Efficient Long-Term Validation of Digital Signatures

Arne Ansper[1], Ahto Buldas[1], Meelis Roos[2], and Jan Willemson[2]

[1] Cybernetica; Akadeemia 21, Tallinn, Estonia
[2] Cybernetica, Tartu Lab; Lai 36, Tartu, Estonia
{arne,ahtbu,mroos,jan}@cyber.ee

Abstract. Digitally signed documents (e.g. contracts) would quickly lose their validity if the signing keys were revoked or the signature scheme was broken. The conventional validation techniques have been designed just for ephemeral use of signatures and are impractical for long-term validation. We present a new scheme that: (1) provides fast revocation while giving no extra power to on-line service providers; (2) supports long-term validation; (3) is lightweight and scalable.

1 Introduction

Bob lends Alice \$100. Alice signs a promissory note stating that she owes Bob \$100. It is in Bob's interest that Alice's digital signature is sufficient to convince a judge that she really owes Bob the money. A dispute in court, however, may take place long after the promissory note was signed. Alice's public key certificate may already be revoked. Therefore, it is necessary to ensure that *electronic documents retain provable authenticity long after their creation.*

The evidentiary function is one of the essential properties of handwritten signatures and thereby it may seem natural to assume that digital signatures have this property as well. However, no human being is able to compute digital signatures by heart and it is also hard (if not impossible), with today's technology, to create a signature device which (from the signer's point of view) is absolutely safe to use. On the one hand, it seems unfair to make the owner of a signature device (Alice) liable for everything ever signed with that device. On the other hand, if the burden of proof is completely on the side of the interested party (Bob), no one would trust digital signatures. The best we can do today is to minimize the technological risks and to define some reasonable (as fair as possible) rules for solving disputes on digital signatures, rules that all parties involved in a dispute are aware of. Among many other things, these rules must define what kind of digital information is considered to be admissible evidence in a court of law.

Secure signature devices, fair rules of liability, and up-to-date public key information are the basic problems to be solved before we can seriously speak about the evidentiary function of digital signatures. In this paper, we address

[0] All authors were supported by the Estonian SF, grant no. 4150

the last issue, i.e. the problem of efficient distribution and storage of public key information for long-term use. The secure signature devices and the liability problems are not discussed, though, we do not claim they are unimportant or easy to solve.

The first solution in the area of public-key distribution was proposed as early as in 1976. In a pioneering paper [6], the fathers of public-key cryptography – Diffie and Hellman – proposed a modified (on-line) telephone book listing public keys instead of telephone numbers. However, this idea was despised two years later by Loren Kohnfelder [11], the father of public key certificates, due to concern that managing such a telephone book would cause a communication bottleneck. The idea of individually signed certificates seemed far more attractive because the public networks were far from their today's overall availability.

Recently Gassko, Gemmell and MacKenzie [8] took a step towards reviving the modified telephone-book idea proposed by Diffie and Hellman. They showed that for long-term validation it is more efficient to manage public key databases rather than to use individually signed certificates.

In this paper, we will take another step by showing that (using modern cryptographic techniques) we can improve both the efficiency and reliability of on-line validation techniques. We introduce a new *notary service* which confirms the validity of the certificate at the time when the relevant digital signature was created. Instead of indicating the specific time, the *notary confirmation* (signed by the notary) provides a direct (one-way) link between the certificate and the signature. In order to avoid the need to use computationally expensive digital signatures extensively, the notary uses Merkle authentication trees [12,13] to compress answers to questions which clients have submitted during each minute (or some other unit) of time. After each minute, the notary signs the root of the tree. This trick (used already in time-stamping [9,3] and certification [8]) significantly reduces the number of signature creations. We also propose methods how to reduce the role of Trusted Third Parties in long-term validation and thereby improve the reliability of these services.

In Section 2, we describe our model of signature validation scheme including the list of involved parties. We also describe the main goals of this paper. In Section 3, we give a brief overview of the basic principles of public key validation techniques and discuss the main advantages and drawbacks of the on-line validation techniques. In Section 4, we outline a new on-line validation protocol that supports both fast revocation and long-term validation. In Section 5, we discuss how to increase the reliability of on-line validation schemes and present a new scheme with lower trust assumptions.

