide trusted corroborative evidence* for signature verification in lieu of or in addition to stored evidence, with inclusion of submitted information and reference pointers to stored evidence information.

There are several attestations that can be made by a DSN during the notarization of a digital signature, each of which vouch for the existence and/or truth of statement(s) (relevant to the authenticity of the digital signature) at a specific time. A list of such statements was given in Table 5.4. The input to the DSN can vary depending on the attestation required by the requestor. The process of notarization is presented as Protocol NT2.

Using Protocol NT2

Since the notary can attest to a variety of requests made by a signature originator or verifier, Protocol NT2 has a number of potential uses. Below, we present several such uses, keeping in mind that Protocol NT2 may have other applications, and even further, can be enhanced so as to satisfy other notarization requirements.

Notarizing a signature. By submitting $(m, c, -, -, cert_u, CA_k\{CA_k\})$ at time t, where $cert_u = CA_k\{u\}$, one can obtain the notarization $sig_{DSN}(S)$ where S is the statement

> The signature c for the message m, which existed at time t, was verified for mathematical correctness using the certificate $cert_u$. The certificate $cert_u$ issued by CA_k was not expired nor revoked as of time t where the latter was verified using the CRL numbered rn and dated time t'. At time t, the certificate chain (CA_k, CA_k, u) existed and was mathematically correct, as verified using the respective cross-certificates numbered (cn_k). Each cross-certificate was valid and operational according to the respective ARLs (an_k) with respective times (t_k).

Notarizing a time stamped signature. Submitting $(m, c, s, t, cert_u, CA_k\{CA_k\})$, where $cert_u = CA_k\{u\}$, one can obtain the notarization $sig_{DSN}(S)$ where S is the statement

Protocol NT2 Notarization of a Digital Signature.

Description: A digital signature notary DSN attests to the truth of selected functions from (5.1) of Definition 5.13, based on the input from the requestor (see sample inputs starting on 159).

Note: Let $\mathcal{A}$ represent Protocol DS1 of Section 5.3.2.

Require: The DSN is assumed to have a trusted clock, and each user is assumed to have a trusted copy of the DSN's self-signed verification certificate. The expiry or revocation this certificate is beyond the scope of this thesis.

Input: A message m, purported signature c, purported time stamp s, time t, purported certificate $cert_u$ of the producer of c and a trust anchor $CA_i\{CA_j\}$. Not all variables must be input, as specified for each step below, though a trust anchor is required in any case.

Output: A signed attestation to statements established by the DSN based on the input variables to the protocol.

```
1: If a time $t$ and $s$ are input (necessarily with $c$), perform step 2 of $\mathcal{A}$. If a time $t$ is input without $s$ (i.e., without corroboration of the correctness of the time $t$),[a] then skip to step 3 if $m$ and $c$ are not input, else goto step 5 with failure. If a time $t$ is not input, then $t$ is assigned the time of receipt of the input.
2: If $m$ and $c$ are input (necessarily with $cert_u$), perform step 1 of $\mathcal{A}$. If successful then assign the statement $S_1$ = "The signature $c$ for the message $m$, which existed at time $t$, was verified for mathematical correctness using the certificate $cert_u$."
3: If $cert_u$ is input, then perform steps 3 and 4 of $\mathcal{A}$. If both are successful, then assign the statement[b] $S_2$ = "The certificate $cert_u$, issued by $CA_k$ was not expired nor revoked as of time $t$ where the latter was verified using the CRL numbered $rn$ and dated time $t'$."
4: For $CA_i\{CA_j\}$ (necessarily input with $cert_u$), execute step 5 of $\mathcal{A}$ to find *a* certificate chain[c] from $CA_i\{CA_j\}$ to $CA_k\{u\} = cert_u$. If successful then assign the statement $S_3$ = "At time $t$, the certificate chain $(CA_i, CA_j, \ldots, CA_k, u)$ existed and was mathematically correct, as verified using the respective cross-certificates numbered $(cn_i, cn_j, \ldots, cn_k)$. Each cross-certificate was valid and operational according to the respective ARLs $(an_i, \ldots, an_k)$ with respective times $(t_i, \ldots, t_k)$."
5: If each step was successful, the $DSN$ produces a signature over the statements as $sig_{DSN}(S_1, S_2, S_3)$ (depending on the input variables presented, not all of these statements will be assigned). If any step fails, then the $DSN$ outputs $sig_{DSN}(input, \text{"failure"})$ where $input$ represents the set of input variables.
```

[a]In this case, the time t has been input as a request for notarization regarding the status of $cert_u$ at time t.

[b]Notice the similarity of this step to the notarization performed in Protocol NT1.

[c]An optional enhancement to the present protocol would involve requesting for more than one certificate chain (if they existed) to be output in S_3.

> The signature c for the message m, which existed at time t, was verified for mathematical correctness using the certificate $cert_u$. The certificate $cert_u$ issued by CA_k was not expired nor revoked as of time t where the latter was verified using the CRL numbered rn and dated time t'. At time t, the certificate chain (CA_k, CA_k, u) existed and was mathematically correct, as verified using the respective cross-certificates numbered (cn_k). Each cross-certificate was valid and operational according to the respective ARLs (an_k) with respective times (t_k).

Notice that the reply is identical to the previous notarization of a signature that was not time stamped. The difference is that for the time stamped signature, validations regarding the revocation status of $cert_u$ were performed as of time t, the time contained in the time stamp, rather than the time of receipt of the request by the DSN.

Notarizing a signature for known verifiers. Submitting $(m, c, s, t, cert_u, CA_i\{CA_j\})$ with certificate $CA_k\{u\} = cert_u$ and knowledge of a trust anchor for an intended recipient of a time stamped signature from user u results in the notarization $sig_{DSN}(S)$ where S is the statement

> The signature c for the message m, which existed at time t, was verified for mathematical correctness using the certificate $cert_u$. The certificate $cert_u$ issued by CA_k was not expired nor revoked as of time t where the latter was verified using the CRL numbered rn and dated time t'. At time t, the certificate chain $(CA_i, CA_j, \ldots, CA_k, u)$ existed and was mathematically correct, as verified using the respective cross-certificates numbered $(cn_i, cn_j, \ldots, cn_k)$. Each cross-certificate was valid and operational according to the respective ARLs $(an_i, \ldots, an_k)$ with respective times $(t_i, \ldots, t_k)$.

This input may also be submitted to a DSN by a verifier of a time stamped signature. If s and t are not included above, then t is taken as the time of receipt of the request by the DSN. Notice that, depending on the organization of the certification authorities, the choice of trust anchor can be chosen so as to allow acceptance of the notarization for a variably large group of users. For example, if the issuer of $cert_u$ is CA_2 and u wants to send a notarized signature to all users that possess certificates issued by CA_1 then the trust anchor $CA_1\{CA_2\}$ allows trust to be obtained in the signature c by all users possessing the self-signed certificate of CA_1.

Determining certificate status. Submitting $(-,-,-,t,cert_u,CA_k\{CA_k\})$, where $cert_u = CA_k\{u\}$, allows one to receive a notarization regarding the status of $cert_u$ as of time t as $sig_{DSN}(S)$ where S is the statement

> The certificate $cert_u$ issued by CA_k was not expired nor revoked as of time t where the latter was verified using the CRL numbered rn and dated time t'. At time t, the certificate chain (CA_k, CA_k, u) existed and was mathematically correct, as verified using the respective cross-certificates numbered (cn_k). Each cross-certificate was valid and operational according to the respective ARLs (an_k) with respective times (t_k).

If t is not input, then t is taken as the time of receipt of the request by the DSN.

Determining certificate chain existence. Submitting $(-,-,-,t,cert_u,CA_i\{CA_j\})$ allows one to receive a notarization regarding the existence of a certificate chain $(CA_i, CA_j, \ldots, CA_k, u)$ by receiving the notarization $sig_{DSN}(S)$ where S is the statement

> The certificate $cert_u$, issued by CA_k was not expired nor revoked as of time t where the latter was verified using the CRL numbered rn and dated time t'. At time t, the certificate chain $(CA_i, CA_j, \ldots, CA_k, u)$ existed and was mathematically correct, as verified using the respective cross-certificates numbered $(cn_i, cn_j, \ldots, cn_k)$. Each cross-certificate was valid and operational according to the respective ARLs $(an_i, \ldots, an_k)$ with respective times $(t_i, \ldots, t_k)$.

If t is not input, then t is taken as the time of receipt of the request by the DSN.

5.5 Digital Signature Renewal

The prevention of signature forgery relies, in part, on the computational infeasibility associated with an attack that would forge a signature subsequent to, for example, exhaustively trying all signature keys. The choice of parameters for signature algorithms may be chosen large enough so as to discourage (and prevent) an attacker from attempting such malicious acts yet small enough so that the computational complexity of computing a legitimate signature is reasonably efficient. However, increases in computational power (e.g., faster computer chips and efficient distribution of programs over increasingly large networks) imply that the parameters chosen at one point in time may not provide the same level of security at subsequent times.

5.5.1 Definitions and Motivation

In this subsection, we motivate and define concepts related to the renewal of digital signatures. We first distinguish between legitimate and fraudently produced signatures with the following definitions.

Definition 5.16 A digital signature $c = sig_u(m)$ (see Definition 2.11) over data m is *legitimate* with respect to a user u (i.e., the user named in the corresponding verification certificate) if u was aware of or participated in the construction of c. Awareness includes knowledge that a signature for m is currently being produced in the absence of disapproval of this action. Participation includes the actions of willfully executing the software that produces the signature.

We use the term *legitimate owner (user)* to identify the entity for whom the public key was certified.

Definition 5.17 A digital signature $c = sig_u(m)$ is *forged* if it is not legitimate with respect to user u.

There are numerous possibilities for compromising a signature system, resulting in the subsequent forgery of signatures. A partial list of such attacks is given in Table 5.5.

1. *Algorithmic Attacks.* The signature algorithm itself (e.g., RSA) has succumbed to mathematical or cryptanalytic attack.

2. *Implementation Failures.* A particular signature algorithm has been poorly implemented. We include here the possibility of weak keys being chosen, a poor random number generator being used, or the private key not being adequately protected. As specific examples, note the attack on ElGamal signatures [Ble96], the timing analysis attack [Koc96] and differential power analysis attacks [KJJ98].

3. *Insider or Physical Attacks.* This includes attacks whereby the private key is read from temporary memory (in which it is stored while being used). As well, an attacker might observe as a user enters the password used to provide access to their private key or to decrypt keying information. This can also include a *social engineering* attack whereby a user may be fooled into giving up a password or key, or a system administrator may be bribed into revealing it.

4. *Brute-force attacks.* An attack whereby the password (used to encrypt keying material) or private key itself is guessed. Schemes with low-entropy passwords are most susceptible to such an attack.

Table 5.5: Attacks to a Signature Scheme. The goal of the attacks is to compromise either the signature algorithm or the private key(s) of a signing user(s). The end result is an ability to forge signatures.

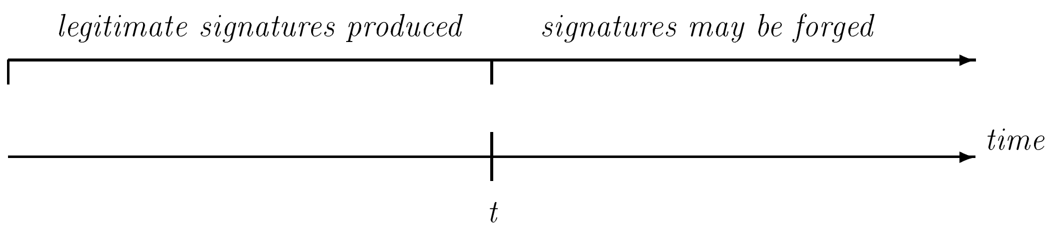

Figure 5.11: Periods of Legitimate and Forged Signature Production. Time t indicates the point of a successful attack to the signature scheme of one or more users. Subsequent to time t, signatures may be forged in the names of those users whose signature production capabilities have been successfully attacked.

Consider also more futuristic attacks involving quantum computing [GC98, Sho94] and possible improvements in the efficiency of factoring [Pom90].

Definition 5.18 A signature scheme is *compromised* if it allows the production of forged signatures.

In Figure 5.11, the partitioning of legitimate and forged signatures is depicted, relative to the time t of a *successful signature attack*. We make the following assumptions with regard to actions performed relative to the time t:

1. only legitimate signatures can be produced before time t;

2. signatures may be forged after time t.

Definition 5.19 A *digital signature renewal process* provides for the *renewal* or *extension* of the message authenticity of a legitimate signature $c = sig_u(m)$ for message m by ensuring that, subsequent to a compromise at time t (see Definition 5.18) of the signature scheme (see Definition 2.11) of which $sig_u()$ is a component,

1. signatures legitimately produced with $sig_u()$ prior to time t are successfully validated after time t, and

2. forged signatures produced with $sig_u()$ at or after time t are successfully invalidated after time t.

5.5.2 Anticipation and Redundancy

The attacks from Table 5.5 can be generally classified as being either *predictable* or *unpredictable*. If a key compromise or equivalent attack is predictable, then digital signatures produced with the soon-to-be-attackable signature scheme can be renewed prior to an attack so as to extend their message authenticity beyond the point in time when the original signature mechanism succumbs to attack. In subsection 5.5.3, we describe the process of renewal.

For those cases in which an attack is not anticipated nor predictable, there may be a number of forgeries produced before the attack has been detected. Once detected, the corresponding certificate can be revoked (see Section 5.2.3). Yet there will still be some question regarding the authenticity of signatures that may have been produced subsequent to an attack yet prior to revocation. In Chapter 6, we discuss techniques for detecting and recovering from an attack. In this subsection, we discuss techniques for anticipating attacks to signature schemes, by decreasing the possibility that a single attack will permit signature forgeries.

One way to anticipate an attack and allow for subsequent renewal is to provide redundancy in the signature production mechanism. Some methods for providing redundancy are:

1. *Multiple Signature Keys.* The use of multiple keys for the production of a digital signature allows one to extend the lifetime of previously constructed signatures (beyond that of a signature scheme in which a single signature key compromise is typically sufficient for forgery production) so long as more than one attack is required to compromise all keys and that attacks are detectable at least before all keys are compromised. This solution is discussed further in Section 6.4;

2. *Multiple Hash Functions.* Consider the technique used by Haber and Stornetta in Protocol HY1 (see Section 4.5). The signature for a message m is computed, not as the traditional signature over a single hash of m (i.e., $sig_u(y)$ where $y = h(m)$), but rather, over two hashes (i.e., $sig_u(y)$ where $y = h_1(m)||h_2(m)$). In the case that one of the hash functions h_1 or h_2 succumbs to an unanticipated attack, the authenticity of the original signature remains and can subsequently

be renewed prior to a possible attack to the other hash function.

Therefore, by constructing the signature algorithm in such a way that multiple, indenpendent attacks are required, the detection of attacks allows one to anticipate and renew signatures.

5.5.3 Signature Renewal

In Section 5.3.1 (see Figure 5.8) it was observed (as discussed by Haber and Stornetta [HS91]) how the time stamping of a signature served to extend the lifetime of the signature by allowing verification of the signature past the point in time at which the corresponding signature verification certificate had expired. In this subsection, we examine how time stamping is also useful for extending the validity period of the signature when attacks to the signature algorithm can be detected.

