High-Performance Qualified Digital Signatures for X-Road

Arne Ansper[1,2], Ahto Buldas[1,2], Margus Freudenthal[1], Jan Willemson[1]

[1] Cybernetica AS, Mäealuse 2/1, Tallinn, Estonia
[2] ELIKO Competence Centre in Electronics-, Info- and Communication Technologies, Mäealuse 2/1, Tallinn, Estonia*

Abstract. In Estonia, the X-Road infrastructure for unified governmental database access has been in use for more than 10 years now. The number of queries mediated over the X-Road has exceeded 240 million per year. Even though all the queries and replies are signed by using the X-Road's own PKI facilities, the resulting signatures are not fully qualified in the sense of the Digital Signatures Act. There are several technical issues to be solved, most notably performance requirements, since the operations needed to achieve qualified signatures (obtaining OCSP responses and time stamps) require time. The topic of this paper is to propose organisational and technical solutions to overcome these challenges. A novel batch signature and time stamp format is proposed allowing to perform many PKI operations by a price of one, helping to meet the performance requirements.

1 Introduction

Recent developments in ICT have made digital communications an integral part of our everyday life. More and more services are provided and consumed over the Internet, and this holds true both for private and governmental sector services.

In Estonia, there are two foundational electronic frameworks that have been running for more than 10 years now and that act as enablers for a large number of other services.

First of them, the national ID-card infrastructure, was launched in 2002. At the time of this writing (April 2013), there are 1,193,050 active ID-cards[3]. Currently there are about 80 different public and private service providers who make use of ID-card authentication and signature mechanisms, including all the commercial banks and telecom companies, Estonian Tax and Customs Board, Center of Registers and Information Systems, etc.[4]

* This research has been supported by European Union through European Regional Development Fund under ELIKO Competence Center (EU30017) and EXCS Center of Excellence in Computer Science.

[3] The whole population population of Estonia is roughly 1.3 million according to the census of 2011. The number of active ID-cards can be seen at `http://www.id.ee/`, last accessed April 25th, 2013.

[4] `http://id.ee/index.php?id=31007`, last accessed April 25th, 2013.

The second important Estonian digital infrastructure facility is called the X-Road, and it was launched already in 2001. It acts as a unified access layer to most of the governmental registers, allowing secure and efficient data access by both the relevant authorities and citizens. Currently, more than 100 organizations are providing services over the X-Road and more than 600 registers are accessible via the infrastructure.

One of the main issues with the current X-Road implementation is that even though the SOAP messages internally used by the X-Road are signed, these signatures can not be considered qualified with respect to the Estonian Digital Signatures Act. The X-Road uses its own Certification Authority and time-stamping service, but relies on software-based signature key management and a non-standard mechanism for distributing certificate validity information. As a result, the legal status of X-Road messages is potentially unclear.

This paper describes the efforts made to overcome these problems. The main issue to tackle when turning X-Road message signatures into fully qualified ones is performance. For example, the existing ID-card infrastructure, even though providing all the required mechanisms (hardware tokens for key management, OCSP responder for certificate validation, etc.), is not built to handle the volumes currently required by the X-Road. The number of digital signatures given by all the ID-cards over 11 years is roughly 112 million[5], but the number of X-Road requests made *per year* exceeded 240 million in 2011 [Kal12], and both the requests and their responses are signed.

This paper presents both organizational and technical measures required to meet these performance goals. The main technical contribution of the paper is the batch data format that can be used to give many signatures and time stamps in one operation.

The paper is organised as follows. First, Section 2 provides the necessary background about the X-Road, and covers the requirements for the new technical solution. It also provides a discussion of organizational measures needed. Section 3 presents various potential technical solutions and selects the one most suitable for the X-Road. In Section 4 we discuss its performance and implementation details. Finally, Section 5 draws some conclusions and sets directions for further work.