2 Statement of the Problem

In this section, we give a brief overview of the problems we try to solve and, in order to avoid misunderstandings, also emphasize some problems we *are not* trying to address in this paper. There are five parties involved in our model of signature validation:

- *Certification Authority* (CA) issues public key certificates that bind public keys to identities of their owners.
- *Signer* (Alice, A) is a person who, after obtaining a public-key certificate from the CA, creates a digital signature.
- *Prover* (Bob, B) or *Interested party* is a party who receives a digital signature from the signer and who is interested in preserving the evidentiary function of this signature.
- *Confirmer* (or *Notary*, N) represents a service for obtaining a confirmation that the certificate was valid at the moment the signature was created.
- *Verifier* (or *Judge*) is a party who needs to be convinced (using the signature, the certificate, and the confirmation) about the validity of the signature (possibly, long after the signature was created).

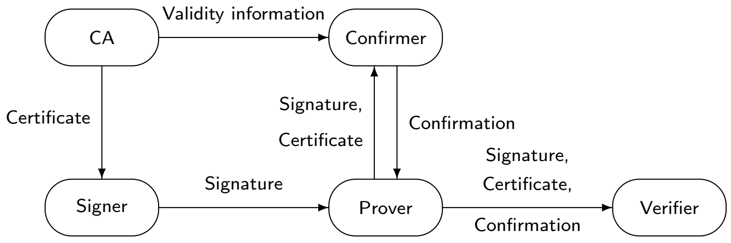

Fig. 1. Our model of signature validation.

We consider that the signature device used by the signer (Alice) is never absolutely secure and may leak the key or be stolen by an adversary. Thereby, the system we use must allow the signer to revoke the certificate immediately when the key or device is suspected of being compromised. Moreover, the Public Key Infrastructure we will develop should guarantee that the revocation notice is (timely) distributed (or made available) to all potential verifiers.

We assume that a signature is considered to be *valid* if it is proved to be created before the corresponding certificate was *revoked*. That is what we call *long-term validation* in this paper. We neither discuss how to develop secure signature devices nor give any hints for the signer how to notice the compromise of these devices. At the same time, we agree that both of these are important practical issues. The problems we are going to address are

1) *how to revoke the certificate timely* if its compromise has been noticed or even suspected; and
2) *how to prove that the certificate was not revoked.*

We *do not* claim that answering to these questions is sufficient to support (or even come close to) *non-repudiation* – the ability to prove (in undeniable manner) to a third party (e.g. a judge) that a document which was received via a public network was, in fact, written and sent by the claimed originator. This property is never assumed to hold even in the case of "true" signatures.

The more specific questions we address in this paper are

1) efficient measures for obtaining and preserving the proofs of validity (compactness of proofs, etc.);
2) computational complexity of obtaining the proofs which has a considerable influence on the prime cost of the validation service;
3) reliability of the service.

3 Existing Solutions and Their Shortcomings

3.1 Traditional Off-Line PKI Solutions

use public directory (that plays the role of the Confirmer) to store the public key information. The directory is updated by the CA in a regular basis (say, daily). The content of the directory is protected with the CA's signature and also with a time stamp in order to guarantee the freshness of the content. The Prover downloads the most recent data from the directory and uses this data later as a part of the validity proof. The verifier checks that the signer's public key certificate is not expired and is not included in the list of revoked certificates. Note that it is necessary to check whether the signature itself was formed at the time the public key information was considered as valid. Therefore, also the signature must be provided with a time stamp in order to prevent back-dating attacks.

The main advantage of this approach is that the Confirmer is a passive intermediary that has no power to modify the validity information. Therefore, only the CA must be assumed to be trusted.

The main shortcoming of this approach, considering the long-term validation, is that if the private key is suspected of being compromised Alice is unable to revoke her certificate immediately, because nobody would know about the revocation until the next update of the directory is issued. On the one hand, it is unfair (from the Alice's point) to consider as *valid* all the signatures given before the next update. Alice should not be liable for events it has no control of. On the other hand, if they are considered as invalid, it would be dangerous (for Bob) to accept any signature before the next update is issued.

Another shortcoming is that if the number of certificates grows, the size of a confirmation (as a linear function of the number of certificates issued by the CA) may become impractically large.

3.2 On-Line Methods

use an intermediary that, having received a request containing the certificate, replies with a *validity confirmation.* The integrity and content of the confirmation is protected with the signature of the Confirmer. As in traditional solutions, the confirmation comprises the date of validity. Generally, a on-line validation protocol runs as follows:

$$\boxed{\begin{array}{l} 1.\ A \to B\text{: } \mathsf{Cert}_A,\ \mathsf{Sig}_A\{X\} \\ 2.\ B \to N\text{: } \mathsf{Cert}_A \\ 3.\ N \to B\text{: } \underbrace{\mathsf{Valid}(\mathsf{Cert}_A), [t_0, t_1]}_{\mathsf{Status}_A}, \mathsf{Sig}_N\{\mathsf{Status}_A\} \end{array}} \tag{1}$$

where t_0 and t_1 denote the bounds of the validity period of the confirmation (not the certificate), and $\mathsf{Valid}(\cdot)$ denotes the validity statement. For example, the status of the certificate may be confirmed as valid, revoked, not revoked, suspended, etc. Obviously, the validity statement should directly point to the certificate it confirms. The *On-line Certificate Status Protocol* (OCSP) [14] developed by the PKIX working group is a typical example of an on-line validation protocol. The short-lived certificate approach [7] is almost equivalent.