Consider the signature $c = sig_u(m)$ produced by user u for the message m and the absolute time stamp $s = sig'_T(c,t)$ produced by time stamp authority T at time t, using Protocol HY2 (see Section 4.5) where $sig'()$ and $sig()$ represent different signature algorithms used respectively by T and u. Notice that, the compromise of u's signature key at time t', for example, prevents an attacker from producing a valid, time stamped signature so long as the certificate of u is revoked prior to the production of a forged signature. In this way, the time of the time stamp for the forged signature would be later than the time of revocation implying that verification of the status of u's certificate before accepting a signature, would fail, i.e., Protocol DS1 (of Section 5.3.2) would output a failed result – an invalid signature. The time stamping of the signature has anticipated the compromise of u's key. (In Chapter 6, we consider the possibility in which a key compromise is not detected.)

The technique for using time stamping as a method of renewal in this manner was described by Haber and Stornetta [HS91]. However, consider the following problem with this technique as recognized by Bayer, Haber and Stornetta [BHS93]. Let us first characterize two methods by which a signature might be forged (subsequent to a successful attack to a signature scheme):

1. by creating a new signature, independent of any previous legitimate signatures;

2. by altering a legitimate signature.

Suppose, for example, that the breach of security regarding sig_u involved the discovery of a computationally feasible method for finding hash function collisions, for the hash function h used in computing c. Recall that in Section 2.1.2, it was indicated that for reasons of efficiency, signatures were actually computed as $c = sig_u(h(m))$ for a message m, using a hash function h. Therefore, the resultant time stamp s would be computed as

$$s = sig'_T(h'(c,t)) = sig'_T(h'(sig_u(h(m)),t)).$$

Notice now that if h were to become insecure (i.e., one were able to find collisions for h), u could effectively absolutely back stamp (cf. Definition 4.10) a message m' so long as $h(m) = h(m')$.

Remark 5.5 *The time stamping of a signature is not sufficient for the renewal of a signature in the case of it becoming computationally feasible to find collisions for the once collision-resistant hash function h, used during signature production.*

Protocol RN1 Digital Signature Renewal with Time Stamping [BHS93].

Description: This protocol provides for a time stamp of a message-signature pair, differing from the time stamping of only the signature as described in Section 5.3 (thereby allowing signature renewal even if the hash function h eventually becomes susceptible to the discovery of hash collisions).
Assumption: Signature protocol independence between sig and sig' whereby sig may be susceptible to attack after time t, while sig' is not compromised.
Note: Inputs are hashed before signing.
Note: The time t' of stamping must be such that $t' \prec t$ for the message authenticity of the original signature c to be renewed.
Input: message m and signature $c = sig_u(m)$
Output: time stamp $s = sig'_T((m,c),t')$ for time t'

```
1: $u$ sends the pair $(m,c)$ to the time stamp authority $T$ where $c = sig_u(m)$.
2: $T$ returns the time stamp $s = sig'_T((m,c),t')$ to $u$ where $t'$ is the time of receipt
   of the pair $(m,c)$.
```

Consider the alternative of Protocol RN1 as presented by Bayer *et al.* [BHS93]. As noted (though not mentioned by the authors), Protocol RN1 should attempt to ensure the independence of the signature algorithms (including hash functions) used by T and other users, so that the current role of the time stamp is met, i.e., extension of the lifetime of the signature in case of attacks to the signature production performed by u. Proposition 5.1 demonstrates that even a weakness of the signature producer's hash function permits the successful renewal of the signature.

Proposition 5.1 *A computationally feasible method for finding collisions for the hash function $h()$ is not sufficient for fraudulently claiming the time stamping of the message-signature pair (m', c) in place of the legitimately time stamped message-signature pair (m, c) for $m \neq m'$ in Protocol RN1.*

Proof Suppose there were such a pair of messages $m \neq m'$, then for the time stamp computation:

$$\begin{aligned} s &= sig'_T(h'((m,c),t')) \\ &= sig'_T(h'((m, sig_u(h(m))), t)) \qquad (5.2) \\ &= sig'_T(h'((m', sig_u(h(m'))), t)). \qquad (5.3) \end{aligned}$$

Although it may be that $h(m) = h(m')$ for $m \neq m'$, the equality of (5.2) and (5.3) implies that $h'((m, sig_u(h(m))), t) = h'((m', sig_u(h(m'))), t)$ which would imply that a collision has also been found for the hash function $h'()$, a contradiction to the assumption of Protocol RN1. ■

Remark 5.6 *The time stamping of a message-signature pair extends the lifetime of the original signature even in the case of a hash function eventually being susceptible to a computational method for finding hash collisions, subsequent to the time stamping.*

Renewal Using Notarization

Time stamping the pair $(m, c = sig_u(m))$ as opposed to only c allows for the authenticity of c to be extended in the event of an anticipated compromise to the signature scheme used to produce c. Since the time stamper T is not concerned with the form of the input and hence, blindly time stamps the input (m, c) as a single input, it appears to make little difference as to whether c or (m, c) are time stamped. Indeed, even though m may be included in the time stamping input, the size of the time stamp remains the same (since the input is hashed first) as does the size of the user's storage since u would be required to maintain the storage of m in any case. However, note that by submitting (m, c) the privacy of m is not maintained, nor are there the bandwidth efficiencies as offered by the submission of a hashed message (see Section 2.2). Therefore, Protocol RN1 is undesirable given such requirements.

Alternatively, one can time stamp c and subsequently *notarize* (m, c) when a signature scheme compromise is anticipated. More specifically, suppose that for the message m, and signature $c = sig_u(m)$, user u first obtains the time stamp $s = sig'_T(c, t')$ at time t'. Further suppose that the signature scheme $sig_u()$ is compromised at time $t \succ t'$. Prior to time t, say time t'' where $t' \prec t'' \prec t$, the time stamped signature can be notarized. This can be accomplished as in the particular example from page 159 whereby $(m, c, s, t, cert_u, CA_k\{CA_k\})$ is submitted to the digital signature notary DSN in Protocol NT2. The output of this protocol corroborates and attests that m was indeed signed at time t', producing the signature c.

This technique has the advantage (over Protocol RN1) that since not all signatures require that their lifetime extend beyond the anticipated lifetime of their signature algorithms, not all require that both the message and corresponding signature are initially time stamped.

Renewing Time Stamps

Just as a time stamp extends the lifetime of a digital signature in anticipation of an attack in which signatures may be forged, the time stamp itself requires renewal (in anticipation of the compromise of the signature scheme used for the production of the time stamp). The time stamp also has a fixed lifetime that may not sufficiently extend the lifetime of the original signature. As observed by Bayer *et al.* [BHS93] Protocol RN1 can also be used with m, signature $c = sig_u(m)$, and time stamped signature $s = sig'_T((m, c), t)$ where the required input would be $((m, c, t'), s)$.

Recording the Time of Compromise

Just as the time of stamping of a signature is compared to the time of revocation of a certificate (e.g., from a CRL) or expiry (as contained within the certificate), the process of renewal relies on the ability to compare the time of stamping to the time of compromise. For this reason, the time of compromise (see Definition 5.18) of a user signature scheme must be authentically recorded, for example by an issuing CA.

Initiation of Renewal

The time stamping of a signature $c = sig_u(m)$ renews the signature c beyond the time of compromise of sig_u, except in the case that the compromise of sig_u includes the discovery of a computationally feasible method for determining hash collisions for the hash h used in the computation of c. If such a compromise is not a concern, or if the required "message authenticity lifetime" for c is relatively short, then the alternative stamping of the message-signature pair (of Protocol RN1) or subsequent notarization, may not be required.

The renewal of a signature or time stamp will typically be initiated by a verifying party, requiring a long lifetime for a particular signature(s). This renewal can be accomplished by repeated, periodic renewal of the digital signature and subsequent time stamps and extends the message authenticity of the original digital signature so long as renewal is performed prior to compromises. In Chapter 6, we consider the possibility of undetected compromises.

Chapter 6

Addressing the Problem of Undetected Signature Key Compromise

The digital signature is the digital counterpart to the physical, handwritten signature. Each permits authorization with respect to the corresponding named individual. A handwritten signature permits authorizations corresponding to the particular name that is being signed. A digital signature private key may be used, together with additional controls, to allow authorizations with respect to the name provided in the corresponding public key certificate. The compromise of the private key results in a loss of exclusive control over associated privileges, and allows impersonation.

Once it is known that a key has been compromised (i.e., once a compromise is detected), suitable recovery actions may be taken to prevent further damage. For example, various means for key revocation (see Section 5.2.3) allow one to prevent future acceptance of forged signatures time stamped later than the date of revocation (see Section 5.3.1). However, revocation can only be performed once a compromise has been reported (and hence detected). The ability to distinguish forged signatures from legitimate ones requires knowledge of *when* the compromise occurred.

To date, the problem of protecting against the forgery of signatures resulting from an undetected compromise of a user's signature private key has not even been considered in the open literature, let alone solved in any way. In this chapter, we introduce and present a first study of the problem of *undetected key compromise*. Moreover, and perhaps counter-intuitively, solutions are provided which prevent even an attacker who has obtained or deduced (by any means, for any signature algorithm) a user's signature private key, from being able to forge signatures that would be accepted by an unsuspecting recipient.

Chapter Outline

In Section 6.1, we define events related to a key compromise and the detection of the compromise. As well, the inadequacy of revocation and time stamping for dealing with the problem of an undetected compromise is discussed. Section 6.2 reviews and examines solutions useful for reducing the damage subsequent to a signature key compromise. In Section 6.3, we overview new solutions in which independent means are used to identify the legitimate signing user, and position the work relative to the current literature and practice as well as to the content of this thesis. In Section 6.4, we elaborate on a first solution in which a secondary (independent) identification is used for enhanced protection against an undetected key compromise. In Section 6.5 techniques are presented which use a secondary (independent) synchronization method to allow the legitimate signer to detect when forged signatures have been produced. Combining a periodic check-in by the legitimate user with a cooling-off period for the acceptance of signatures allows for the detection to be enhanced so that forged signatures will not be accepted by a recipient. This is discussed in Section 6.6.

6.1 Definitions, Assumptions and Motivation

In this section, the problem of undetected key compromise is introduced and the need for new solutions is motivated.

Definition 6.1 A *key compromise* has occurred if the signature (private) key sig_u (see Definition 2.11), or equivalent key, is possessed by an individual other than the legitimate signature key owner u, and there exists a potential for the misuse of sig_u by this other entity (e.g., the forging of signatures - see Definition 5.17).

A variety of possible attacks to both the privacy of a signature key and security of signature algorithms were given in Table 5.5 of Section 5.5. Although access controls are necessary, in many cases they might not be sufficient for protection of the signature key. It is important to recognize that despite various controls and protections, some keying material may eventually be compromised.

Definition 6.2 An *undetected key compromise* is a key compromise for which the legitimate key owner (see Definition 5.16) is not aware of the possession of sig_u by another individual.

If a private key compromise or equivalent attack is detected by u, the corresponding public key certificate can be revoked, and a new key pair can be introduced. Throughout this chapter, we use the term 'key compromise' to indicate an attacker's ability to impersonate u using an ability to forge signatures.

6.1.1 Compromise Detection

In discovering or becoming aware of (possibly only suspected) attacks, it is important to consider the storage of the signature key. For example, the key might not be stored on any physical medium, but rather computed as some function of a memorized password which appears in a computer system (e.g., RAM) for only a very short period of time.[1] We refer to this as an *ephemeral token.* The key may be stored on a user's disk. This is referred as a *software token.* As well, the key may be stored on a smartcard, i.e., *hardware token.* Alternatively, the key may be physically recorded (e.g., on a piece of paper) and is thus referred to as a *physical token.* A physical token differs from a hardware token in that the former is easier to duplicate, while the latter, generally is not.

[1]For example, a password could be used as a seed for a (reproducible) deterministic process which generates the signature key.

Implemented correctly, a hardware token allows compromise to be easily detected, i.e., the user would recognize the missing token. However, it does not necessarily protect against an algorithmic attack or implementation failure (see Table 5.5), and care must be taken in their use [BDL97, KJJ98], e.g., if a weak random number generator were used. The techniques presented in this chapter provide protection even in the case of signature key compromise due to these failures. Additional methods for detecting a signature key compromise include:

1. detection of a physical attack to a user's computer system in which keying material is stored (this includes theft of a hardware token);
2. the legitimate user receives a signature(s) from some other user, which the legitimate user identifies as a forged signature; and
3. public announcement of a computationally feasible attack or protocol failure applicable to the digital signature algorithm or related components (e.g., hash function).

6.1.2 Events Related to a Key Compromise.

Consider the timeline of actions related to a signature key compromise as given in Figure 6.1 (similar to Figure 5.10 with the additional indication of a key compromise (or equivalent attack) as the cause for revocation). The compromise of u's key takes place at time t_0. The compromise may be suspected at time t_1 (the time of detection of the key compromise; u may or may not be aware of the precise time of its occurrence). u reports the compromise at time t_2 (for example, to the certification authority (CA) who issued the corresponding verification certificate) and this information is received by the CA at time t_3. Knowledge of the information is made available to users at time t_4, e.g., using CRLs (see Section 5.2.3). Note that some time may elapse between t_3 and t_4, e.g., if protocol dictates that compromises are published within 12 hours, as opposed to publishing after each revocation request is received. Knowledge of the key compromise is obtained by users as early as time t_5; different users may obtain

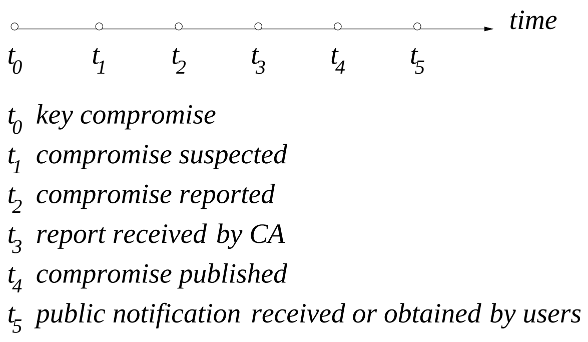

Figure 6.1: Timeline of events related to a key compromise. From time t_0 to t_1 is a period of *undetected key compromise*.

this information at different times. We have $t_0 \preceq t_1 \prec t_2 \preceq t_3 \prec t_4 \preceq t_5$ (see Definition 4.3).

6.1.3 Limits of Revocation

Even if existing certificate revocation techniques were used in response to a key compromise, they were not intended to handle the case of an undetected compromise since they rely on the compromise being reported, and hence detected. During the period of time starting at t_0 and ending prior to t_1 (see Figure 6.1), a period of *undetected key compromise* (see Definition 6.2), a number of messages may be signed, including both forged and legitimate signatures. In the worst case, a compromise may not be detected at all, thereby allowing signatures to be forged until the date of expiry of the corresponding public key certificate. Using current techniques, it is difficult to distinguish whether, for the case of disputed signatures (produced and time stamped during the period of undetected key compromise),

1. u did not actually sign the messages (i.e., an attacker did), or
2. u legitimately signed the messages and is attempting to repudiate the signatures, by either claiming
    (a) a signature private key compromise prior to any actual compromise, or
    (b) a compromise when in fact there was no actual key compromise.