2 X-Road

By early 2000s the level of computerization in the Estonian state databases had reached both the level of sufficient technical maturity and a certain critical volume so that the need for a unified secure access mechanism was clear. To address that need, the development activities on the modernization of national databases started in the beginning of 2001 [KV02,Kal02]. The first version of the developed X-Road infrastructure was launched on December 17th 2001. Today, already the fifth generation of the X-Road is in operation [Kal12].

[5] `http://www.id.ee/`, last accessed April 25th, 2013.

Since the first version of the X-Road, several requirements were governing its development [ABFW03,Kal12]:

- *Integrity and authenticity.* Since information received over the X-Road will potentially be used to take legally binding actions, information integrity and authenticity are the primary security requirements.
- *Confidentiality and authentication.* Most (though not all) the data transferred over the X-Road is meant to be processed only by certain authorities. Thus, strict access control mechanisms had to be built into the system.
- *High availability and scalability.* Operating as an overlay network on the Internet, the X-Road is a subject to availability threats. Hence it had to be designed to work even if some central services were unavailable. As a result, the X-Road is very distributed with only a very limited number of central services (like certification, directory and logging). This architecture has also allowed the infrastructure to scale extremely well. As noted above, in 2011, the number of queries made over the X-Road per year reached 240 million.

To ensure integrity and access control, the X-Road is provided with its own public key infrastructure (PKI) solution, where all the service providers have public key certificates, all the queries and replies are digitally signed and time-stamped [WA08].

By the time the first version of the X-Road was launched, the Estonian national PKI was still in its infancy (the first ID-cards were issued only in 2002, i.e. the next year after the X-Road was deployed). Hence, the X-Road PKI and the national PKI were developed independently, and have remained so for 10 years.

This has caused the problem of signatures on the X-Road queries and responses having unclear legal status in the light of the Estonian Digital Signatures Act. Although no X-Road signature has ever been disputed in court, it does not make sense to use an underlying technology that leaves the room for such disputes.

The first issue to resolve when trying to turn X-Road signatures into legally valid ones is the matter of key management. Currently, the service providers keep the signature keys in the software of X-Road security servers. Security server is a component that encapsulates security-related functionality of an organisation connected to the X-Road infrastructure. It is responsible for signing the outgoing messages, verifying signatures on incoming messages, maintaining a secure transaction log, and enforcing access control. Security server acts as a gateway between the organisation connected to the X-Road and the X-Road infrastructure.

Even though signing by a security server does not directly contradict the current wording of the Digital Signatures Act, there are several indications that in the near future, it will not be possible to comply with the legal requirements without keeping the signature keys in a physically protected hardware environment.

For example, Annex III of the Directive 1999/93/EC of the European Parliament and of the Council on a Community framework for electronic signatures

states the requirements put onto secure signature-creation devices. Namely, the directive states that

> *Secure signature-creation devices must, by appropriate technical and procedural means, ensure at the least that:*
> *(a) the signature-creation-data used for signature generation can practically occur only once, and that their secrecy is reasonably assured;*
> *(b) the signature-creation-data used for signature generation cannot, with reasonable assurance, be derived and the signature is protected against forgery using currently available technology;*
> *(c) the signature-creation-data used for signature generation can be reliably protected by the legitimate signatory against the use of others.*

Even though to the best of the authors' knowledge no legal act explicitly states that these requirements can only be met by hardware-protected signature keys, we argue that given the current development and sophistication of malware attacks, software-only protection mechanisms are no more adequate to meet the above-cited requirements. Hence, a design decision was taken to provide the next generation of the X-Road with the hardware-enabled signature creation devices.

Currently, there are roughly two kinds of cryptographic hardware devices available on the market.

- Hardware Security Modules (HSM) provide high throughput for cryptographic operations like signing, but are also rather expensive. Some of the organizations who have joined the X-Road infrastructure already make use of HSMs, but it is clear that HSMs are not cost-efficient for most of the smaller service providers.
- Chip card or USB token based devices are considerably cheaper, but their performance is considerably slower as well. A typical device of this kind is capable of giving only a few signatures per second, which is clearly insufficient given the current volumes of the X-Road traffic (please refer to Section 4 for some more detailed performance estimates).