For the persistent use of the validity confirmation we need a time stamp t on $\mathsf{Sig}_A\{X\}$, issued by a trusted *Time-Stamping Authority* (TSA) [2,9,3,5], such that $t_0 \le t \le t_1$. Therefore, provided that the public keys of CA, N and TSA are obtained authentically, the verifier should check that:

– $\mathsf{Sig}_A\{X\}$ is properly verifiable using the public key written in Cert_A;
– Cert_A is signed by the CA;
– Status_A confirms the validity of Cert_A at $[t_0, t_1]$;
– Status_A is signed by N;
– the time stamp confirms that $\mathsf{Sig}_A\{X\}$ was created at t, where $t_0 \le t \le t_1$;
– the time stamp is signed by the TSA.

The main advantage of this approach over the off-line approach is that the validity information is always up-to-date. We are able to overcome the inconvenient trade-off between the latency of revocation and the service time, because revocation messages can be sent directly to the Confirmer. An example of using secure revocation notes is given by Rivest [15].

However, there are also several shortcomings in this approach. First, the Confirmer must extensively use a digital signature scheme, which is time-consuming and may require special hardware accelerators when the number of requests becomes large. Second, we introduce two additional trusted parties: (1) the Confirmer, and (2) the TSA. Both parties are, in principle, able (without cooperation) to cheat clients.

3.3 Notary Protocols

eliminate the need for a trusted TSA and, therefore, reduce the number of additional third parties to one. Instead of using time stamps, notary protocols give a direct (one-way) link between the signature to be confirmed and the confirmation. Notary protocols are similar to the conventional validation protocols. The main difference is that the notary confirmation comprises the signature the confirmation is about. The protocol itself goes through the following steps:

$$\boxed{\begin{array}{l} 1.\ A \to B\text{: } \mathsf{Cert}_A,\ \mathsf{Sig}_A\{X\} \\ 2.\ B \to N\text{: } \mathsf{Cert}_A,\ \mathsf{Sig}_A\{X\} \\ 3.\ N \to B\text{: } \underbrace{\mathsf{Valid}(\mathsf{Cert}_A), \mathsf{Sig}_A\{X\}}_{\mathsf{Status}_A}, \mathsf{Sig}_N\{\mathsf{Status}_A\} \end{array}} \qquad (2)$$

Note that the Notary does not check the validity of users' signatures. A party who verifies the signature would check for the correctness anyway. Thereby, the signature verification by the Notary would give no additional assurance. This fact was noticed by Roos [16].

Provided that the public keys of CA and N are obtained authentically, the verifier should check that:

– $\mathsf{Sig}_A\{X\}$ is properly verifiable using the public key written in Cert_A;
– Cert_A is signed by the CA;
– Status_A confirms the validity of Cert_A and comprises the signature $\mathsf{Sig}_A\{X\}$;
– Status_A is signed by N;

Time stamps are unnecessary because $\mathsf{Status_A}$ directly points to the signature $\mathsf{Sig}_A\{X\}$ to be confirmed. Thereby, we have a proof that $\mathsf{Sig}_A\{X\}$ was created before N signed the confirmation.

Most notary protocols proposed to date are computationally expensive. For example, in the Data Validation and Certification Service (DVCS) [1] (PKIX Working Group) the service provider validates users' signatures and checks the full certification path from the certificate's issuer to a trusted point. Thereby, the processing of requests requires large amount of computation. The notary protocol (2) described in this section is much less expensive because request processing does not require extensive computations. However, each request still needs a signed reply. Therefore, we still have the same efficiency concerns as in the case of OCSP.

Our main goal is to develop a signature validation scheme capable of supporting (1) long-term validation, (2) scalability of services and (3) accountability (trust elimination). In the next sections, we discuss some hints that help us to achieve these goals.