Note that the revocation information may be the only evidence available to an adjudicator (see Section 5.3.2) asked to resolve if and when a key compromise may have occurred.[2] Thus, it may be reasonable to assume that signatures time stamped prior to the date of revocation are considered legitimate. However, this may place an unexpected burden or unfair penalty on the legitimate user in cases where a user's private signature key is indeed compromised without his/her knowledge. Indeed, u may be unable to pinpoint the exact time of the compromise. However, allowing u to repudiate signatures that may have already been accepted is equally unfair to the recipients of the signatures.

6.1.4 Time Stamping is Necessary but not Sufficient

The determination of whether a signed message is valid involves, among other things, a comparison of the time of stamping of the signature with events related to the status of the corresponding verification certificate (see Protocol DS1 of Section 5.3). The time of a certificate's revocation (or expiry) can be compared to the time of signing of messages to allow determination of whether a message was signed before or after a revocation. Such a procedure may not be sufficient in the case of an undetected key compromise. Let t_s be the time of stamping of a signature c. The apparent legitimacy of c (from the point of view of a signature verifier) given that $t_s \prec t_2$ (see Figure 6.1) may be unclear in the case that t_1, the claimed suspected time of compromise, is determined only by the legitimate user. For example, in an attempt to repudiate a legitimate signature, the legitimate signer may dishonestly report to the CA that $t_1 \prec t_s$. Indeed, until the compromise is detected, a number of legitimate and forged signatures may have been time stamped during the interval from t_0 to t_1, making it difficult to arbitrate a dispute regarding the legitimacy of such signatures. Thus, the time stamping of the signature is insufficient in this case. In the remainder of this chapter, we examine methods for dealing with this situation.

[2]In some cases, additional information may be available, for example *physical* evidence. However, we focus on solutions that do not rely on such evidence.

6.2 Dealing with Signature Key Compromise

In this section, we review and discuss techniques that can be used to deal with a signature key compromise by using either of the following general methods:

1. *Providing redundancy.* A single key compromise is rendered insufficient to allow the forgery of signatures by requiring multiple keys for signature production. For example, requiring a private key to be compromised from each of a group of users, thereby requiring multiple, subsequent attacks against different users in order to successfully forge a signature;

2. *Limiting exposure.* Limiting the number or type of signatures that may be forged or the amount of time that undetected forgery can persist may limit the quantity of forged signatures resulting from a key compromise.

Threshold signatures. Threshold signature schemes (e.g., [Des94, Lan95]) are protocols in which n shares or pieces of a secret signing key are distributed amongst n users (one share per user). To produce a signature (verifiable with a single verification key), at least $t \leq n$ users must cooperate, each producing partial signatures that are thereafter combined to produce a resultant signature.

Redundancy (against one class of attacks) is provided since compromise of a single user's share does not allow one to forge a signature (unless cooperation is obtained from $t-1$ other users). Exposure is limited so long as compromises are detected and subsequent regeneration of signature keys is performed. However, there exists the possibility that over a period of time, t signature shares may be compromised.

Proactive signatures. In anticipation of the possibility of a long-term attack in which multiple shares of a signature key are eventually compromised (without detection), a proactive approach has been proposed [HJJK97] whereby the shares corresponding to a single signature key (where as above, a threshold of signature key pieces are required to produce a signature verifiable by the single verification key) are periodically renewed so that an attacker would be required to compromise a threshold of the shares all within a given time period in order to successfully forge a signature. One advantage is that despite the refreshment of the shares, the underlying private/public key pair can remain fixed for a long time, e.g. several years. This renewal of shares can be performed periodically or can be triggered by the detection of a share compromise. A second advantage is that if one of n parties holding a key share leaves an organization or is dismissed, even without explicit revocation of his key share, the periodic update will cause his key share to be invalidated.

Though suitable for some applications, for protecting individual users against key compromise a disadvantage of using threshold schemes (proactive or otherwise) is the requirement of involving a number of users to produce a single, verifiable signature. Furthermore, it is important to note that such threshold and proactive schemes do not preclude an algorithmic or brute force attack that would discover the single signing equivalent key.

Proactive certification. To remove the requirement of multiple users for the production of a verifiable signature, Canetti, Halevi and Herzberg [CHH97] use the same proactive, distributed concept (as described above for 'Proactive signatures') to allow for a proactive distributed certification of an individual user's signature key, whereby a single signature key is sufficient for the production of a signature, as opposed to a distributed signature construction. Their proactive solution requires periodic refreshment phases in which new signing key pairs (i.e., sig_u and pub_u as in Definition 2.11) are generated by each user. Users additionally store shares of a global, private signature (certification) key, corresponding to a global, public verification key. These shares are used in process (similar to the proactive signature scheme described above) to certify the new signature keys (just as would be done in a centralized scheme by a certification authority in Section 5.2.1). The shares are also periodically refreshed (in addition to the signature key pairs).

A weakness of this approach is that, although signing key pairs are refreshed at regular intervals, there is no protection in the case that a single user's signing key is compromised (without detection by the private key owner) and used to produce a signature within a given time unit. This technique therefore provides some protection against malicious certification of public keys (by providing for a decentralized certification process in which the shares corresponding to the private certification key are periodically renewed) and simultaneously limits the number of (as opposed to preventing) forged signatures that can be produced for a user by imposing periodic renewal of the user's personal signature keys.

Restricted signature privileges. An alternative technique for limiting the effects of key compromise (e.g., forgery of signatures) is related to the idea of attribute certificates. These are certificates that allow for additional information, other than a public-key, to be conveyed in an authentic manner [ITU93]. For example, the additional information may be privileges which can be certified by an attribute authority in separate certificates, or included as an optional field directly in a user's certificate. Suppose, for example, that different privileges were assigned to different users so that only certain classes of messages can be signed by particular users. For example, only users with "signing officer" privileges might be able to sign cheques in the name of their company. An attacker with such a goal in mind, now has a smaller number of users that can be attacked since the compromise of a particular signature key may not allow for the production of forged cheques. This technique can be combined with threshold signatures or proactive signatures (see above) whereby combinations of users with different attributes are required to produce a signature.

Limiting the number of signatures. While such a solution above limits the *types* of signatures that can be produced (and hence forged), one might also try to bound the *number* of signatures that can be produced for a given certificate. This idea can be implemented by using an intermediate trusted third party to decrement the remaining signature count after the production of each signature. Such a technique is used in Protocol PV4 (see Section 6.4).

Signature insurance. Related to the reduction of risk for a particular user or group of users is the protection against liability in the case of undetected key compromise. Paralleling the paper world, insurance might be useful for protection in such situations, i.e., each user pays insurance premiums for each certificate, protecting against the results of a key compromise. For example, comparing a system where single user signatures are required with one where threshold signatures are required, the former might require for higher insurance premiums.

6.3 Overview of New Approach

The verification of a time stamped signature was discussed in Section 5.3. This verification forms what we refer to here as a *primary* or *first level authentication.* The first level authentication allows a signature verifier to ensure that, among other things, the mathematical correctness of the signature is verifiable with the public key contained in the corresponding certificate identified by the purported originator of the signature. Thus, the signer must have had knowledge of the corresponding private signature key (or equivalent information). The binding of the name to the public key by the certification authority (CA) (see Section 5.2.1) is intended to identify the named individual as owner of the corresponding verification key. However, as implied by a key compromise (see Definition 6.1), the first level authentication alone does not necessarily identify the named individual as the only possessor of the (private) signature key and hence as the producer of a particular signature.

The novelty of the new approach to dealing with a key compromise (in comparison to the techniques of Section 6.2) is that it makes use of a second level of authentication, the result of which allows the verifier v of a signed message to confirm (with a higher degree of assureness than with the first level digital signature protocol) that u (the user named in the public key verification certificate used to successfully validate the first level digital signature) did indeed sign the message m despite a malicious attacker's possible possession of sig_u (or equivalent key). A successful second-level authentication results in a signature, produced by a third party Trusted Register (TR), over (at least) the signed message submitted by the originating user. After successful verification by the originating user, the original (time stamped) signature and message (to which the first level signature was applied) can be sent to other users, accompanied by the TR-signed message. More detailed descriptions of the particular techniques whereby an enhanced second-level authentication is used are given in Section 6.4 and Section 6.5.

In Section 6.3.1, the general structure and form of a second level authentication, incorporated with the first level digital signature, is described. Section 6.3.2 positions this second level approach relative to the previous work (of Section 6.2) and the provision of temporal authentication.

6.3.1 Second Level Authentication for Signature Production

The secondary method provides additional, corroborative evidence for the verifier of a digital signature, beyond the possession of the original first level signature. This corroborative evidence results from an exchange made between the signing user and the TR. The function of the TR is to validate an exchange between itself and the submitter of a secondary authentication request, and subsequently produce some substantiating information (to enhance the acceptability of a message signed with sig_u) that is bound to the signature in question. Throughout this chapter, we consider the technique whereby the TR produces a signature in response to a successful second level authentication. Optionally, one might record user signatures in an integrity-protected database at the TR; signature recipients could verify the success of the second level authentication for a particular signature by querying the TR regarding the membership of the signature in the database.

The generic structure of the combined first and second level authentications is described by Protocol SL1. The properties of the secondary authentication mechanism are given in Table 6.1. The beginning of a *round* is defined as the moment a (legitimate or fraudulent) secondary authentication request is received by the TR and ends when the corroborating evidence r is received by the requester. Synonymous to Definitions 5.16 and 5.17, we can also identify the legitimacy of secondary requests with the following.

Protocol SL1 Generic Structure of Signature Protocols Using a Second Level Authentication.

Description: This protocol provides the general steps combining a first level signature with a second level authentication. This second level authentication combines a secondary request to the TR with a return of secondary corroborative evidence to the submitting user. This corroborative evidence provides additional corroboration to the named producer of the first level signature.

Note: The more specific protocols described in Sections 6.4 replace steps 4 and 5 below while enhancements to allow for synchronization are described as Protocol DT1 of Section 6.5.

1: User u computes the signature $c = sig_u(m)$ for the message m.

2: User u obtains a time stamp s for the signature c, using for example, Protocol AB1 of Section 2.4.1.

3: User u also sends the signature $c = sig_u(m)$ to the trusted register TR.[a]

4: Along with c, u submits c' as an algorithm dependent, second level authentication request (see Remark 6.1) for the signature c to the TR.

5: The TR validates the second-level authentication, and if successful, returns

$$r = sig_{TR}(c', c, u, \ldots),$$

the secondary authentication corroborating evidence, along with its contents to u. Here '...' refers to additional algorithm specific data.

6: u verifies the mathematical correctness of the TR-signed r using an *a priori*, authentically stored copy of the TR's verification public key and ensures that its contents match what was submitted by u.

7: u sends $\{m, c, r, s\}$ to a signature recipient v along with contents required to determine the mathematical correctness of any signatures.

8: v validates u's signature c over m and checks the time stamp s using Protocol DS1 (Section 5.3.2) and verifies the mathematical correctness of the TR's signature r over $(c', c, u, \ldots)$.

9: If each validation is successful, v accepts the signature c over the message m.

[a]Alternatively, u might submit the time stamp s to the TR.

1. Any secret information or algorithms upon which the secondary authentication mechanism relies (or more generally, things that may be vulnerable to the attacks mentioned in Table 5.5) should be 'independent' of the signing private key or algorithm used for the signature production itself, i.e., compromise of one doesn't reveal information sufficient enough to allow a computationally feasible key compromise attack against the other.
2. The secondary authentication corroborating evidence is cryptographically bound or associated with the current signature in question, i.e., is computed as a cryptographic function of the signature.
3. The method permits a suitable identification to the TR, i.e., allows the TR to verify that only u could have produced a particular signature, given an *a priori* agreed upon identification technique between u and the TR. This identification is performed via submission of a *secondary authentication request* (see Remark 6.1).

Table 6.1: Properties Required for the Second-Level Authentication Mechanism.

Definition 6.3 A secondary authentication request c' (see Remark 6.1) is *legitimate* if the request received by the TR is the same as that transmitted by the legitimately identified requestor. A request that is not legitimate is *fraudulent* or *forged.*

Remark 6.1 *The secondary authentication request serves a purpose similar to a response in a challenge-response protocol. For our purposes, the "challenge" data refers to data shared (possibly secretly) between the legitimate user and the TR, combined with the digital signature data for which a secondary request is being made. The response (i.e., the user's secondary authentication request) is a function of this information.*

The independence of the mechanism used to perform the secondary authentication request, from the first level signing private key and algorithm, typically ensures that a second attack would be required subsequent to compromise of the first level signing key. In this way, the independence of the second level from the first allows one to better survive attacks that may only succeed against the first level.

Identification of a user to the TR (facilitated by the secondary request) can be based on something known (e.g., a password), something possessed (e.g., a smart-card), or something inherent to an individual entity (e.g., a fingerprint). Isolating on 'something known', we observe that the known information can be either *static* (e.g., mother's maiden name, birthdate) or *dynamic* (e.g., a periodically changed password). We can also identify non-secret *synchronization parameters* which are specifically used for a synchronization scheme with the TR (see Section 6.5). In this case, a lack of synchronization allows for the detection of forged signatures. Only the authenticity of this parameter need be maintained, not its confidentiality.

6.3.2 Positioning of New Work

In this subsection, we outline the relationship of the new, second level solution to the previous work of Section 6.2 and to the provision of temporal authentication.

Outline of Solutions and Relationship to Previous Work

In Section 6.4, second level authentication solutions are presented which provide for an identification of the user to the TR using either 'something possessed' or 'something inherent' to the requesting user. These solutions provide for redundancy whereby compromise of the first level signature key is not sufficient for the production of a signature that would be accepted by a verifier. They differ from the threshold schemes reviewed in Section 6.2 in which single shares of a key are held by multiple users in that multiple keys are held by a single user.

In Section 6.5, schemes which limit the exposure to successfully forged signatures subsequent to a compromise are examined. These schemes use a synchronization between the legitimate user and the TR, allowing the detection of fraudulent secondary authentication requests and thus of forged signatures to be detected. Suitable revocation techniques can subsequently be performed subsequent to a detection. This solution differs from the threshold and proactive signature schemes of Section 6.2 in that individual users can unilaterally produce signatures. It differs from the proactive certification scheme reviewed in Section 6.2 in that it is proactive in the sense of providing for the detection of forged signatures, but does not require the regeneration of keying material unless a forgery is detected. In short, the new schemes do not require a key pair refreshment unless a forged signature attempt has been detected. As well, the new proposals protect against other possibilities for the compromise of a private key, beyond a break-in (see Table 5.5).

In Section 6.6, we build on the detection schemes of Section 6.5 whereby once a fraudulent request is detected, the acceptance of any forged signatures can be prevented. This is accomplished by creating a cooling-off period for the acceptance of signatures while requiring legitimate key owners to acknowledge signatures for which secondary, corroborative evidence has been obtained, yet are currently cooling off and have not been accepted by signature recipients.