Hence one of the key problems to address is enabling the low-end cryptographic hardware to improve the throughput by several orders of magnitude. This can be done by *batching* several signature requests into one data structure and taking just one signature per batch. The details of this solution will be presented in Section 3.

There are also other aspects to consider when making the X-Road signatures compatible with the ones acceptable from the viewpoint of the Digital Signatures Act and the Estonian national PKI.

First, all the digital signatures require a time-stamp. Given that the number of X-Road queries per day can reach up to one million [Kal12], the time-stamping service of the Estonian national PKI would not be able to handle this. Similarly, the OCSP-based certificate validity confirmation service would not be able to

provide sufficient throughput. Second, both the time-stamping and OCSP services may be subjects to availability problems, and their temporary unavailability should not block the core operations of the X-Road infrastructure. We will discuss these issues in more detail in Section 2.1.

Finally, the signature format currently used by X-Road differs from the one used in the national PKI. This makes X-Road signatures impossible to verify with the software available for the standard ID-card operations. As a part of the projects of updating both the X-Road infrastructure and the Estonian national PKI, it was decided to improve their interoperability by moving to a unified signature format.

This process was first started by the developers of the national PKI. After studying different standards suggested by European Commission [EC211] it was decided to base the next Estonian digital signature standard on XAdES format and ASiC container [XAd10,ASi12].

Given the current state of standards, these choices are indeed reasonable. However, they do not provide a solution for the high availability requirements listed above. Presenting such a solution will be the main contribution of this paper.

2.1 X-Road Message Signature Validation Workflow

X-Road communication operates by exchanging queries and responses, both of which are signed messages. Signatures are used to provide both authenticity and integrity, and hence their validity is essential in order to meet the security requirements put onto the whole infrastructure.

Two main mechanisms are used in conjunction to ensure the validity of the public key certificate at the moment of signing. First, Online Certificate Status Protocol (OCSP) is used to make a statement concerning the validity interval of the certificate, and second, a time-stamp is used to prove that the signature existed at some moment in time. If this moment falls into the validity interval granted by the OCSP statement, the certificate and the corresponding signature are considered to be valid.

In its current setting, the Estonian national PKI solution makes one OCSP query per signature. By using the hash of the signature as OCSP nonce, it is possible to obtain behaviour similar to time-stamping (in [ABRW01] it is called notarization). However, in its vanilla form it has several performance issues.

1. The OCSP responder is a single point of failure. Even short-term unavailability of the service would block the operations of the entire X-Road if every message would need a dedicated OCSP response.
2. The OCSP server will be overloaded. If an OCSP confirmation would be required for all the X-Road messages, the OCSP server would not be able to handle this volume.
3. The requirement to make one OCSP request per X-Road message also introduces latency. When the latencies start accumulating in case of complex X-Road queries, this poses a serious availability threat.

All these issues can be efficiently solved by caching the OCSP responses at the message sender's side. This will guarantee the availability of signing functionality even in case of OCSP responder's temporary malfunctioning. It will still be possible to give valid signatures as long as the cached OCSP response remains fresh. The length of the freshness interval depends on the policy, but it could be, say, a couple of hours. When a temporarily unavailable OCSP responder becomes functional again during this period, service continuity is not harmed.

Regarding the possible overload of the OCSP server, pre-caching the responses allows to use one response for several signatures, hence the number of the responses does not depend on the number of signatures any more, but rather on the number of currently valid certificates, the number of which is significantly lower.

And third, a pre-cached copy of OCSP response allows for immediate message signing, removing the potential threat of latency.

The problems with the time-stamping service are similar. If the signer would need to wait for the time-stamping server's reply, this would again create a single point of failure together with the latency issues and the potential overload if every message would be required to have a separate time stamp.

Solution to this problem starts from the observation that it is actually the message receiver who is interested in the validity of the signature, since he is the one who may need to prove that the actions he took were a consequence of a valid message. Hence we let the message receiver to take care of obtaining the time stamp. The above-mentioned three problems still stand, but are easier to solve in the message receiver's end.