4 A New Efficient Notary Protocol

As noticed above, in conventional on-line validation protocols, the server must create one signature per request. Hence, it must be as powerful as all the signers altogether. This can constitute a serious computational bottleneck if the users community grows large. To avoid this problem, we could give one signature to several responses at a time. The server collects all requests obtained during some time interval (referred to as *round*). It then prepares the set S of the corresponding response statements and organizes it as a certain data structure. Instead of signing all the statements independently, the server signs only a short digest $d = D(S)$ of the whole data structure. Note that it is impractical to sign a plain list of all requests because this list must be sent back to all clients, which may cause undesirable communication costs. Merkle authentication tree described in the next subsection turns out to be a suitable data structure for this purpose.

4.1 Merkle Authentication Trees

The Merkle authentication tree [12,13] is one example of a suitable data structure. The Merkle authentication tree for a set S of data items is a labeled tree the leaves of which are labeled with the elements of S and each non-leaf of which is labeled with a hash of the labels of its child-vertices (Figure 2). The digest $d = D(S)$ is defined to be equal to the label of the root vertex.

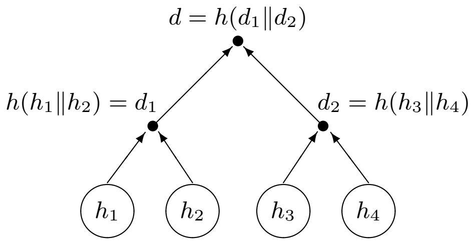

Fig. 2. Merkle tree for $S = \{S_1, S_2, S_3, S_4\}$. Here $h_i = h(S_i)$ for $i = 1...4$.

Each element $S_i \in S$ can be provided with a succinct membership proof $P(S_i, S)$ of the statement $S_i \in S$. Each proof is a formal expression of one variable x. For example (Figure 2), $P(S_2, S)(x) := h(h(h_1, x), d_2)$, where h denotes the hash function used. For verifying the statement $S_i \in S$ one needs to check the equation $P(S_i, S)(h(S_i)) = d = D(S)$. The most important property of Merkle authentication trees is that the size of a proof is $O(\log |S|)$.

4.2 Notary Protocol with Merkle Trees

The notary server works in rounds. Suppose that the duration of these rounds is set to one minute. Instead of answering the requests immediately the server collects the requests submitted during the round. It then organizes the corresponding confirmations as a Merkle tree and signs the root hash of this tree. The protocol with Merkle tree runs as follows:

$$\boxed{\begin{array}{l} 1.\ A \to B\text{: } \mathsf{Cert}_A,\ \mathsf{Sig}_A\{X\} \\ 2.\ B \to N\text{: } \mathsf{Cert}_A,\ \mathsf{Sig}_A\{X\} \\ 3.\ N \to B\text{: } \underbrace{\mathsf{Valid}(\mathsf{Cert}_A), \mathsf{Sig}_A\{X\}}_{\mathsf{Status}_A}, P(\mathsf{Status}_A, S), \mathsf{Sig}_N\{D(S)\} \end{array}} \quad (3)$$

Hence, a replay to a request is a triple $(\mathsf{Status}_A, P(\mathsf{Status}_A, S), \mathrm{Sig}\{D(S)\})$. No matter how many requests we have, only one signature per round is required. Such an approach was first used in time-stamping [4,3]. The verification procedure is almost the same as in Protocol (2), except that instead of the final signature verification the verifier

– checks the equation $P(\mathsf{Status}_A, S)(h(\mathsf{Status}_A)) = D(S)$;
– checks that $D(S)$ is signed by N.

Note that this approach has sense only if the request processing itself is computationally much cheaper than signing. This is certainly the case for the PKIX time-stamping and OCSP protocols (though, they do not use this approach!), but is not true for DVCS because each request requires several signature verifications.

4.3 Efficiency Calculations

Let T_h and T_s denote the time needed for hashing and for signing, respectively. Let T_p denote the time which is needed to find an answer to a request. In conventional on-line protocols (OCSP, TSP etc.), serving R requests requires $R \cdot (T_p + T_s)$ time-units (Figure 3). In the hash-tree protocol (Figure 4), if there

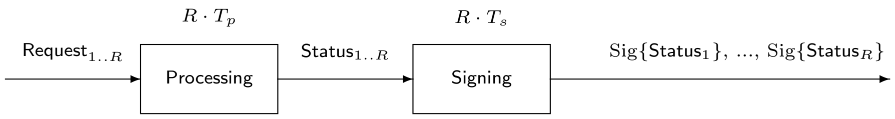

Fig. 3. Computations of the Notary in the conventional protocol.

are R requests in one round, we need $R \cdot T_p$ time units to find the corresponding answers, $R \cdot T_h$ time units to compute the hash-tree and find the digest d and proofs $P(\mathsf{Status}_i, S)$, and finally, T_s time units to sign the digest $D(S)$. Assuming