Temporal Functions of the Trusted Register

While the solutions presented here are positioned primarily for the purpose of providing a second level authentication to protect against the case when an undetected signature key compromise (or equivalent attack) has occurred, they are intimately related to the temporal digital signature requirements discussed in Section 5.3.1. The additional requirement introduced in this chapter (beyond attempting to limit the effects of a single key compromise), in the case of forged signatures, is the determination of when signatures were produced relative to the time of compromise. Although, as indicated in Section 6.1.4, time stamping is not sufficient to solve this problem, other techniques (such as the synchronization methods of Section 6.5) can be used to help determine a time around which forged signatures were first produced. As an additional role, the TR authority used in this solution may also simultaneously act as a time stamp authority T or a digital signature notary authority DSN.

Additionally incorporating the role of a time stamp authority, the TR would return $r = sig_{TR}(c', c, u, \ldots, t)$ to u, in place of $r = sig_{TR}(c', c, u, \ldots)$ (see Protocol SL1 of Section 6.3.1), where t may be the time of receipt of c. The time stamping of the signature is necessary in any case, and this is one option for implementing time stamping with a TR. (See Protocol DT7 in Section 6.5.3 for a case in which the time stamp is incidently provided in the second level authentication response from the TR.)

Considering that the TR is performing a role similar to a DSN, by verifying (but not fully attesting to) the success of an attempted second-level authentication request by u, the TR might also act as a notary and verify the signature c (as in Section 5.4) upon submission. In this way, the TR might also include a full attestation to the success of the second as well as the first level authentication.

6.4 Preventing Forged Signature Production

The successful forgery of a signature in a two-level signature scheme requires that beyond the compromise of the signature key, an attacker is also able to obtain a second-level authentication from the trusted register (TR). Both a first level digital signature and second level corroborative evidence from the TR are required for a signature recipient to accept a signature.

Definition 6.4 We say that the forgery of signatures is *prevented* in a two-level signature scheme (i.e., Protocol SL1) if compromise of the primary signature key does not allow one to produce a signature that would be acceptable by a signature verifier.

A successful attack would require the attainment of a forged signature and second level corroborative evidence that would be accepted by a signature verifier. In Section 6.4.1, we present several protocols in which a secondary, private key, some function of which is shared between the legitimate user and the TR, is used in a secondary authentication request by the legitimate user. In Section 6.4.2, the storage and transmission efficiency of the protocols is compared.

6.4.1 Second Level Protocols

Consider, for motivational purposes, Protocol PV1. It is impractical so long as current technology is unable to consistently recognize a user's voice while also preventing impersonations. As well, it requires the "physical" intervention of the signing user as opposed to a completely automated process. However, it does allow an originating user to obtain corroborative evidence regarding the source of the signature c that can be supplied to potential signature recipients. In subsequent protocols, we present more cryptographic solutions.

Protocol PV1 Using Biometrics as Secondary Authentication.

Description: This protocol describes the secondary authentication request and return of corroborative evidence by a trusted register (TR), replacing the like-numbered steps from the general secondary authentication Protocol SL1 (see Section 6.3.1).

4: u places a phone call to the TR identifying himself as u, reading the output of $h(c)$ to the TR for signature c.

5: The TR validates that the voice of the requestor matches the stored vocal properties for user u, determines the mathematical correctness of $h(c)$ by computing and comparing $h(c)$ upon receipt of the signature c, and returns $r = sig_{TR}(c, u)$ to u if the validation is successful.

Whereas Protocol PV1 uses a separate channel for the secondary authentication request, the remaining protocols in this subsection transmit the secondary authentication request along with the first level signature. Consider the scheme described as Protocol PV2. At least one of the secondary algorithm or key must be independent from their primary (first level) counterparts. For example, if the secondary algorithm were DSA [FIP94] and the primary were RSA [RSA78], the second level would likely be resistant to potential attacks that existed only against RSA. In this case, the signature algorithms are independent with respect to attacks that do not simultaneously compromise the security of both schemes. Regarding the use of a secondary key whose secrecy must be maintained, similar to the original signature key, both the construction and storage of the keys must be independent. In other words, an attack to the first key should not allow recovery of the second key. In the best case, the compromise of the first key would be detected prior to compromise of the second, allowing revocation of the verification certificate corresponding to the first key. Notice that a certificate need not necessarily be constructed for the secondary public key pub'_u since this public key will only be used by the TR (as opposed to other users).

Protocol PV2 Using a Signature as Secondary Authentication.

Description: This protocol describes the secondary authentication request and return of corroborative evidence by a trusted register (TR), replacing the like-numbered steps from the general secondary authentication Protocol SL1 (see Section 6.3.1).

Require: u must possess a secondary private signature method sig'_u parameterized by a key independent from the primary signature key and corresponding secondary public key pub'_u. The TR maintains a copy of pub'_u.

```
4: $u$ computes $c' = sig'_u(c)$ and sends $(u, c')$ to the $TR$.
5: The $TR$ verifies the mathematical correctness the secondary signature $c'$, using
   $pub'_u$, and if successful, returns $r = sig_{TR}(c', c, u)$ to $u$.
```

Alternatively, a private key algorithm can be used whereby u privately shares a symmetric key K with the TR as in Protocol PV3. This solution prevents an attacker from succeeding at having forged signatures accepted so long as he/she is not able to recover K, in addition to the private, first level signature key. The storage location and algorithm used with K must be independent of the location of the signing private key and signature algorithm. E can be either an encryption function or preferably a MAC algorithm since no decryption operation need be performed by the TR. Note that a MAC provides for a smaller secondary request size since encryption of c results in a request size equal in length to the size of the signature, which is longer than the output of a typical MAC function (cf. Table 6.2). Note that an attacker, in possession of only the signature private key sig_u, would not be able to obtain corroborative evidence for a forged signature.

Protocol PV3 Using a Symmetric Key as Secondary Authentication.

Description: This protocol describes the secondary authentication request and return of corroborative evidence by a trusted register (TR), replacing the like-numbered steps from the general secondary authentication Protocol SL1 (see Section 6.3.1).

Require: u shares a symmetric key K with the TR.

```
4: $u$ computes $c' = E_K(c)$ (for first level signature $c$) and sends $(u, c')$ to the $TR$.
5: The $TR$, using knowledge of $K$ and receipt of $c$, computes $c'' = E_K(c)$ and returns
   $r = sig_{TR}(c', c, u)$ to $u$ if $c' = c''$.
```

A variation from Protocol PV3 in which the TR need not maintain the secrecy of any information (that would be required for verification of the next secondary authentication requests) for u uses Lamport one-time keys [Lam81], and is presented as Protocol PV4. E must be invertible in this case (differing from Protocol PV3) to allow recovery of K_{i+1} as the key required for the next secondary request. An advantage of this scheme (as compared to Protocols PV2 and PV3) is that a different, pseudo-independent key is used to produce c'_i for each i. As well, compromise of s limits an attacker to a fixed number of forged secondary requests. (A variation of this scheme whereby the secrecy of s is not required by the user u, is given in Section 6.5 as Protocol DT4.)

Protocol PV4 Using a Private Seed as Secondary Authentication.

Description: This protocol describes the secondary authentication request and return of corroborative evidence by a trusted register (TR), replacing the like-numbered steps from the general secondary authentication Protocol SL1 (see Section 6.3.1).

Note: Signature c_i refers to the round i instance of signature c from Protocol SL1. Initially, $i = 0$, and is incremented by 1 for each secondary authentication request.

Require: u privately shares a secret encryption function key $K_i = f^{n-i}(s)$ with the TR where $s \in S$ is a random, secret seed, $f : S \to S$ is a one-way function (i.e., a function for which it is easy to compute an image for all domain elements but computationally infeasible to compute a pre-image for almost all images) and n is a positive integer denoting the number of signatures u may produce before requiring reinitialization with the TR.

4: For signature c_i, u computes $c'_i = E_{K_i}(c_i, K_{i+1})$ (where ',' denotes concatenation) and sends (u, c'_i) to the TR.

5: Given possession of K_i, the TR decrypts c'_i, recovers K_{i+1} and computes $f(K_{i+1}) = f(f^{n-i-1}(s)) = f^{n-i}(s)$ to ensure that it equals K_i. If true, the TR subsequently stores K_{i+1} in place of K_i and returns $r = sig_{TR}(c', c, u)$ to u.

6.4.2 Comparative Analysis

In this subsection, we provide some comparative analysis of the storage and transmission efficiency of Protocols PV2, PV3 and PV4. Each of these schemes requires a secondary, secret key to be maintained by each user u. u might have several signature keys (corresponding to several public key certificates) but need only keep a single secondary key. The independence of this secondary key (and algorithm) increases the likelihood that an additional attack would be required to compromise the secondary mechanism given a compromise of the first level.

Table 6.2 displays a comparison of the protocols with regard to several storage criteria. Each user stores and maintains the privacy of only a single secondary key while the TR need only maintain a single key corresponding to each user (though see the footnote to 'TR Storage' of Table 6.2). Although Protocol PV2 does not require the TR to maintain the privacy of the public keys for each user, it does require larger storage for u (and the TR if the secondary signature key is not implemented using a certificate) as well as a larger transmission size when compared to Protocol PV3. Protocol PV4 matches Protocol PV3 for user and TR storage, but because of the requirement of a reversible function (allowing the TR to recover K_{i+1} from K_i), Protocol PV4 requires a larger secondary authentication request size.

6.5 Detecting Forged Signatures

Attack detection, per the techniques proposed herein, involves the discovery of a lack of synchronization between the legitimate signing user and the TR; this implies the detection of a fraudulent secondary authentication request and may imply that a forged signature has been constructed. The techniques for achieving this detection involve the use of so-called *synchronization parameters*. Only the authenticity of this parameter need be maintained by both the user and the TR, not its privacy. For every message signed by a user (even if a message is signed by an attacker in possession of the legitimate user's signature private key), for which a secondary authentication request is made, the parameter is updated by the TR. Detection occurs when the legitimate signer is not synchronized with the TR at a given legitimate message signing, implying an attacker has fraudulently and successfully submitted a secondary authentication request since the last request made by the legitimate signing user. Unless otherwise noted, it is assumed that the detection of a fraudulent request implies a detection of signature forgery and hence key compromise.

	Protocols		
Properties	PV2	PV3	PV4
user storage	1 signature key	1 MAC key	1 encryption key
	160 bits	128 bits	128 bits
TR storage[a]	1 public key[b]	1 MAC key	1 decryption key
	1696 bits	128 bits	128 bits
request size	1 signature	1 MAC	1 signature + 1 key
	320 bits	128 bits	508 bits

Table 6.2: Comparison of Techniques Using a Secret Key for Secondary Authentication. User storage refers to the storage required by u to allow use of the secondary authentication mechanism (ignoring the requirement to store the verification key of the TR which is required for all schemes). The TR's storage refers to the storage required for each participating user, to be maintained in a central database. Request size refers to the size of the secondary authentication request from u to the TR. This table assumes the use of DSA (Protocol SG1 of Section 2.1.2) for digital signature production and verification and 128-bit MAC and encryption keys with 128-bit MAC output.

[a]At the risk of concentrating too much reliance on a single master key, a standard proposal for simplifying key management would be for the shared secret key for user i to be $K_i = h(K, u_i)$. Then TR need only store one key K to allow regeneration of all user keys. This technique applies to Protocols PV4 and PV3.

[b]Not required if sent in a CA-signed certificate with each request, and TR has a trusted copy of the CA's verification key.

Detection alone does not prevent an attacker (in possession of a user's signing private key) from forging signatures which would normally be accepted as valid. However, it does allow detection, and action can be taken to prevent continued forgeries. In Section 6.6 we introduce techniques that can be used to enhance this detection so that forged signatures are not accepted by an unknowing recipient and no legitimate signatures accepted by another user can be repudiated.

Outline of Section 6.5

In Section 6.5.1, properties and assumptions related to the detection of forged signatures using synchronization are discussed. In Sections 6.5.2 and 6.5.3, techniques are presented in which one-way function variant and time variant parameters are respectively used for synchronization. In Section 6.5.4, the storage of the secondary token is examined and the necessity of maintaining the parameter's authenticity is also discussed.

6.5.1 Use of Synchronization for Detecting Forgeries

The proposed method for detection of a signature forgery involves the detection of a lack of synchronization between the legitimate signing user u and the TR, and occurs at points when a signer requests secondary authentication evidence. This synchronization can be implemented using a *synchronization parameter* locally stored by both u and the TR, and updated by the TR after each secondary authentication request and by u after each legitimate request. The key feature with this parameter (when compared to the private key techniques of Section 6.4) is that it need not be kept private; only the authenticity of the parameter need be maintained.

Definition 6.5 Let s_i^u and s_i^T respectively represent the value of the *synchronization parameter* stored by u and the TR after round $i \geq 0$ where initially

$$s_0^u = s_0^T = IV$$

for an initialization value IV.

The equality of these synchronization parameters during the normal running of a secondary protocol is critical to the detection of a lack of synchronization.

Protocol DT1 Generic Secondary Authentication Using Synchronization.

Description: The steps in this protocol expand on steps 4 and 5 from Protocol SL1, particularly for synchronized secondary authentication. Each round begins with a secondary authentication request and ends with the return of corroborating evidence from the TR.

1: u and the TR initially share an initialization value IV so that for their respective synchronization parameters, $s_0^u = s_0^T = IV$.

2: During round $i \geq 1$, u submits the value $c_i' = s_{i-1}^u$ as a secondary authentication request (see Remark 6.1) to the TR,[a] along with the signature c_i.

3: The TR receives the signature c_i and secondary authentication request c_i' and verifies the correctness of the request by determining whether $c_i' = s_{i-1}^T$. If equal, the TR

   1. updates the synchronization parameter stored for u from s_{i-1}^T to s_i^T, and
   2. returns the secondary corroborative evidence $r_i = (c_i', s_i^T, c_i, u, sig_{TR}(c_i', s_i^T, c_i, u))$ to u,

   where inclusion of s_i^T is not required in the case that u is able compute s_i^u. If $c_i' \neq s_{i-1}^T$, the TR follows Protocol DT2.

4: Upon receipt of r_i, u verifies its mathematical correctness and ensures that it was indeed signed by the TR (using the *a priori* stored copy of the TR's verification public key). u also ensures that the returned value of c_i' matches s_{i-1}^u. If correct, u updates the locally stored synchronization parameter from s_{i-1}^u to s_i^u (using s_i^T if returned by TR, else computing independently). If the signature verification is not successful or the contents of the signature are erroneous, u follows step 5 of Protocol DT2. If u does not receive r_i after some predetermined amount of time, that is *a priori* set between u and the TR, then u contacts the TR through out-of-band means to determine the status of the response r_i.

[a]Alternatively, u may choose to combine the signature c_i with the secondary request value c_i' and send $c_i' = sig_u(s_{i-1}^u, c_i)$.