Since the OCSP response has a validity interval, taking a time-stamp is not a time-critical action. In fact, the decision whether to request the time stamp instantly or to queue the request for later, is a policy issue. As such, it will constitute a trade-off between service availability and the risk of potential inability to prove the signature validity at a later time in case the time-stamping server will become unavailable.

Request queueing also allows us to solve the problem of time-stamping server's overload. This way it will be possible to aggregate the time-stamping requests into one batch data structure and letting the server to time-stamp this instead of time-stamping all the requests separately.

It turns out that for this purpose, the same data structure can be used as for batching several signature requests. We will now discuss this solution in detail in Section 3.

3 Batch Signatures and Timestamps

Signature schemes such as RSA are time-consuming compared to other cryptographic operations like symmetric encryption or hashing. For servers that have to generate hundreds or thousands of signatures per second it would be desirable to have methods that enable more efficient signing than just signing every message separately.

By a batch signature we mean an algorithm S that, having as input a list $M_1, \ldots, M_\ell$ of messages, creates signatures $s_1, \ldots, s_\ell$, so that there is an efficient verification algorithm V, so that $V(s_i, M_i) = 1$ whenever $(s_1, \ldots, s_\ell) \leftarrow S(M_1, \ldots, M_\ell)$. Batch signature schemes make sense if S uses less computational resources than signing all messages separately with ordinary signature schemes. There are several methods known for batch signatures.

3.1 Fiat's Batch RSA

The first batch signature scheme—batch RSA—was proposed by Amos Fiat in 1989 [Fia89,Fia97]. It enables to effectively perform several modular exponentiations at the cost of a single modular exponentiation, which is very useful if many RSA signatures (or pure-RSA encryptions instead of hybrid encryptions) must be performed at some central site. A batch RSA signature looks like an ordinary RSA signature except that the choice of public exponents is different—instead of one fixed public exponent, a batch signature scheme uses many public exponents $e_1, \ldots, e_\ell$ that are relatively prime to $\varphi(N)$ and to each other. For signing a batch $(M_1, \ldots, M_\ell)$ of messages, the batch RSA computes

$$M_1^{\frac{1}{e_1}} \bmod N,\ M_2^{\frac{1}{e_2}} \bmod N, \ldots, M_\ell^{\frac{1}{e_\ell}} \bmod N\ ,$$

at a time using just one full-size modular exponentiation, where N is the RSA modulus. There are several restrictions when using such a scheme for encryption. For example, one must take care that the same message M is not encrypted with two different relatively prime public exponents e_i and e_j because M can easily be computed from the two cryptograms $M^{e_i} \bmod N$ and $M^{e_j} \bmod N$.

One more obstacle of using such a scheme in practice is that the public-key certificates we have to use either have to contain many public exponents instead of one, which is not supported by standards, or there have to be many different public-key certificates issued with different public exponents that all correspond to the same secret key, which again is not foreseen by standard PKI solutions. More universal solutions for batch signatures are thereby necessary.

3.2 Simple Batching with Hash Lists

There is a very simple method for converting any conventional signature scheme to a batch signature scheme which is not that size economic as the Fiat's batch RSA, but is simple and universal. The so-called *hash list* scheme works as follows. In order to create a batch signature for a batch $M_1, \ldots, M_\ell$, the signer

1. Hashes all the messages: $m_i = h(M_i)$ for all $i = 1 \ldots \ell$.
2. Creates the *hash list* $L = (m_1, m_2, \ldots, m_\ell)$ and computes the hash value $m = h(L)$ of the list.
3. Signs the hash value m by using an ordinary signature scheme: $s = \sigma(m)$.
4. The signature s_i for any M_i is a pair $s_i = (s, L)$ that consists of the ordinary signature and the hash list L.

In order to verify (s, L) as the signature of M_i, the verifier:

1. Computes $m_i = h(M_i)$;
2. Checks if $m_i \in L$;
3. Computes $m = h(L)$; and
4. Verifies the ordinary signature $s = \sigma(m)$ on m.