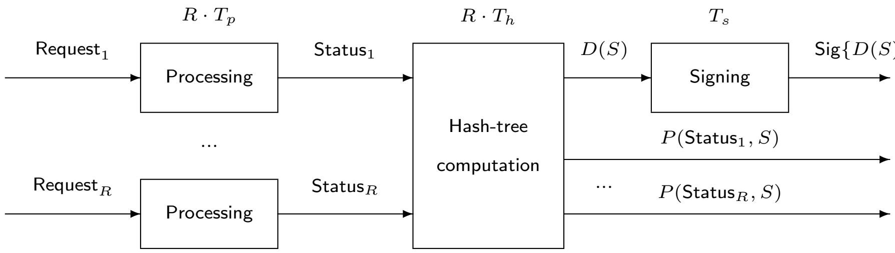

Fig. 4. Computations of the Notary in the protocol with hash-tree.

that hashing is K times faster than signing ($T_s = K \cdot T_h$) and request processing is P times faster than signing ($T_s = P \cdot T_p$), we get that the hash-tree scheme is

$$\lambda = \frac{R \cdot T_p + R \cdot T_s}{R \cdot T_p + R \cdot T_h + T_s} = \frac{1 + \frac{1}{P}}{\frac{1}{P} + \frac{1}{K} + \frac{1}{R}} \tag{4}$$

times faster. If the processing is much faster than signing (i.e. $T_p \ll T_s$), we have that $\lambda \approx KR/(K + R)$. In OCSP, TSP and the notary protocol, this is indeed the case. Chosen values of λ in that case are shown in Figure 5. For example, if $K = 10,000$ [8] and $R = 1000$, we get that our scheme is nine hundred times faster and if $R = 100$, it is still about one hundred times faster (Fig.5). For example, in OpenSSL (ver 0.94) library 1024-bit RSA with private

λ	$R = 1$	$R = 10$	$R = 100$	$R = 1,000$	$R = 10,000$
$K = 1$	0.5	0.91	0.99	≈1	≈1
$K = 10$	0.91	5.0	9.1	9.9	≈10
$K = 100$	0.99	9.1	50	91	99
$K = 1,000$	≈1	9.9	91	500	910
$K = 10,000$	≈1	≈10	99	910	5000

Fig. 5. Chosen values of λ, provided that $T_p \ll T_s$.

exponent is about 3300 times slower than the hash function SHA-1 with a single 512-bit input block. We see (Figure 5) that the main advantage of our scheme – lower computational cost – is obvious when the number of requests per round becomes large. Therefore, using longer rounds would lower the prime costs of validation services. However, duration of rounds should not exceed the limits of a reasonable service delay.

4.4 Practical Remark: Signatures with Hash Chains

Regardless of the fact that the notary protocol (3) with Merkle trees may increase the efficiency of numerous existing on-line protocols (OCSP, TSP etc.), the standards describing these protocols would be changed in order to use this advantage in practice. A way to overcome such a concern is to define new Object Identifiers (`OID`) for *signatures with hash chains*. Indeed, in the protocol (3) the proof $P(\mathsf{Status}_A, S)$ and the "ordinary" signature $\mathsf{Sig}_N\{D(S)\}$ may be viewed as a new type of signature on the message Status_A, i.e.

$$\mathsf{Sig}'_N(\mathsf{Status}_A) := (P(\mathsf{Status}_A, S),\ \mathsf{Sig}_N\{D(S)\}). \tag{5}$$

If these new signature schemes are supported by the cryptographic libraries, the protocols (2) and (3) are identical from the software engineer's point of view.

4.5 Notary Protocols in Multi-level PKI

Notary protocols (3) with hash-trees and also the signatures with hash-chains (5) allow using on-line validation in the hierarchical multi-level Public Key Infrastructures. The signatures of a Notary service provider (N) on the validity confirmations may be confirmed by a higher level Notary ($\bar{N}$). Such a hierarchical validation scheme allows us to reduce the question of the validity of a certificate to the question of the reliability of a single top-level notary server ("supervised" by a top-level CA). The confirmation protocol runs as follows:

$$\boxed{\begin{array}{ll}
1.\ A \to B: & \mathsf{Cert}_A,\ \mathsf{Sig}_A\{X\} \\
2.\ B \to N: & \mathsf{Cert}_A,\ \mathsf{Sig}_A\{X\} \\
3.\ N \text{ finds}: & \underbrace{\mathsf{Valid}(\mathsf{Cert}_A), \mathsf{Sig}_A\{X\}}_{\mathsf{Status}_A} \\
4.\ N \to \bar{N}: & \mathsf{Cert}_N,\ \mathsf{Sig}_N\{\mathsf{Status}_A\} \\
5.\ \bar{N} \to N: & \underbrace{\mathsf{Valid}(\mathsf{Cert}_N), \mathsf{Sig}_A\{\mathsf{Status}_A\}}_{\mathsf{Status}_N}, \mathsf{Sig}_{\bar{N}}\{\mathsf{Status}_N\} \\
6.\ N \to B: & \mathsf{Status}_A,\ \mathsf{Sig}_N\{\mathsf{Status}_A\} \\
 & \mathsf{Status}_N,\ \mathsf{Sig}_{\bar{N}}\{\mathsf{Status}_N\}
\end{array}} \tag{6}$$

Note that using ordinary signature schemes in this protocol would create a communication bottleneck on the top of the hierarchy. The top-level notary server would need as much communication as the leaf-servers altogether. Using hash-chain-signatures (5) in protocol (6) is crucial for preventing such a bottleneck. Indeed, no matter how many requests the notary N obtains from its clients during a round, only one signature is created and needs to be confirmed by the higher-level notary $\bar{N}$. Note also that the total service delay in a hierarchical on-line validation scheme is equal to the sum of delays on a path from the leaf-CA to the top-CA. This fact should be taken into account when developing client-friendly on-line validation services.

5 Reliability Issues

As we mentioned above, one of the main advantages of the traditional (off-line) PKI is that it does not give any extra powers to on-line service providers. In on-line validation protocols, however, we can notice two kinds of reliability concerns:

1) Private key of the Confirmer is used in a device connected to a public network and is therefore a potential target of attacks by network hackers.
2) On-line Confirmers are able to abuse their power to declare a certificate as valid, even if they know it is not. Though, Confirmers can be made accountable for their actions there may exist no effective ways to detect or prove their misbehavior.

On the one hand, we need fast on-line revocation/validation protocols in order to lower the risk of possible abuses of a stolen signature key. On the other hand, on-line services are less secure because of the abovementioned concerns of reliability.

If the value of a digitally signed document is relatively small, a confirmation signed by an on-line service provider may be considered as a sufficient proof. In this case, on-line validation service is a tool that helps to make fast decisions less riskier for the interested party. However, if the value of a signed document is high, for some cases only the CA itself may be reliable enough for confirming the validity of a certificate. But, if the CA updates the validity information once a day (which is a typical practice) the interested party would wait (in the worst case) the whole day before it becomes safe for him/her to accept the signature. Therefore, it seems we have a fundamental trade-off between *reliability* and *service time*.

In the following, we discuss several techniques how to increase the reliability of on-line validation services.

5.1 A Protocol with Short-Lived Certificates

Attacks where the on-line Confirmer (Notary) declares revoked certificates as valid are easily avoidable if the certificates issued by the CA have a short validity period (say one day). Using hash-tree signatures (5) it is relatively easy for the CA to re-issue all the certificates daily [8] and to send these certificates to the Notary. The Notary removes a certificate from its database (of valid certificates) once it has received a suitable revocation note. A verifier of the notary confirmation must check that the certificate confirmed was "fresh" at the moment of confirmation. The message flow in such a protocol would be as follows:

$$\begin{array}{|lrl|}\hline (\textit{daily}) & CA \to N\text{:} & \underbrace{(\mathsf{ID}_A,\ \mathsf{PK}_A,\ \mathsf{date}),\ \mathsf{Sig}_{CA}\{\mathsf{ID}_A,\ \mathsf{PK}_A,\ \mathsf{date}\}}_{\mathsf{Cert}_A(\mathsf{date})} \\ 1. & A \to B\text{:} & \mathsf{Sig}_A\{X\} \\ 2. & B \to N\text{:} & \mathsf{ID}_A,\ \mathsf{Sig}_A\{X\} \\ 3. & N \to B\text{:} & \underbrace{\mathsf{Valid}(\mathsf{Cert}_A(\mathsf{date})),\ \mathsf{Sig}_A\{X\}}_{\mathsf{Status}_A},\ \mathsf{Sig}_N\{\mathsf{Status}_A\} \\ \hline \end{array} \tag{7}$$