The general synchronized secondary authentication protocol is described as Protocol DT1. Table 6.3 identifies some assumptions regarding the execution of Protocol DT1. For Item 2 (of Table 6.3), observe that detection serves to aid in protecting an honest signer from signatures forged subsequent to a key compromise. Of course,

1. secondary authentication evidence must be verified before accepting a user's first-level signature;
2. the legitimate signer u is honest, i.e., behaves according to the protocol (we remove the need for this requirement in Section 6.6). In particular,
   (a) u honestly reports the receipt of invalid secondary responses from the TR,
   (b) u submits correct secondary authentication requests;
3. the TR is honest, i.e.,
   (a) the TR verifies the correctness of the sychronization parameter received from a requestor of a secondary authentication request (which may or may not be the legitimate signature key owner),
   (b) the TR will report any lack of synchronization detected from a secondary authentication request;
4. the authenticity of the secondary synchronization parameter is maintained, both the version stored by u and by the TR.

Table 6.3: Requirements for Protocol DT1.

the earlier the compromise is detected, the less the effect on u. We expand on this idea in Section 6.6 to protect signature recipients from fraudulent non-repudiation of legitimately produced and accepted signatures.

In an ideal system in which the legitimate signer is honest and there are no fraudulent secondary authentication requests, the values of the synchronization requests, as observed by u and the TR, would be the same. Realistically, this cannot be assumed. We say that a protocol is *detection resilient* (*D-resilient*) if either u or the TR are able to detect the differences in their synchronization parameters. We formalize this concept below with Definition 6.8.

Detection Resilience

Definition 6.6 The ith *view* of a secondary synchronization protocol for u and TR is defined as the value of the synchronization request respectively sent by u and received by the TR during the ith round. For the legitimate signer u, this view is denoted by v_i^u and v_i^T for the TR, $i \geq 1$, where

$$\begin{aligned} v_i^u &= s_{i-1}^u \\ v_i^T &= s_{i-1}^T, \end{aligned}$$

for the synchronization parameters (see Definition 6.5) s_{i-1}^u and s_{i-1}^T. ■

At the end of a legitimate round i, s_{i-1}^T is updated to s_i^T to reflect the change with the new synchronization parameter, so that the view of TR for round $i+1$ is the updated value of the synchronization parameter. The same holds true for the view of the legitimate requestor u. For a fraudulent request, s_{i-1}^T is updated as above to s_i^T, but s_{i-1}^u is not updated, since for a fraudulent request, we assume that u did submit the request. Therefore, the $i+1$st view of u is vacuously updated to the same view (i.e., $v_{i+1}^u = v_i^u$). The view of the protocol is critical to its proper running as well as its ability to detect fraudulent secondary authentication requests. These properties, are respectively defined below as the protocol's correctness and detection resilience.

Definition 6.7 Let P be a synchronized secondary authentication protocol (as in Protocol DT1) satisfying the requirements of Table 6.3. P is *correct* if, when only legitimate secondary authentication requests are performed, then

$$v_i^u = v_i^T, \forall i \geq 1$$

■

A subtle point in the case that fraudulent secondary authentication requests are made is that it is important that not only are the fraudulent requests expected to alter the equality of the legitimate user and TR views, but also, that the attacker should not be able to "resynchronize" these views. This point is captured with the following definition.

Definition 6.8 Let P be a synchronized secondary authentication protocol (as in Protocol DT1) satisfying the requirements of Table 6.3. P is *detection resilient* (*D-resilient*) if subsequent to a fraudulent authentication request in round i, then for

$j > i$

$$v_j^u \neq v_j^T. \tag{6.1}$$

In other words, an attacker cannot compute v_j^T for $i < j < k$ such that $v_j^T = v_i^u$, where k represents a number for which $k - j$ is a computationally feasible number of secondary authentication requests to make. (For example, the attacker might try to make additional fraudulent requests in an attempt to reach a point which matches the view of TR to the earlier view of u. ■

Notice that (6.1) holds whether or not $v_{i-1}^u = v_{i-1}^T$ (for the views prior to a fraudulent request), so that an initial fraudulent request (subsequent to a legitimate one) or repeated fraudulent requests cannot resynchronize u (through any amount of computationally feasible computations) with the TR, using a D-resilient protocol. The key to the D-resilience of a protocol lies in the ability of detecting the inequality $v_j^u \neq v_j^T$ by u or the TR during some round $j > i$. In the following, we consider the attacks for which a D-resilient protocol is suitable protection against.

Attacks Considered on Secondary Synchronization Protocols

Assuming that a fraudulent secondary authentication request results from a protocol requirement to obtain secondary authentication evidence for any (including a forged) signature, there are at least two cases to consider regarding the forgery of a signature by an attacker X (in possession of u's signature key):

1. X alters a current legitimate secondary signature request made by u;
2. X constructs a secondary authentication request that is either
   (a) newly constructed, independent of any previous request made by u, or
   (b) constructed as a function of previous legitimate or fraudulent secondary authentication requests.

We argue that for Item 1, if X alters a request from u to the TR, the alteration will be detectable upon receipt by u of the secondary authentication response from the TR since the signature and synchronization parameter are included in the secondary authentication response (see Protocol DT1). We assume that if X were to block this response, then u would interpret the absence of a response as a suspected compromise and report a possible key compromise to u's CA, whereby subsequent revocation actions may be taken.

For the protocols presented in Sections 6.5.2 and 6.5.3, we consider attacks as described by Item 2. Demonstrating a protocol's D-resilience (see Definition 6.8) will involve demonstrating that fraudulent secondary authentication requests are detectable by u or the TR.

Updating the Synchronization Parameter

A stronger restatement of the D-resilience requirement of Definition 6.8 is to require that for the sequence

$$v_1^T, v_2^T, \ldots, v_k^T, \tag{6.2}$$

there are no $1 \leq i < j \leq k$ such that $v_i^T = v_j^T$. We identify two types of values that can be used for the synchronization parameter so as to ensure this property.

1. *Time-variant.* The use of a time-variant synchronization parameter such that the value of the parameter is monotonically increasing with time can be used. In this way, for round j that occurs later than round i, we have $v_j^T > v_i^T$.
2. *One-way function variant.* Combining a synchronization parameter that is updated each round with a one-way, collision-resistant hash function (see Definition 2.10) produces a "non-repeating" sequence satisfying (6.2).

Protocols satisfying Item 2 are discussed in Section 6.5.2 while protocols satisfying Item 1 are examined in Section 6.5.3.

Dealing with Fraudulent Secondary Requests

The TR detects fraudulent secondary authentication requests upon receipt of a synchronization parameter for a particular user, that doesn't match the value stored by the TR.[3] The legitimate signing user u detects a fraudulent request, either by the

[3]Assuming correct operation of the protocol by legitimate parties, and the absence of network transmission errors, etc.

return of a response from the TR that does not match the request submitted, or upon notification from the TR that a fraudulent request has been received. Once a fraudulent secondary authentication request has been detected by u or the TR, it is not always necessary for immediate revocation of the legitimate user's verification certificate to be performed. Protocol DT2 describes the actions taken subsequent to a detected fraudulent request.

6.5.2 One-Way Function Variant Requests

In this subsection, we present two secondary authentication protocols that follow the general structure of Protocol DT1 (see Section 6.5.1). Both use a secondary request that is a hash of a non-secret synchronization parameter shared between u and the TR. One way to provide a synchronization is for u to acknowledge the signing of each of the past signatures legitimately produced (i.e., from u's point of view) with sig_u, each time a new request for secondary authentication is made. The synchronization parameter is a function of the past signatures. An efficient way to perform this is offered by Protocol DT3.

Protocol DT3 uses a round variant, based on the variety of signatures submitted from one round to the next. Protocol DT4, a variation of Protocol PV4 (see Section 6.4.1) uses an iterative function of an initially shared seed. An incrementing count of the current round is used to vary the number of iterations performed for the hash function.

Detecting Forged Signatures

Before discussing the security of DT3 and Protocols DT4 , we present Protocol DT5 which illustrates a potential insecurity for such synchronization protocols. This insecurity may not be obvious because of similarities with Protocols DT3 and DT4.

Consider the following series of steps performed by an attacker X (in possession of u's signature key), subsequent to the legitimately signed $c_i = sig_u(m)$ (i.e., signed by u) for round i. The current views of u and the TR, for anticipated use in round

Protocol DT2 Dealing with Fraudulent Secondary Authentication Requests.

Description: This protocol supports the detection of fraudulent secondary authentication requests as detected by Protocol DT1 (see Section 6.5.1), by describing the actions taken by the TR and u subsequent to a detection.

Note: Requests for certificate revocation result in a revocation of the primary signature key as well as a reinitialization of the secondary synchronization parameter.

1: If the TR receives a fraudulent secondary authentication request, u is contacted through out-of-band means, using a protocol pre-arranged with u, e.g., contacting u through a telephone number supplied by u upon registration with the TR.

2: If reliable contact is not made from step 1, then the TR proceeds to request a revocation of u's certificate from u's CA.

3: If reliable contact is made so that the legitimate u is informed of the fraudulent request, the TR subsequently sends u, $w = sig_{TR}(c_i', c_i, \text{‘fraudulent’})$.

4: If u did not send the request, then u determines the extent of fraudulent requests by comparing c_i' with the u's current view of the synchronization parameter, $v_i^u = s_{i-1}^u$.

   1. If they are equal, and the protocol is D-resilient (see Definition 6.8), then u can conclude that only 1 fraudulent request has been made. u verifies the correctness of the signature c_i (using u's own verification key). If correct, u requests a certificate revocation from the CA. If incorrect, nothing is done since a forged signature has not been detected.

   2. If they are not equal and the protocol is D-resilient, then u can be sure that at least 1 successful fraudulent request has been made prior to the current unsuccessful one. u requests a certificate revocation from the CA (to prevent additional frauds).

5: If u did send the fraudulent request, and the protocol is D-resilient and u is honest, then the fraudulent request occurred because of the lack of synchronization, indicating that previous, successful fraudulent requests have occurred. u requests a certificate revocation from the CA (to prevent additional frauds).

Protocol DT3 Synchronization by Verifying Recursive Representation of Past Signatures.

Description: This protocol replaces the protocol-specific functions as described in Protocol DT1. Initially, $i = 0$ and is incremented by 1 at the start of each round. This protocol is D-resilient (see Proposition 6.1).

Require: It is necessary that $IV \notin \{0,1\}^l$ where $\{0,1\}^l$ is the co-domain for the hash function h.

```
1: $u$ and the $TR$ initially share the synchronization parameter $s_0^u = s_0^T = IV$ for initialization value $IV$, whereas prior to round $i$, assuming no fraudulent secondary requests have been made, they share $s_{i-1}^u = s_{i-1}^T = h(s_{i-2}^u, c_{i-1})$ where $h$ is a collision-resistant hash function (see Definition 2.10).
2: During round $i \geq 1$, $u$ submits the value $c_i' = s_{i-1}^u$ to the $TR$ as a secondary request along with the signature $c_i$.
3: The $TR$ determines the correctness of the request by ensuring that $c_i'$ is equal to $s_{i-1}^T$ (the $TR$'s stored value for $u$). If equal, the $TR$ computes and stores $s_i^T = h(s_{i-1}^T, c_i)$ and returns $r_i = (c_i', c_i, u, sig_{TR}(c_i', c_i, u))$ to $u$.
4: $u$ verifies the mathematical correctness of $r_i$ and ensures that its contents match what was originally sent by $u$. If successful, $u$ computes and stores $s_i^u = h(s_{i-1}^u, c_i)$.
```

Protocol DT4 Using a Shared Seed for Synchronization.

Description: This protocol replaces the protocol-specific functions as described in Protocol DT1. Initially, $i = 0$ and is incremented by 1 at the start of each round. This protocol is D-resilient (see Proposition 6.1).

Require: It is necessary that $IV \notin \{0,1\}^l$ where $\{0,1\}^l$ is the co-domain for the hash function h.

```
1: $u$ and the $TR$ initially share the synchronization parameter $s_0^u = s_0^T = IV(= h^0(IV))$ for initialization value $IV$, whereas prior to round $i$, assuming no fraudulent secondary requests have been made, they share $s_{i-1}^u = s_{i-1}^T = h^{i-1}(IV)$ where $h$ is a collision-resistant hash function (see Definition 2.10) and $h^n(IV) = \underbrace{h(h(\cdots h(IV)\cdots))}_{n \text{ times}}$.
2: During round $i \geq 1$, $u$ submits the value $c_i' = s_{i-1}^u$ to the $TR$ as a secondary request along with the signature $c_i$.
3: The $TR$ determines the correctness of the request by ensuring that $c_i'$ is equal to $s_{i-1}^T$ (the $TR$'s stored value for $u$). If equal, the $TR$ computes and stores $s_i^T = h(s_{i-1}^T)$ and returns $r_i = (c_i', c_i, u, sig_{TR}(c_i', c_i, u))$ to $u$.
4: $u$ verifies the mathematical correctness of $r_i$ and ensures that its contents match what was originally sent by $u$. If successful, $u$ computes and stores $s_i^u = h(s_{i-1}^u)$.
```

Protocol DT5 An Insecure, Signature-Dependent Synchronization (that is not D-resilient).

Description: This protocol replaces the protocol-specific functions as described in Protocol DT1. Initially, $i = 0$ and is incremented by 1 at the start of each round. This protocol is *not* D-resilient.

Require: It is necessary that $IV \notin \{0,1\}^l$ where $\{0,1\}^l$ is the co-domain for the hash function h.

1: u and the TR initially share the synchronization parameter $s_0^u = s_0^T = IV$ for initialization value IV, whereas prior to round i, assuming no fraudulent secondary requests have been made, they share $s_{i-1}^u = s_{i-1}^T = h(c_{i-1})$ where h is a collision-resistant hash function (see Definition 2.10).

2: During round $i \geq 1$, u submits the value $c_i' = s_{i-1}^u$ to the TR as a secondary request along with the signature c_i.

3: The TR determines the correctness of the request by ensuring that c_i' is equal to s_{i-1}^T (the TR's stored value for u). If equal, the TR computes and stores $s_i^T = h(c_i)$ and returns $r_i = (c_i', c_i, u, sig_{TR}(c_i', c_i, u))$ to u.

4: u verifies the mathematical correctness of r_i and ensures that its contents match what was originally sent by u. If successful, u computes and stores $s_i^u = h(c_i)$.

$i + 1$ are

$$v_{i+1}^u = s_i^u = h(c_i) = s_i^T = v_{i+1}^T.$$

In the following attack, X obtains a successful secondary authentication from the TR, but does so in a way that makes the attack undetectable to u and the TR:

1. In round $i + 1$, X forges the signature $c_{i+1} = sig_u(m')$ and obtains secondary authentication corroboration evidence from the TR, by submitting $c_{i+1}' = h(c_i)$ as a secondary authentication request which the TR verifies as correct. The TR subsequently stores $s_{i+1}^T = h(c_{i+1})$;

2. To "cover his tracks", during round $i + 2$, X resubmits the signature $c_{i+2} = c_i = sig_u(m)$ for secondary authentication, where X submits $c_{i+2}' = h(c_{i+1})$ as a secondary request which the TR verifies as correct. The TR subsequently stores $s_{i+2}^T = h(c_{i+2})$.

At the end of this attack (i.e., after round $i + 2$), the view of the TR will be

$$v_{i+3}^T = s_{i+2}^T = h(c_{i+2}) = h(c_i) = s_i^u = v_i^u$$

so that according to the TR's "state" information, the state from before the forgery is equal to the state after the forgery and is hence, not detectable by u nor the TR (so long as u did not submit a legitimate request during the attack).