This scheme is very easy to implement and is feasible if the batch size is relatively small. The size of the signature is $\ell \cdot \mid h \mid + \mid s \mid$, where $\mid s \mid$ is the size of the conventional signature and $\mid h \mid$ is the number of output bits of h.

A batch signature scheme based on hash lists was recently implemented in Azerbaijan in a system very similar to X-Road [Kal12]. However, due to its performance limitations, a more sophisticated solution was required for Estonia, and we will present this solution in the next Section.

3.3 Signatures with Batch Residue

In 1999, Pavlovski and Boyd [PB99] proposed a general technique for converting any public-key signature scheme (with appendix) efficiently to a batch signature scheme by using Merkle hash trees [Mer80]. Their batch signatures $s_1, \ldots, s_\ell$ for a batch $M_1, \ldots, M_\ell$ consist of an ordinary signature s that depends on all M_i, and a *batch residue* r_i which varies with every message, i.e. $s_i = (s, r_i)$. Signature verification consists of recalculating the input to the ordinary signature s using the message M_i and batch residue r_i, and then verifying the ordinary signature s.

The calculation of the batch residue and its verification only use hash computations. Therefore, creating a batch signature is almost as efficient as generation of a single ordinary signature. Batch signatures are somewhat longer than ordinary signatures because they contain batch residues of size $\mid h \mid \cdot \log \ell$, where $\mid h \mid$ is the number of output bits of the hash function h.

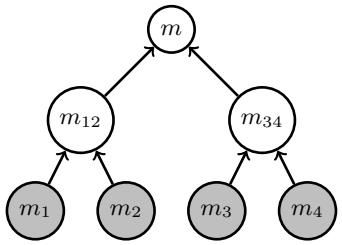

Fig. 1. A Merkle hash tree for $m_1, \ldots, m_4$.

For example, in case of $\ell = 4$, the batch signature of [PB99] for a batch M_1, M_2, M_3, M_4 is created (by using a hash function h and an ordinary signature scheme σ) via the following steps:

1. All the messages are hashed: $m_i = h(M_i)$ for all $i = 1 \dots 4$.
2. The Merkle hash tree (Fig. 1) is computed: $m_{12} = h(m_1, m_2)$, $m_{34} = h(m_3, m_4)$, $m = h(m_{12}, m_{34})$.
3. The root hash value m is signed by using the ordinary signature scheme: $s = \sigma(m)$.
4. The batch residues are composed as follows: $r_1 = \{m_2, m_{34}\}$, $r_2 = \{m_1, m_{34}\}$, $r_3 = \{m_4, m_{12}\}$, $r_4 = \{m_3, m_{12}\}$.

In order to verify $s_3 = (s, r_3) = (s, \{m_4, m_{12}\})$ as the signature of M_3, the verifier:

1. Computes $m_3 = h(M_3)$;
2. Computes $m = h(m_{12}, h(m_3, m_4))$; and
3. Verifies the ordinary signature $s = \sigma(m)$ on m.

All specific Fiat type batch signatures schemes have some restrictions, in terms of batch size limitations, being for verification only, having no support for heterogeneous signature generation for different recipients, etc. The scheme of [PB99] has obvious advantages over the Fiat-type schemes, including an improved signature size, the ability to batch-sign for independent recipients, and unlimited batch size. Hence, we propose using it to meet the performance requirements set by the X-Road.

Next, Section 4 discusses the gains of this approach. In the end of the paper the reader will also find Appendix A presenting the data structure for batch signatures and time stamps in XML XSD format, and Appendix B with the corresponding example XML data structures.

4 Discussion

Concluding from Section 2, we had two major improvement targets for the X-Road.

The first target was achieving a clear legal status of the X-Road message signatures as qualified ones in terms of the Estonian Digital Signatures Act. This is achieved by satisfying the main requirement unsatisfied this far and enabling hardware-based signature key management. Section 2.1 also discussed the organizational mechanisms needed to support standard signature validation mechanisms (OCSP responses and time stamps). Hence we can conclude that the first target has been met (at least as far as the technical aspects are concerned).