Suppose that each morning (say 8 am) the CA issues for each client A a new certificate $\mathsf{Cert}_A(\mathsf{date})$ which is valid only for a day (denoted as date). The certificate contains the identity ID_A and the public key PK_A of A. In order to obtain a confirmation for a message $\mathsf{Sig}_A\{X\}$ signed by A, the interested party B sends this signature together with the identity ID_A of A to the Confirmer N. If the certificate of A is in the database of valid certificates, the Confirmer signs a confirmation and sends it back to B. In this protocol, we must assume that the signature $\mathsf{Sig}_A\{X\}$ is also provided with a reliable time stamp t issued by a trusted Time-Stamping Authority (TSA). Each verifier of the signature must always check that the time stamp t belongs to date (i.e. was obtained when date was current). Provided that the public keys of the CA, N and the TSA were authentically obtained, the verifier should check that

– $\mathsf{Sig}_A\{X\}$ is properly verifiable using the public key written in $\mathsf{Cert}_A(\mathsf{date})$;
– $\mathsf{Cert}_A(\mathsf{date})$ is signed by the CA;
– Status_A confirms the validity of $\mathsf{Cert}_A(\mathsf{date})$ and comprises $\mathsf{Sig}_A\{X\}$;
– Status_A is signed by N;
– the time stamp confirms that $\mathsf{Sig}_A\{X\}$ was created at t, where $t \in \mathsf{date}$;
– the time stamp is signed by the TSA.

In such a scheme, the Notary is able to revoke certificates but is unable to declare a certificate as valid if actually the certificate was revoked a day before (or earlier). The most harmful attack the Notary is able to perform is being ignorant to the revocation notes sent by clients. However, using several Notary servers reduces the probability of even this attack.

As we mentioned above, a trusted Time-Stamping Authority is necessary for reliable validity proofs because incorrect time stamps may affect the results of validation. Time-stamping (as the Notary) is an on-line service and suffers thereby from the reliability concerns mentioned at the beginning of this section. Therefore, any advantage of protocol (7) over the previous protocols may seem questionable. In the next paragraph we present a better (though, more complex) solution which does not use trusted on-line parties.

5.2 A Protocol With Off-Line Time Stamps

In order to overcome the need for a trusted on-line Time-Stamping Authority we may use a protocol where the CA itself issues time stamps once a day. Each certificate $\mathsf{Cert}_{A,i}$ (issued by the CA for the i-th day) comprises a certain nonce value d_{-1} which is a digest $D(\Sigma_{i-1})$ of the set Σ_{i-1} of all the signatures submitted during the previous day. This digest is computed using the Merkle authentication tree. If A wants to sign a message X she adds her certificate to the message to be signed. If the i-th day is over, the Notary sends the CA the digest $d_i = D(\Sigma_i)$ of the set of all signatures submitted during the day. The CA then issues a time stamp $\mathsf{Sig}_{CA}\{\mathsf{date}_i, d_i\}$ and sends it back to the Notary. The protocol runs as follows:

$$\boxed{\begin{array}{lll}
(day_i) & CA \to N\colon & \underbrace{\underbrace{(\mathsf{ID}_A,\ \mathsf{PK}_A,\ \mathsf{date}_i,\ d_{i-1})}_{\mathsf{binding}_{A,i}},\ \mathsf{Sig}_{CA}\{\mathsf{binding}_{A,i}\}}_{\mathsf{Cert}_{A,i}} \\
1. & A \to B\colon & \sigma_A = \mathsf{Sig}_A\{X, \mathsf{Cert}_{A,i}\} \\
2. & B \to N\colon & \mathsf{ID}_A,\ \sigma_A \\
3. & N \to B\colon & \underbrace{\mathsf{Valid}(\mathsf{Cert}_{A,i}),\ \sigma_A}_{\mathsf{Status}_A},\ \mathsf{Sig}_N\{\mathsf{Status}_A\} \\
 & & N \text{ adds } \sigma_A \text{ into } \Sigma_i. \\
(day_{i+1}) & N \to CA\colon & d_i := D(\Sigma_i) \\
 & CA \to N\colon & \mathsf{Sig}_{CA}\{\mathsf{date}_i, d_i\}
\end{array}} \qquad (8)$$

In this protocol we do not need additional time-stamping services because the one-way links between the time-stamps and the signatures give an undeniable proof that the A's signature was in fact created at the i-th day. Indeed, we have a one-way relationship

$$d_{i-1} \longrightarrow \mathsf{Cert}_{A,i} \longrightarrow \mathsf{Sig}_A\{X, \mathsf{Cert}_{A,i}\} \longrightarrow d_i.$$

Moreover, if we do not want to take risk of accepting the signatures using on-line notary confirmations we can always wait till the next morning and obtain a one-way link

$$\mathsf{Sig}_A\{X\} \longrightarrow d_i \longrightarrow \mathsf{Cert}_{A,i+1}$$

which proves the validity of the signature entirely independent of any on-line validation services.