Notice that for Protocol DT5, the ability of an attacker to resynchronize is not restricted to dependencies on previous signatures or synchronization tokens. Each secondary request c'_j submitted during round $j > i$ can be constructed so that it equals any $v_j^T = v_i^u$; Protocol DT5 is not D-resilient. On the other hand, Protocols DT3 and DT4 are constructed so that each s_i^u has a cryptographically strong dependence on previous signatures and synchronization tokens. The strength of this bind is illustrated in Proposition 6.1.

Proposition 6.1 *Protocols DT3 and Protocol DT4 are D-resilient (see Definition 6.8).*

Proof Let $v_i^u = s_{i-1}^u = h(s_{i-2}, c_{i-1}) = s_{i-1}^T = v_i^T$ be the view of both the legitimate signing user u and the TR after the $(i-1)$st legitimate secondary authentication request for the signature c_{i-1}. To show D-resilience, we need to demonstrate that for no $j > i$, can an attacker X produce v_j^T such that $v_j^T = v_i^u$.

Suppose that X did find such a j, and let j be the smallest positive integer greater than i for which $v_j^T = v_i^u$. Expanding, we have

$$v_j^T = s_{j-1}^T = h(s_{j-2}^T, c_{j-1}) = h(s_{i-2}^u, c_{i-1}) = s_{i-1}^u = v_i^u. \tag{6.3}$$

There are two cases to consider:

1. if $(s_{j-2}^T, c_{j-1}) \neq (s_{i-2}^u, c_{i-1})$, then one obtains a contradiction to the assumption that h is a collision-resistant hash function;

2. if $(s_{j-2}^T, c_{j-1}) = (s_{i-2}^u, c_{i-1})$, then it must be that $s_{j-2}^T = s_{i-2}^u$ and $c_{j-1} = c_{i-1}$. The latter equality can be satisfied by submitting the same signature for both rounds $i-1$ and $j-1$. Having $s_{j-2}^T = s_{i-2}^u$, requires

$$v_{j-1}^T = s_{j-2}^T = h(s_{j-3}^T, c_{j-2}) = h(s_{i-3}^u, c_{i-2}) = s_{i-2}^u = v_{i-1}^u$$

   similar to (6.3). Continuing recursively, avoiding the contradiction of a hash collision, we arrive at a requirement whereby $v_{j-k}^T = v_{i-k}^u$ when $k = i-1$. In other words

$$v^T_{j-i+1} = s^T_{j-i} = h(s^T_{j-i-1}, c_{j-i}) = IV = s^u_0 = v^u_1.$$

However, since IV was chosen such that $IV \notin \{0,1\}^l$ for an l-bit hash, it cannot be that $s^T_{j-i} = h(s^T_{j-i-1}, c_{j-i}) = IV$ for any $j > i \geq 1$.

Therefore, Protocol DT3 is D-resilient.

A similar argument can be used to demonstrate the D-resilience of Protocol DT4. Briefly, suppose that for $j > i \geq 1$,

$$v^T_j = s^T_{j-1} = h^{j-1}(IV) = h^{i-1}(IV) = s^u_{i-1} = v^u_i.$$

Then for $i \geq 2$, we have

$$h(h^{j-2}(IV)) = h(h^{i-2}(IV))$$

implying that a collision has been found for h, since $j \neq i$. If $i = 1$, we have that

$$h(h^{j-2}(IV)) = IV,$$

which cannot be true for any $j \geq 2$ since IV was chosen such that $IV \notin \{0,1\}^l$ for the l-bit hash h. Therefore, Protocol DT4 is D-resilient. ∎

6.5.3 Time Variant Requests

It is important to use a synchronization parameter for which the ordered set of all such parameters contains distinct elements, i.e., it is computationally infeasible to obtain or use the same synchronization parameter twice. This was accomplished in Section 6.5.2 using the output of a collision-resistant hash function h. In this section, we present the use of time variant parameters that, as the name implies, monotonically increase with time.

Protocol DT4 (of Section 6.5.2) implicitly used a count of the current round to specify the number of cumulative hashes of the initialization value. Protocol DT6 uses this round counter (referring to the number of secondary authentication requests) on its own as a synchronization parameter that is sent in the clear.

Protocol DT6 Using a Counter for Secondary Synchronization.

Description: This protocol replaces the protocol-specific functions as described in Protocol DT1. Initially, $i = 0$ and is incremented by 1 at the start of each round. This protocol is D-resilient (see Proposition 6.2).

Require:

```
1: $u$ and the $TR$ initially share the synchronization parameter (counter) $s_0^u = s_0^T = 0$,
   whereas prior to round $i$, assuming no fraudulent secondary requests have been
   made, they share $s_{i-1}^u = s_{i-1}^T = i - 1$.
2: During round $i \geq 1$, $u$ submits the value $c_i' = s_{i-1}^u$ to the $TR$ as a secondary
   request along with the signature $c_i$.
3: The $TR$ determines the correctness of the request by ensuring that $c_i'$ is equal
   to $s_{i-1}^T$ (the $TR$'s stored value for $u$). If equal, the $TR$ computes and stores
   $s_i^T = s_{i-1}^T + 1$ and returns $r_i = (c_i', c_i, u, sig_{TR}(c_i', c_i, u))$ to $u$.
4: $u$ verifies the mathematical correctness of $r_i$ and ensures that its contents match
   what was originally sent by $u$. If successful, $u$ computes and stores $s_i^u = s_{i-1}^u + 1$.
```

Alternatively, one can also use the time at which signatures are produced as a parameter used to synchronize u with the TR. The use of the time here is advantageous in that beyond the usefulness of allowing a synchronization, it can allow the TR to simultaneously provide a time stamp for the submitted signature as well as possessing sufficient information for the implementation of the cooling-off period described in Section 6.6.

Detecting Forged Signatures

Proposition 6.2 *Protocols DT6 and DT7 are D-resilient (see Definition 6.8).*

Proof Let $v_i^u = (i-1) = v_i^T$ be the view of both the legitimate signing user u and the TR after the ith legitimate secondary authentication request. To show D-resilience, we need to show that for no $j > i$, can an attacker X produce v_j^T such that $v_j^T = v_i^u$.

Suppose that X did find such a j, and let j be the smallest positive integer greater than i for which $v_j^T = v_i^u$. Expanding, we have

$$v_j^T = s_{j-1}^T = (j-1) = (i-1) = s_{i-1}^u = v_i^u. \tag{6.4}$$

However, this implies that $j = i$, contradicting the assumption that j is the smallest

Protocol DT7 Using the Time of Last Signature for Secondary Synchronization.

Description: This protocol replaces the protocol-specific functions as described in Protocol DT1. Initially, $i = 0$ and is incremented by 1 at the start of each round. This protocol is D-resilient (see Proposition 6.2).

Require: t_i denotes the time of receipt of the signature c_i during round i by the TR.

```
1: $u$ and the $TR$ initially share the synchronization parameter (time) $s_0^u = s_0^T = t_0$,
   whereas prior to round $i$, assuming no fraudulent secondary requests have been
   made, they share $s_{i-1}^u = s_{i-1}^T = t_{i-1} \succ t_{i-2}$ (see Definition 4.4).
2: During round $i \geq 1$, $u$ submits the value $c_i' = s_{i-1}^u$ to the $TR$ as a secondary
   request along with the signature $c_i$.
3: The $TR$ determines the correctness of the request by ensuring that $c_i'$ is equal to
   $s_{i-1}^T$ (the $TR$'s stored value for $u$). If equal, the $TR$ computes and stores $s_i^T = t_i$
   and returns $r_i = (c_i', s_i^T, c_i, u, sig_{TR}(c_i', s_i^T, c_i, u))$ to $u$.
4: $u$ verifies the mathematical correctness of $r_i$ and ensures that its contents match
   what was originally sent by $u$. If successful, $u$ computes and stores $s_i^u = s_i^T$.
```

integer strictly greater than i for which $v_j^T = v_i^u$. Therefore, Protocol DT6 is D-resilient.

Similarly for Protocol DT7, and assuming that an attacker could produce $v_j^T = v_i^u$ for $j > i$, so that

$$v_j^T = s_{j-1}^T = t_{j-1} = t_{i-1} = s_{i-1}^u = v_i^u. \tag{6.5}$$

However, since $j > i \geq 1$, then $(j-1) > (i-1)$, so that having $t_{j-1} = t_{i-1}$ contradicts the requirement that $t_{j-1} \succ t_{i-1}$. ■

6.5.4 Modification of the Synchronization Parameter

In this subsection, we consider the possibility of an attacker, already in possession of the legitimate signing user u's signature key, modifying the synchronization parameter stored by u. The possibility of such a modification is considered relative to how the token is stored as well as how predictable it is.

Parameter Storage

How a synchronization parameter is stored can depend on how "memorizable" the parameter is. Consider, for example, the respective use of a counter and time in Protocols DT6 and DT7. Ephemeral storage can be used whereby the current value of the counter is memorized by the legitimate signing user. The recognizable structure of these parameters allows for a potentially easily remembered parameter. On the other hand, schemes such as Protocol DT3 do not provide easily memorizable synchronization parameters since their value is the output of a one-way hash function.

For all of the synchronization parameters presented in Section 6.5, the storage can be maintained similar to how a password or private signature key is stored, e.g., on u's local disk, or on a hardware token. The main difference is that the privacy of the secondary parameter need not be maintained, only its authenticity. However it is stored, the storage of the synchronization parameter must be "independent" of the signature key storage so that compromise of the signature key does not simultaneously allow *modification* of the synchronization parameter. (The effect of such a modification is discussed below.) In other words, the *integrity* of the parameter must be maintained.

Malicious Parameter Modification

The maintenance of the authenticity of the secondary authentication synchronization parameter is crucial to the provision of D-resilience (see Definition 6.8). Notice that modification of the parameter, either subsequent to or coinciding with a signature key compromise, allows an attacker to submit fraudulent secondary authentication requests, and subsequently "resynchronize" the legitimate signing user u with the TR (by resetting the synchronization parameter to the value obtained by the attacker subsequent to the last fraudulent request).

A difference from the use of only first level signatures is that in such schemes, once the signature key is compromised, an attacker can continue to forge signatures until either the legitimate user detects or is informed of the compromise or the corresponding verification key is revoked or expires. For schemes incorporating a second level authentication, alteration of the synchronization parameter, allows for only a limited number of signatures. This point requires further clarification.

Consider an attacker X in possession of user u's signature key. If modification of the synchronization parameter were possible, the modification can occur either

1. prior to the forgery of any signatures, or

2. subsequent to the forgery of any signatures.

As demonstrated for the first item in the section below on 'Parameter Predictability' and for the second item in the next paragraph, so long as the legitimate signing user does not request a secondary authentication during the time that the first fraudulent secondary authentication request was made, till the time that the view of u is modified, an attack can be successful.

A parameter modification subsequent to the forgery of a signature would proceed as follows. Prior to round $i+1$, the attacker X possesses sig_u (the signature key of u) and v_i^u (the view or value of the synchronization parameter stored by u after round i) where $v_i^u = v_i^T$. X proceeds to forge signatures and submit secondary authentication requests for k rounds, after which $v_i^u = v_{i+1}^u = \ldots = v_{i+k}^u$ and u is not synchronized with the TR since $v_{i+k}^u \neq v_{i+k}^T$ based on the D-resilience of the protocol. Using an ability to modify u's synchronization parameter, X would reset v_{i+k}^u so that $v_{i+k}^u \leftarrow v_{i+k}^T$. A practical barrier to such an attack is not only the requirement of an ability to modify the parameter, but modify subsequent to the initial compromise of the signature key. This might require a physical attacker to alter u's synchronization parameter, subsequent to the signature key compromise. The forgery of additional signatures requires a subsequent modification of the parameter.

Parameter Predictability

Protocols DT6 and DT7 use parameters that are predictable. In other words, given the view v_i^u of the legitimate signing user u subsequent to round i, one can predict, with high probability, v_j^u for $j > i$. For Protocol DT6, this is trivial since $v_j^u \leftarrow v_i^u + (j - i + 1)$. For Protocol DT7, one can determine v_j^u so long as one can estimate the time of the jth secondary request. This estimation can be self-fulfilled by submitting a secondary request around the predicted time (for example, if an attacker were able to modify the parameter when compromising the signature key - see next paragraph). The determination of the exact time depends on, among other things, the granularity of the time used (is more difficult if a granularity of milliseconds as opposed to seconds is used), as well as the lag time involved subsequent to the submission of the request by u and prior to the assignment of the time by the TR.

This predictability can be used to the advantage of an attacker X, already in possession of u's signature key (assuming that u does not perform a legitimate secondary authentication request until after round $i + k$). Rather than requiring a modification of the synchronization parameter subsequent to the forgery of signatures, X can now modify u's synchronization parameter prior to the forgery. If this happens to coincide with the time of, for example, theft of the signature key, then only one occasion of theft (in which the signature key is compromised and the synchronization parameter is modified) is required by X. Such an attack would proceed for Protocol DT6, for example, as follows.

Upon obtaining sig_u from the legitimate signing user u, prior to the $(i + 1)$st round, X also modifies v_i^u so that $v_i^u \leftarrow v_i^u + k$. In this way, X can submit k secondary authentication requests, as user u, so that subsequent to the k requests, u will be synchronized with the TR.[4] One way to make the information less predictable would be for the TR to return c_i'' and n_i to u, where $c_i'' = h(n_i, c_i')$ and n_i is a random value chosen by the TR. The synchronization parameter stored by both u and the TR is the pair (c_i', n_i). This mechanism is the same as was used for the one-way function variant schemes of Section 6.5.2.

6.6 Preventing Forged Signature Acceptance

For the detection schemes described in Section 6.5.2 and Section 6.5.3, the legitimate signer u or the TR is able to detect when a fraudulent secondary request has been

[4]This attack would typically be more difficult to mount against Protocol DT7 since the time applied by the TR at a subsequent round is likely difficult to predict.

received, possibly indicating the forgery of a signature. Yet this still does not prevent the possibility that

1. u may repudiate a legitimately signed message; or
2. a recipient v may unknowingly receive a forged signature, prior to the detection of a compromise by u.

However, suppose that signed messages are, by rule, not accepted as being valid until some period of time has elapsed, i.e., a *Cooling-Off PEriod* (COPE). The purpose of this COPE is to allow for "late" forgery detections or revocations, possibly resulting from a compromise, i.e., in the case a forged signature has been detected. For example, if a message is signed on Friday, it may be part of policy to not accept the signature until Saturday. (Finer or coarser granularities may also be used.) This allows a day of grace for the owner of the private signature key to claim the possible compromise of his/her key.

However, on its own, this COPE does not preclude the possibility that a compromise is not detected until after the COPE has expired (and hence some forged signatures may have been accepted). As well, even if the compromise is detected on time, there may be a delay before the corresponding certificate is revoked (see Figure 6.1). To facilitate both items 1 and 2 above, we incorporate the COPE with a so-called *Check-In Period* (CHIP) giving CHIP/COPE.