The second target was dealing with the performance issues caused by the heavy and increasing volume of X-Road traffic. As noted in Section 2, affordable chip card or USB token based solutions do not provide sufficient throughput. For example, SafeNet's Smart Card 400 chip card is able to produce one RSA1024 signature in 0.45 and one RSA2048 signature in 1.23 seconds[6]. At peak loads, this is performance is insufficient[7].

[6] `http://www.safenet-inc.com/products/data-protection/two-factor-authentication/smart-card/`, last accessed April 29th, 2013.

[7] According to the statistics obtained from the X-road manager Heiko Vainsalu, the current documented peak occurred on April 5th, 2013, when the Digital Prescription

By using the Merkle tree, we will be able to aggregate a large number of messages into one batch. It is clear that in order to produce a Merkle tree on ℓ leaves, $\ell - 1$ hash evaluations need to be computed. We will use the ECRYPT II project benchmarks on SHA-3 candidates to evaluate the time needed for hash computations[8]. According to this source, Keccak-256 (a member of the contest-winning SHA-3 family) uses 101.3 cycles per byte for a 64-byte message on a 2400MHz AMD Athlon processor. This corresponds exactly to our scenario where the Merkle tree is built stepwise by hashing two values into one digest. By selecting 64-byte input and 256-bit output, we obtain exactly the required 2-to-1 compression rate at a high security level. In one second, this setup allows us to perform

$$\frac{2400 \cdot 10^6}{101.3 \cdot 64} \approx 370000$$

hash operations on this relatively modest hardware.

To estimate the actual throughput of the signature creation device, we also need to take the time required to compute the digests m_i from the documents M_i into account. As the messages M_i may be of different size, we have to measure their hashing time in terms of elementary 64-byte hashing operations. For that, we imagine that the messages M_i consist of m_i blocks of 32-byte in length and every elementary hashing operations decreases the number of such blocks by one, until one single hash value remains which is the output of the hash function. Exactly $m_i - 1$ such operations are needed for that. Note that this is not the way the hash is actually computed, but gives an upper bound for the hash computation time—hashing a long message is more efficient in practice than hashing a number of short messages.

Let ℓ be the batch size and let the smart-card used for batch signatures be capable of creating one signature per second. If the messages $M_1, \ldots, M_\ell$ have lengths $32 \cdot m_1, \ldots, 32 \cdot m_\ell$ bytes, respectively, then hashing them takes $(m_1 - 1) + \ldots + (m_\ell - 1) = m_1 + \ldots + m_\ell - \ell$ hash operations (with 64-byte input). In addition, the Merkle tree requires $\ell - 1$ such hash operations. Hence, the total number of hash computations is $m_1 + \ldots + m_\ell - 1$. If we create one batch per second, then

$$m_1 + \ldots + m_\ell - 1 \approx 370000 \ , \tag{1}$$

whereas the total throughput of the batch signing device is $32(m_1 + \ldots + m_\ell)$ bytes per second, i.e. $m_1 + \ldots + m_\ell$ blocks (of 32-bytes) per second. Hence, from (1) the througput f of the signing device is

$$f = 32(m_1 + \ldots + m_\ell) \approx 32 \cdot 370000 = 11.84\,\text{MB/s} \ ,$$

which is sufficient considering that the communication lines between the X-Road security servers would not allow much more traffic anyway.

Center had to process 4194 messages during a 5-minute period (11:45-11:50). This amounts to roughly 14 messages per second on average. However, X-Road load is expected to grow continuously, and this record is likely to be beaten in the near future.

[8] `http://bench.cr.yp.to/results-sha3.html`, last accessed on April 29th, 2013.

5 Conclusions and further work

This paper discussed some of the problems encountered during the first 11 years of deployment of the X-Road infrastructure together with possible solutions. Two central issues we covered were the legal status of the X-Road messages and rapidly growing performance requirements.