Thus, a verifier who has authentic copies of the public keys of the CA and N must check that

- $\sigma_A = \mathsf{Sig}_A\{X, \mathsf{Cert}_{A,i}\}$ is properly verifiable with PK_A written in $\mathsf{Cert}_{A,i}$;
- $\mathsf{Cert}_{A,i}$ is signed by the CA;
- Status_A confirms the validity of $\mathsf{Cert}_{A,i}$ and comprises σ_A;
- Status_A is signed by N;
- the equation $P(\sigma_A, \Sigma_i)(h(\sigma_A)) = d_i$ holds;
- $\mathsf{Sig}_{CA}\{\mathsf{date}_i, d_i\}$ is properly verifiable using the public key of the CA.

A verifier who does not trust N completely may also obtain the next certificate $\mathsf{Cert}_{A,i+1} = \mathsf{Sig}_{CA}(\mathsf{ID}_A, \mathsf{PK}_A, \mathsf{date}_{i+1}, d_i)$ and check that

- $\mathsf{Cert}_{A,i+1}$ comprises d_i and PK_A,

which confirms the validity of the signature independent of N because if A's certificate is revoked during date_i the CA does not issue $\mathsf{Cert}_{A,i+1}$.

6 Conclusions

We presented a long-term digital signature validation scheme that does not suffer from the main disadvantages often associated with on-line techniques. Our scheme supports fast revocation while giving no extra power to on-line validation services. Due to the efficient hash-chain-signatures our protocol is efficient and scalable. One advantage over the previous schemes is that our scheme does not require additional trusted parties.

References

1. Adams, Sylvester, Zolotarev, and Zuccherato. Data Validation and Certification Server Protocols. Technical report, PKIX Working Group, October 1999.
2. Carlisle Adams and Robert Zuccherato. Time stamp protocols. Technical report, PKIX Working Group, 1999.
3. Dave Bayer, Stuart Haber, and W. Scott Stornetta. Improving the efficiency and reliability of digital time-stamping. In *Methods in Communication, Security, and Computer Science – Sequences'91*, pages 329–334, 1992.

4. Josh Benaloh and Michael de Mare. Efficient broadcast time-stamping. Technical Report 1, Clarkson University Department of Mathematics and Computer Science, August 1991.
5. Ahto Buldas, Peeter Laud, Helger Lipmaa, and Jan Villemson. Time-stamping with binary linking schemes. In *Advances in Cryptology – CRYPTO'98*, volume 1462 of *LNCS*, pages 486–501, Santa Barbara, 1998. Springer-Verlag.
6. Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, 1976.
7. Barbara Fox and Brian LaMacchia. Online certificate status checking in financial transactions: the case for re-issuance. In *Financial Cryptography – FC'99*, volume 1648 of *LNCS*, pages 104–117, Anguilla, February 1999.
8. Irene Gassko, Peter S. Gemmell, and Philip MacKenzie. Efficient and fresh certification. In *Public Key Cryptography – PKC'2000*, volume 1751 of *LNCS*, pages 342–353, Melbourne, Australia, January 2000. Springer-Verlag.
9. Stuart Haber and W.Scott Stornetta. How to time-stamp a digital document. *Journal of Cryptology*, 3(2):99–111, 1991.
10. Paul C. Kocher. On certificate revocation and validation. In *Financial Cryptography: FC'98*, volume 1465 of *LNCS*, pages 172–177, Anguilla, February 1998. Springer-Verlag.
11. Loren M. Kohnfelder. Toward a practical public-key cryptosystem. 1978.
12. Ralph C. Merkle. Protocols for public key cryptosystems. In *Proceedings of the 1980 IEEE Symposium on Security and Privacy*, pages 122–134, 1980.
13. Ralph C. Merkle. A certified digital signature. In *Advances in Cryptology – CRYPTO'89*, volume 435 of *LNCS*, pages 218–238, Santa Barbara, 1989. Springer-Verlag.
14. Michael Myers, R. Ankney, A. Malpani, S. Galperin, and Carlisle Adams. RFC2560: X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. June 1999.
15. Ronald Rivest. Can we eliminate certificate revocation lists? In *Financial Cryptography: FC'98*, volume 1465 of *LNCS*, pages 178–183, Anguilla, February 1998. Springer-Verlag.
16. Meelis Roos. Integrating time-stamping and notarization. MSc Thesis, Tartu University, `http://home.cyber.ee/mroos/thesis/`. May 1999.