Definition 6.9 A CHIP/COPE refers to a *check-in period* (CHIP) during which time the legitimate owner of the signing private key is required to (at least once during the period) ensure synchronization with the trusted register (TR) (e.g., by obtaining a second level authentication for a signature),[5] and a *cooling-off period* (COPE) during which time, received signatures are still considered to be temporarily unverifiable. The maximum length of time between two CHIPs is denoted $length(CHIP)$, while $length(COPE)$ denotes the minimum length of time that must elapse before a signature can be accepted as valid, subsequent to its receipt (or subsequent to a time contained in a time stamp computed for the signature).

[5]Certain scalability and denial of service issues would have to be considered in practice, related to the potential inability of a user to check in because of an overwhelmed TR.

1. The legitimate signing user u is responsible for performing a check-in, every $length(CHIP)$ time units.
2. A signature is not to be accepted until subsequent signature verification, $length(COPE)$ time units after the receipt of the signature.
3. $length(CHIP) \leq length(COPE)$.

Table 6.4: Requirements of Protocols Implementing Check-In Periods (CHIPs) and Cooling-Off Periods (COPEs); see Definition 6.9.

Remark 6.2 *A CHIP for a synchronized secondary authentication protocol $\mathcal{P}$ is simply the submission of a secondary authentication request, accompanied by a signature for which secondary corroborative evidence is required. If, before the end of the CHIP, a legitimate user does not have a signature that requires corroborative evidence, a signature for a generic message such as "This message is a simple secondary authentication message required for a check-in prior to time t" can be constructed to facilitate a check-in.*

If the length of the COPE is a single day (i.e., $length(COPE) = 24$ hours), then the legitimate user can wait no longer than 24 hours after a legitimate signing, before performing a check-in. To allow for other tasks to be performed subsequent to the detection of a compromise (cf. Figure 6.1), in practice the length of the COPE should be buffered slightly so that it exceeds the length of the CHIP.

The requirements of the CHIP/COPE are given in Table 6.4. Notice that since the legitimate owner of the signing private key is responsible for checking-in (i.e., verifying synchronization) during a given time period, he is not able to repudiate a message that was legitimately signed. This is because for signatures that have been accepted by the recipient (i.e., signature has been received and the COPE has since expired), the latency period must have passed and the loss of synchronization would have been detected for the time period in which the signature was sent. Also, forged signatures need not be accepted. The application of the CHIP/COPE with the detection of forged signatures can achieve these goals (see Proposition 6.3).

Combining a Cooling-Off Period with Detection

Definition 6.10 We say that a synchronized secondary authentication protocol is *detection-and-repudiation resilient* (*DR-resilient*) if it is D-resilient (see Definition 6.8) and if both

1. u cannot successfully repudiate legitimate signatures that have been accepted as valid by a signature recipient(s), and
2. forged signatures can be detected and rejected prior to their acceptance by an unknowing signature recipient.

■

The CHIP/COPE can be combined with Protocol DT1 to produce a DR-resilient protocol. The is captured by the following proposition.

Proposition 6.3 *Let P be a D-resilient synchronized secondary authentication protocol (as described by Protocol DT1) augmented with a CHIP/COPE (as defined in Definition 6.9). Assume that u must check-in (see Remark 6.2) every $length(CHIP)$ time units and that signatures (accompanied by second level authentication) are not accepted until $length(COPE)$ time units after receipt and given the requirements in Table 6.4. Then P is DR-resilient (see Definition 6.10).*

Proof (Outline) Let us first suppose that u could repudiate a legitimately signed message c by claiming it was forged. This would imply that $length(COPE)$ time had elapsed subsequent to the receipt of c by a recipient, and hence, that no compromise was detected nor reported through the revocation of the corresponding verification certificate. Therefore, since P is D-resilient and such a forgery would be detected by u, the last check-in by u must have been performed prior to the start of the COPE. However, since $length(COPE)$ time has subsequently elapsed and no check-in was performed by u during the COPE, then $length(CHIP) > length(COPE)$, a contradiction.

Similarly, suppose that a recipient v has accepted a forged signature c'. By the design of the COPE, c' must have been accepted at least $length(COPE)$ time units subsequent to the receipt of c' and subsequent to a determination of whether a compromise has been reported. However, since P is D-resilient, if the forged signature was not detected by the legitimate signer u, the last check-in by u must have been performed prior to the start of the COPE. Since $length(COPE)$ time has subsequently elapsed and no check-in was performed by u during the COPE, then $length(CHIP) > length(COPE)$, a contradiction.

Therefore, D-resilient synchronized secondary authentication protocols augmented with a CHIP/COPE are DR-resilient. ■

In this way, once a recipient of a signature has waited a length of time equal to the COPE (plus additional time allowing for revocation, latency delays etc.), and subsequent to a check of the revocation status of u's public key, she can be sure that the signature was legitimately constructed. The signatures are *committed* at this time, in the sense that the CHIP/COPE is similar to an *atomic* transaction or protocol. The legitimate signer must have legitimately signed a message subsequent to the signing of the message for the aforementioned user, yet before the CHIP expiry for the recipient. By designing a protocol in such a way that the legitimate user confirms that the messages signed during the last CHIP were indeed signed by him, the signing user is limited in his ability to later deny having signed any of the messages in question.

How does the use of a CHIP/COPE alter, for example, Protocol DT7? Let the length of the CHIP/COPE (see Definition 6.9) be t time units. Beyond requiring u to check-in (see Remark 6.2) with the TR at least every t units, the TR would also perform a check that $t_i - t_{i-1} < t$ (indicating that the amount of elapsed time between times t_i and t_{i-1} is less than t). So long as a recipient waits t time units before accepting a signature, forged signatures can be detected by u or the TR. As well, t may be different for each user. Allowing the recipient of a signed message to determine the length of the COPE for a particular message can be achieved by having the TR return $r_i = (c'_i, s_i^T, t, c_i, u, sig_{TR}(c'_i, s_i^T, t, c_i, u))$. Alternatively, it might be included as a parameter in the user's first level public key certificate.

Remark 6.3 *Although described as a period of waiting subsequent to the receipt of a signature, the CHIP/COPE concept can be generalized to refer to the elapse of* *the COPE subsequent to the time of stamping of the signature, so long as the time stamp is produced no later than the secondary authentication request. A simple way of achieving this time stamp is for the TR to apply a time stamp as part of the returned secondary corroborative evidence.*

Implementation and Practicality

Coordinating the CHIP with an actual user may require, for example, that "suspensions of the CHIP requirement" are allowed in the case of long-term absences by a user, e.g., possibly by placing the verification certificate "on hold" [ANS97]. Also related to the practical implementation of such a scheme is that once a lack of synchronization is detected by the TR, additional time will be required before revocation information can be obtained by signature recipients. Therefore, in practice, the length of the COPE should be $t + \epsilon$ for a suitable ϵ, where the CHIP is t time units.

With regard to the practicality of using a CHIP/COPE, imposing such restrictions on both the signer and recipient may appear unreasonable. However, there already exist examples of its use in current society (e.g., depositing a cheque normally requires a waiting period before the amount may be withdrawn from the account), it is certainly not practical for all situations. Yet there are situations in which it can be very helpful, i.e., schemes for which undetected key compromise is intolerable, yet which can tolerate a time delay before the acceptance of a signature. Such high valued transactions include major business deals, mergers and acquisitions, and real estate deals; transactions that want to use digital signatures for their convenience, but are so high-valued that they require an extra level of assurance.

Chapter 7

Concluding Remarks

In this chapter, we examine the significance of this thesis as a contribution to the field of cryptographic authentication and discuss some future directions for further research.

7.1 Positioning of Contributions

Section 1.3 provided a summary of the contributions from the more detailed results given in each chapter of this thesis. In this section, we attempt to predict the significance of these results for the study of cryptographic authentication.

The assimilation and classification of the previous work from Chapter 2 allows for a quick review of the previous work and convenient classification of new time stamping protocols. The critical analysis of this work from Chapter 3 allows one to determine the suitability of the previous time stamping protocols and permits comparisons and analysis of newly proposed protocols. Motivated by the discovery of protocol failures for two previous schemes, the time stamping framework of Chapter 4 permits the construction of a variety of sound new protocols.

Illustration of the necessity of time stamped digital signatures allows for consistent and less disputable signature verification. The notarization of digital signatures, as performed by Protocol NT2 in Chapter 5, is suitable for environments in which the validation of digital signatures by signature recipients is costly. Time stamping or notarization are useful for renewing the lifetime of a digital signature in the case that the lifetime required for the signed message's authenticity exceeds the provisions of the original authentication of the message.

The protocols of Chapter 6 allow one to enhance the legitimacy of a digital signature by providing additional corroborative evidence from a trusted authority regarding the success of an independent, second level authentication. Such a mechanism is useful, for example, for high-valued, distributed transactions in which a subsequent claim of key compromise cannot be tolerated.

In the "grand scheme of public-key cryptography", the concept of time is quite important and relevant, especially with regard to digital signatures. The origination of public key cryptography [DH76] required only the storage of a private signature key by the signing user and authentic publication of the verification key. Requirements for distribution of public keys introduced the concept of certificates [Koh78], while limiting the lifetime of these certificates introduced revocation [ITU93]. It is now clear that certificate-based digital signatures require time stamping of the signatures as well as temporally authenticated and stored certificate state information. Even further, time stamping alone does not help in the case of a key compromise that is undetected. A timeline representation of these ideas is given by Figure 7.1.

7.2 Future Work

Throughout the production of this thesis, several topics were discovered that were considered either beyond the scope or direction of the current discussion or thought better suited for future research. In this section, we briefly discuss some of these ideas.

Group Hashing. Section 2.3 reviewed several group hashing techniques that were subsequently analyzed in Section 3.2. An interesting investigation would involve the discovery of new group hashing protocols or identification and proof of some sufficient or necessary properties that a group hash function would possess. Some work in this direction has been performed by Nyberg [Nyb95]. Also of interest, especially with regard to the distribution of revocation information, would be the discovery of efficiently incremental group hash functions for which modification of a previously constructed $member_y$ is not required (if they exist).

Signature Key Lifetime. Beyond the lifetime of individual user's signature keys, the renewal of the keys of trusted authorities is an important practical concern. Should such a renewal be required in response to the compromise of a trusted authority's signature key then, as one example, the authenticity of certificates produced by that certification authority are called into question. Alternatively, consider that the compromise of a time stamper's signing private key may prevent the proper verification of signatures that had been purportedly time stamped using the compromised key, prior to the compromise. Other techniques to deal with key compromise, either of a user or trusted authority, are therefore an interesting avenue for future research.

The relationship between authorities may be helpful here as well. Suppose, for example, that a time stamp authority is issued the signature verification certificate $cert_T$ by a CA, just as the CA would for a user. Given a finite validity period for this certificate, notice that the compromise of T's signature key can limit the "reach" of forged time stamps to the period of validity of the corresponding verification certificate. This can be implemented by requiring that, during verification of a certificate, it is ensured that the time contained within the time stamp, is no later than the expiry date of $cert_T$ and no earlier than the creation date.

Network Delay There are often several factors (e.g., system components, entities) that contribute to the performance of a particular action or event. For example, recall the series of events subsequent to a user's signature key compromise, as displayed in Figure 6.1. Excessive delays resulting from any of these events diminishes the performance of a protocol (relying on the completion of each of these events) and more importantly, can lead to the improper running of a protocol, e.g., delay regarding the reporting of a user's key compromise detection may result in a recipient unknowingly accepting a forged signature. An important area of research would therefore involve the studying optimizations to the performance of various critical cryptographic events.

This is especially relevant for the implementation of the CHIP/COPE of Section 6.6. Minimizing the delay required before a user can accept a signature (i.e., minimizing the length of the COPE and length of time between CHIPs) is an important practical concern.

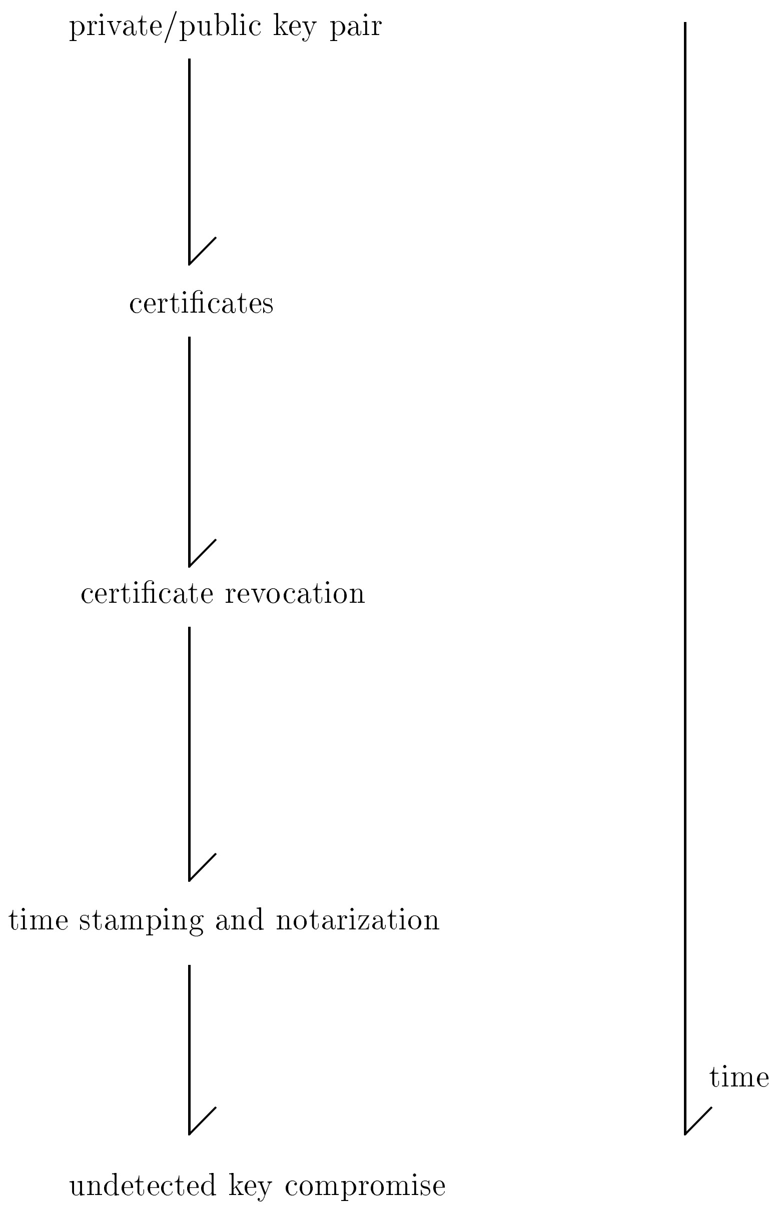

Figure 7.1: Timeline of relevant and related concepts since the origination of public-key cryptography.

Bibliography

[ACPZ98] Carlisle Adams, Pat Cain, Denis Pinkas, and Robert Zuccherato. Time stamp protocols. Internet draft (work in progress), Internet Engineering Task Force (IETF), July 1998. Available as http://www.ietf.org/internet-drafts/draft-adams-time-stamp-02.txt.

[Adl83] Leonard Adleman. Implementing an electronic notary public. In *Advances in Cryptology: Proceedings of Crypto '82*, pages 259–265. Plenum Press, 1983.