In order to make the signatures on the X-Road messages compliant with the requirements of the Estonian Digital Signatures Act, two main aspects need to be improved. First, signature key must be managed in a hardware-protected environment (chipcard or HSM), and second, standard signature validation methods (OCSP and time stamps) must be used. These improvements have their inherent technical limitations, as the currently available and affordable solutions are not built to handle the current X-Road communication volumes.

In order to meet these requirements, this paper proposed a set of solutions. First, to reduce the workload of OCSP and time-stamping servers, OCSP precaching and time-stamp batching were discussed. Second, to allow low-end hardware (chipcards) to produce more signatures per second, signature batching was proposed.

It turns out that time stamp and signature batches can be built on top of the same Merkle tree structure. We presented a suitable structure as and XML schema and evaluated the performance gains it provides. We concluded that using even a moderate hardware, we can achieve signature device throughput up to 11.84 MB/s. Clearly, this number can be increased even further by deploying more advanced hardware. On the other hand it is also clear that for a full implementation, other operations besides hashing also require time. Hence, the actual performance benchmarking must still be performed after the implementation of the proposed scheme.

Implementation of this scheme is already in the works at the time of this writing. Real benchmarking will remain part of the future work once the implementation will be finished.

Another interesting future challenge will be deployment of this solution in an environment other than Estonia, where the legal framework, existing PKI, etc. might differ considerably. A system based on the Estonian X-Road experience was recently launched in Azerbaijan [Kal12], but other similar deployments will depend more on political, rather than the technical issues.

References

[ABFW03] Arne Ansper, Ahto Buldas, Margus Freudenthal, and Jan Willemson. Scalable and Efficient PKI for Inter-Organizational Communication. In *Proceedings of the 19th Annual Computer Security Applications Conference ACSAC 2003*, pages 308–318, 2003.

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

[ASi12] Electronic Signatures and Infrastructures (ESI); Associated Signature Containers (ASiC), February 2012. ETSI TS 102 918.

[EC211] European Commission Decision of 25 February 2011 establishing minimum requirements for the cross-border processing of documents signed electronically by competent authorities under Directive 2006/123/EC of the European Parliament and of the Council on services in the internal market. 2011/130/EU, February 2011.

[Fia89] Amos Fiat. Batch RSA. In *Proceedings on Advances in cryptology*, CRYPTO '89, pages 175–185, New York, NY, USA, 1989. Springer-Verlag New York, Inc.

[Fia97] Amos Fiat. Batch RSA. *J. Cryptology*, 10(2):75–88, 1997.

[Kal02] Ahto Kalja. The X-Road Project. A Project to Modernize Estonia's National Databases. *Baltic IT&T review*, 24:47–48, 2002.

[Kal12] Ahto Kalja. The first ten years of X-road. In *Estonian Information Society Yearbook 2011/2012*, pages 78–80. Department of State Information System, Estonia, 2012.

[KV02] Ahto Kalja and Uuno Vallner. Public e-Service Projects in Estonia. In Hele-Mai Haav and Ahto Kalja, editors, *Databases and Information Sustems, Proceedings of the Fifth International Baltic Conference, Baltic DB&IS 2002*, volume 2, pages 143–153, June 2002.

[Mer80] Ralph C. Merkle. Protocols for public key cryptosystems. In *Proc. of the 1980 IEEE Symposium on Security and Privacy*, pages 122–134, 1980.

[PB99] Christopher J. Pavlovski and Colin Boyd. Efficient batch signature generation using tree structures. In *International Workshop on Cryptographic Techniques and E-Commerce: CrypTEC'99*, pages 70–77. City University of Hong Kong Press, 1999.

[WA08] Jan Willemson and Arne Ansper. A Secure and Scalable Infrastructure for Inter-Organizational Data Exchange and eGovernment Applications. In *Proceedings of The Third International Conference on Availability, Reliability and Security ARES 2008*, pages 572–577. IEEE Computer Society, 2008.

[XAd10] Electronic Signatures and Infrastructures (ESI); XML Advanced Electronic Signatures (XAdES), December 2010. ETSI TS 101 903.