[ANS97] ANSI X9.57. Public key cryptography for the financial services industry: Certificate management. Draft standard, American National Standard for Financial Services, February 1997.

[Bar96] T. S. Barassi. The cybernotary: Public key registration, certification and authentication of international transactions. http/www.intermarket.com/ecl/notary.html, 1996. Digital Commerce Services.

[BDL97] Dan Boneh, Richard A. Demillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults. In *Advances in Cryptology: Proceedings of Eurocrypt '97*, pages 37–51. Springer-Verlag, 1997.

[BdM91] Josh Benaloh and Michael de Mare. Efficient broadcast time-stamping. Technical Report TR 91-1, Clarkson University, Department of Math and Computer Science, 1991.

[BdM93] Josh Benaloh and Michael de Mare. One-way accumulators: A decentralized alternative to digital signatures. In *Advances in Cryptology: Proceedings of Eurocrypt '93*, pages 274–285. Springer-Verlag, 1993. Also appeared as Clarkson University Technical Report TR-MCS-93-1, April 1993.

[BGG94] Mihir Bellare, Oded Goldreich, and Shafi Goldwasswer. Incremental hashing: The case of hashing and signing. In Yvo G. Desmedt, editor, *Advances in Cryptology: Proceedings of Crypto '94*, pages 216–233. Springer Verlag, 1994.

[BHS93] D. Bayer, S. Haber, and W.S. Stornetta. Improving the efficiency and reliability of digital time-stamping. In *Sequences II: Methods in Communication, Security and Computer Science*. Springer-Verlag, 1993.

[Ble96] Daniel Bleichenbacher. Generating ElGamal signatures without knowing the secret key. In *Advances in Cryptology: Proceedings of Eurocrypt '96*, pages 10–18. Springer-Verlag, 1996.

[BLLV98] Ahto Buldas, Peeter Laud, Helger Lipmaa, and Jan Villemson. Time-stamping with binary linking schemes. In *Advances in Cryptology: Proceedings of Crypto '98*. Springer-Verlag, 1998.

[CHH97] R. Canetti, S. Halevi, and A. Herzberg. Maintaining authenticated communication in the presence of break-ins. In *Proceedings of the 16th Annual ACM Symposium on Principles of Distributed Computing*, pages 15–24, 1997.

[Cus87] Charles Cushing. *Cushing's Notarial Form Book, with a Treatise or Historical Outline of the Notarial Profession.* A. Periard, Montréal, Québec, 1887.

[Des94] Yvo Desmedt. Threshold cryptography. *European Transactions on Telecommunications*, 5(4):449–457, July 1994.

[DH76] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, November 1976.

[Dif82] Whitfield Diffie. Conventional versus public key cryptosystems. In Gustav Simmons, editor, *Secure Communications and Asymmetric Cryptosystems*, pages 41–72. Westview Press Inc., Boulder, Colorado, 1982. (Based on a paper presented at a 1980 symposium. See [Sim82].).

[DP84] D. Davies and W. Price. *Security for Computer Networks.* John Wiley & Sons, 1984.

[DS93] C. Dyreson and R. Snodgrass. Timestamp semantics and representation. *Information Systems*, 18(3):143–166, 1993.

[FB97] Warwick Ford and Michael Baum. *Secure Electronic Commerce: Building the infrastructure for digital signatures and encryption.* Prentice Hall PTR, Upper Saddle River, New Jersey 07458, 1997.

[FIP94] FIPS 186. Digital signature standard. Federal Information Processing Standards Publication 186, U.S. Department of Commerce/N.I.S.T., National Technical Information Service, Springfield, Virginia, 1994.

[FIP95] FIPS 180-1. Secure hash standard. Federal Information Processing Standards Publication 186, U.S. Department of Commerce/N.I.S.T., National Technical Information Service, Springfield, Virginia, April 1995. (supersedes FIPS PUB 180).

[GC98] N. Gershenfeld and I. Chuang. Quantum computing with molecules. *Scientific American*, 1998.

[HJJK97] A. Herzberg, M. Jakobsson, S. Jarecki, and H. Krawczyk. Proactive public key and signature systems. In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, 1997.

[HKS95] Stuart Haber, Burt Kaliski, and W. Scott Stornetta. How do digital timestamps support digital signatures? *CryptoBytes*, 1(3), Autumn 1995. (Available from `http://www.rsa.com/rsalabs/pubs/cryptobytes.html`.).

[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 *Proceedings of the 4th ACM Conference on Computer and Communications Security*. ACM Press, April 1997.

[Ill91] Illinois notary act. http://www.notaryexpress.com/handbook.html, 1991.

[ITU93] ITU-T Recommendation X.509. The directory - authentication framework. Technical report, International Telecommunication Union, Geneva, Switzerland, November 1993. (equivalent to ISO/IEC 9594-8:1990&1995).

[Jus98] Mike Just. Some timestamping protocol failures. In *Proceedings of the 1998 Symposium on Network and Distributed System Security*, pages 89–96, March 1998.

[JvO98] Mike Just and Paul C. van Oorschot. Addressing the problem of undetected signature key compromise. Technical Report TR-98-06, Carleton University, School of Computer Science, June 1998. To appear in the *Proceedings of the 1999 Symposium on Network and Distributed System Security.*

[Kan86] H. Kanare. *Writing the Laboratory Notebook*, chapter 6. American Chemical Society, 1986. (2nd printing).

[KJJ98] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. http://www.cryptography.com/dpa/, 1998.

[Koc96] Paul Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Advances in Cryptology: Proceedings of Crypto '96*, pages 104–113. Springer-Verlag, 1996.

[Koc98] Paul Kocher. A quick introduction to certificate revocation trees (CRTs). http://www.valicert.com/resources/whitepaper/bodyIntroRevocation.html, 1998.

[Koh78] L. M. Kohnfelder. Toward a practical public-key cryptosystem. B.Sc. Thesis, MIT Department of Electrical Engineering, 1978.

[Lam81] L. Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24:770–772, 1981.

[Lan95] Susan K. Langford. Threshold DSS signatures without a trusted party. In Don Coppersmith, editor, *Advances in Cryptology: Proceedings of Crypto '95*, pages 397–409. Springer-Verlag, 1995.

[LB92] Kwok-Yan Lam and Thomas Beth. Timely authentication in distributed systems. In Y. Deswarte, G. Eizenberg, and J.-J. Quisquater, editors, *2nd European Symposium on Research in Computer Security (ESORICS'92)*, pages 293–303. Springer-Verlag, November 1992.

[Mer80] Ralph Merkle. Protocols for public-key cryptosystems. In *Proceedings of the 1980 IEEE Symposium on Security and Privacy*, April 1980.

[Mer82] Ralph Merkle. Protocols for public-key cryptosystems. In Gustav Simmons, editor, *Secure Communications and Asymmetric Cryptosystems*, pages 73–104. Westview Press Inc., Boulder, Colorado, 1982. See [Sim82]. A more detailed version of [Mer80].

[Mer98] Merriam-webster online dictionary. http://www.m-w.com/dictionary.htm, 1998.

[MM82] C. Meyer and S. Matyas. *Cryptography: A New Dimension in Computer Data Security.* John Wiley & Sons, 1982.

[MQ97] Henri Massias and Jean-Jacques Quisquater. Time and cryptography. Technical Report WP1, Université Catholique de Louvain, March 1997.

[MvOV97] Alfred Menezes, Paul C. van Oorschot, and Scott Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1997.

[Nyb95] Kaisa Nyberg. Commutativity in cryptography. In *Proceedings of the First International Workshop on Functional Analysis*, Trier University, Berlin, 1995. Walter de Gruyter & Co.

[Nyb96] Kaisa Nyberg. Fast accumulated hashing. In Dieter Gollmann, editor, *Fast Software Encryption*, pages 83–87, Cambridge, UK, February 1996.

[PF96] F. Pinto and V. Freitas. Digital time-stamping to support non repudation in electronic communications. In MCI (Manifestations and Communications Internationales), editors, *Proceedings SECURICOM '96 - 14th Worldwide Congress on Computer and Communications Security and Protection*, pages 397–406, CNIT, Paris, France, June 1996. (Available from `http://marco.uminho.pt/CCG/ccom-pub.html`.).

[PK79] Gerald J. Popek and Charles S. Kline. Encryption and secure computer networks. *Computing Surveys*, 11(4):332–356, December 1979.

[Pom90] Carl Pomerance. Factoring. In Carl Pomerance, editor, *Cryptology and Computational Number Theory*, pages 27–47. American Mathematical Society, 1990.

[Riv92] Ronald L. Rivest. The MD5 message-digest algorithm. Internet Request for Comments (RFC) 1321, April 1992. Also presented at Rump Session of Crypto'91.

[RS97] Muhammad Rabi and Alan Sherman. An observation on associative one-way functions in complexity theory. *Information Processing Letters*, 64(5):239–244, December 1997.

[RSA78] Ron Rivest, Adi Shamir, and Len Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978.

[Sha81] Adi Shamir. On the generation of cryptographically strong pseudo-random sequences. In *Proceedings of ICALP*, pages 544–550, 1981.

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings of the* 26*th Symposium on Theory of Computing (STOC)*, pages 124–134, Montreal, Canada, 1994.

[Sim82] Gustavus J. Simmons, editor. *Secure Communications and Asymmetric Cryptosystems.* Westview Press, Inc., Boulder, Colorado, 1982. This book is based on a symposium held at the 1980 American Association for the Advancement of Science (AAAS) National Annual Meeting in San Francisco, California.

[Sti95] Doug Stinson. *Cryptography: Theory and Practice.* CRC Press, 1995.

[Tro95] D. Trowbridge. Imagine a notary stamp for electronic documents. *Computer Technology Review*, XV(4), April 1995.

Appendix A

A Historical Review of Notarization

The term *notary* is taken from the *notary public* whose responsibilities within the United States are to witness documents and administer oaths (we refer here to the physical entity as opposed to a digital one). The traditional witnessing of physical documents involves the verification of the identity of the individual signing the document (see [Ill91]). The *digital notary* (notary agent in [MvOV97]) can have a greater range of powers, similar to the overseas conception of a notary public. Such powers include establishing the truth of statements ([Bar96, MvOV97]). A notary can also, for example, implement a time stamping scheme.

In this appendix, we review the role a notary public. Definitions related to the more recent digital incarnation are also reviewed. Further examinations regarding the role of a notary appear in Section 5.4.

A.1 Notaries Public

> Notaries are public officers appointed to prepare and execute deeds and contracts to which the parties desire or are bound to impart that character of authenticity which is attached to acts entered into under public authority; to assure their date, to preserve them, and to deliver copies

thereof, or authentic extracts therefrom [Cus87].

The need for notaries arose from the concept of ownership, around 5000 years ago. The growth of land settlements, materials and commerce increasingly necessitated a need for proprietary attachments. Since many people lacked the ability to write, a designated individual was usually appointed the responsibility. As well, the traditional oral contract did not always allow for simple dispute resolution. It is believed that the "Babylonians are regarded as the first who introduced the customs of passing private deeds in writing." [Cus87]

The original "notary" was essentially a simple scribe responsible for the recording of information. As a result of cost concerns and the fact that writing was not considered an honorable task, many of the original scribes were slaves. The earliest mention of some form of notary comes from the Roman Empire. Various titles were given: Scribæ (responsible for maintenance of public records); Tabularii (writers on tablets); Notarii (denotes user of abbreviation or notes); Cursores (uses rapid writing); Logographi (a sort of shorthand writer or stenographer); Testamentarii (writers of wills); and Argentarii (works with monetary contracts). The functions of the scriba are similar to the current *prothonotary* (derived from the Greek word *protos* (meaning first) and Latin *notarius*, refers to the chief notary though current meaning is the chief clerk for various courts of law).

It was not until the 5th Century that citizens other than slaves were allowed to function as notaries. These so-called *tabellions* were given far greater powers than their counterparts, so that "the Scribæ, Tabularii, Cursores and Notarii became their clerks." [Cus87]. Though, even at that time the tabellion did not provide for authentication. "Although binding on the parties, the acts of tabellions were not authentic or executory until verified or compared, and to avoid the trouble of verification they were published or insinuated in court." [Cus87]

Further change took place from the 13th to 15th centuries in countries such as France and England. Several varieties of notaries were appointed (e.g., by the Pope or Archbishops) though only those appointed by the King had the power to authenticate writings with their seal [Cus87, page xiv]:

[The] seal was the authentic sign of the authority given by the king to the deeds passed by his officers; so that, when an act was sealed it had an *execution parée*, that is, it was executory without any judicial order or sentence.

The role of the tabellion was limited to the recording of information, "and did not affect the authenticity of notorial acts." [Cus87]

A.2 Digital Notary

Most of the current interpretations (since 1979) regarding the role of a so-called digital notary are derived from their physical ancestor. However, there is consistent confusion equating a notary with only a simple time stamper. The notary, as opposed to a time stamper, does more than simply authenticate the time at which a statement was made.

As indicated at the start of Chapter 2, Popek and Kline [PK79, page 353] acknowledge David Redell for first suggesting the use of a notary public in the digital world. Though the term *notary* was not used explicitly, Merkle [Mer80, Mer82] discusses so-called *witnessed digital signatures* where a witness that was *a priori* agreed upon between parties A and B "physically confirms that A signed message m [by computing] sig_W('I, W, physically saw A agree to and sign message m')".

Differing from above, the following definitions equate a notary to a time stamper. Diffie [Dif82] discusses "a *digital notary public* which dates the document and signs the date with its own private key" as a solution to the problem of contract signing between two untrusting parties. The notary public, according to Diffie, allows the receivers of signed messages to protect themselves from the compromise of the signer's key. According to Adleman [Adl83], "[t]he function of a *notary public* is to certify that an 'event' took place at a particular time and place." Stinson [Sti95, page 254] defines an *electronic notary public* as a *trusted time stamping service.*

Menezes *et al.* [MvOV97] renewed the concept of differing roles between a notary and a time stamper. Whereas a *time stamp agent* is "used to assert the existence of a specified document at a certain point in time, or affix a trusted date to a transaction or digital message", a *notary agent* is "used to verify digital signatures at a given point in time to support non-repudiation, or more generally establish the truth of any statement (which it is trusted on or granted jurisdiction over) at a given point in time." [MvOV97, page 550] They go on further to point out that a "*time stamping service* [...] is a document certification or document notarization service. A *notary service* is a more general service capable not only of ascertaining the existence of a document at a certain time, but of vouching for the truth of more general statements at specified points in time." [MvOV97, page 582]

This sentiment is further echoed by Barassi [Bar96]. Beyond fulfilling various duties performed by a physical notary (and extending even further to aid in international agreements), an electronic notary or "CyberNotary" has three responsibilities:

1. *Attestations, oaths and declarations.* Among other things, digitally attesting to the signature produced by a requesting party;

2. *Certification.* Beyond the witnessing of a signature, yet short of a (legal) authentication, a certification may involve, for example, ensuring the proper translation of a particular document;

3. *Legal Validity.* This involves validating "not only the legality of the message, but also its conformity to the norms of electronic commercial practice."[Bar96][1]

[1]Rather than using the term 'legal validity', Barassi [Bar96] used the term 'authentication.' We avoid this use of the term to prevent confusion with forms of cryptographic authentication.