A XML XSD for Batch Signatures and Timestamps

In this Section, we present the XML XSD that supports presenting the Merkle hash tree structure for batch signatures and time stamps. The data structure does not actually contain the whole tree (it may be too large to handle), but just the minimal part of it to be able to prove that the given leaf item participated in forming the root value.

For the example presented in Figure 1, in order to prove that the root value m depends on the leaf m_3, we need to add the the batch residue values m_4 and m_{12}, as explained in Section 3. The resulting data structure (m_3, m_4, m_{12}) is called a *hash chain*, and this is essentially what we will formally describe next.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.example.org/hashchain"
    xmlns:tns="http://www.example.org/hashchain"
    elementFormDefault="qualified
        xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <import schemaLocation="xmldsig-core-schema.xsd"
        namespace="http://www.w3.org/2000/09/xmldsig#"></import>
```

The schema defines a new namespace `http://www.example.org/hashchain` used to describe the hash tree structure. As a base schema, XMLDsig is used, as indicated by the `ds` namespace.

```
    <complexType name="HashChainType">
        <sequence>
            <element name="DefaultTransforms"
                type="ds:TransformsType" minOccurs="0">
            </element>
            <element name="DefaultDigestMethod"
                type="ds:DigestMethodType" minOccurs="0">
            </element>
            <element name="HashStep" type="tns:HashStepType"
                minOccurs="0" maxOccurs="unbounded">
            </element>
        </sequence>
    </complexType>
```

The main data structure for representing hash computations is hash chain (of type `HashChainType`), consisting of a series of hash steps (of type `HashStepType`). Before the actual hash function application, some standard canonization transformations (of type `TransformsType`) may be used. It is also possible to specify the default hash algorithm (of type `DigestMethodType`) to the whole tree.

```
    <complexType name="HashStepType">
        <sequence>
            <choice maxOccurs="unbounded" minOccurs="0">
                <element name="HashValue" type="tns:HashValueType"/>
                <element name="RefValue" type="tns:RefValueType"/>
            </choice>
        </sequence>
```

```
        <attribute name="id" type="ID" use="optional"/>
    </complexType>
```

One hash step (of type `HashStepType`) can be used to hash together a series of values (represented either by elements `HashValue` or `RefValue`). The values have the base type `AbstractValueType` that defines the common elements and attributes.

```
<complexType name="AbstractValueType">
    <sequence>
        <element name="Transforms"
            type="ds:TransformsType" minOccurs="0"></element>
        <element name="DigestMethod"
            type="ds:DigestMethodType" minOccurs="0"></element>
    </sequence>
</complexType>
```

If necessary, it is possible to define non-default transformations and hash algorithms for some particular hash steps

```
<complexType name="RefValueType">
    <complexContent>
        <extension base="tns:AbstractValueType">
            <attribute name="URI" type="anyURI"></attribute>
        </extension>
    </complexContent>
</complexType>
```

One of the possible value types to be used as the input for a hash step is `RefValueType`. Its value is an URI referring to the leaf data item of the Merkle tree (m_3 in the example above), or to another branch of the tree.

```
<complexType name="HashValueType">
    <complexContent>
        <extension base="tns:AbstractValueType">
            <sequence>
                <element name="DigestValue"
                    type="ds:DigestValueType">
```

```
                    </element>
                </sequence>
            </extension>
        </complexContent>
    </complexType>
```

The other possible value type to be used as the input for a hash step is `HashValueType`. Its values may be the other hash values used in the hash chain computation (m_{12} in the example above) or other leaf data items in the Merkle tree (m_4 in the example above).

```
    <element name="HashChain" type="tns:HashChainType"></element>

    <element name="HashChainResult" type="ds:ReferenceType"></element>
</schema>
```

Finally, elements of the defined types are declared. The `HashChainResult` element contains the root hash value (m in the example above) and the `HashChain` element contains the hash chain itself ((m_3, m_4, m_{12}) in the example above).

B Examples of the Data Structures

First we will give an example hash chain for the Merkle tree in Figure 1. The hash chain contains two hash steps. The first step, `m`, represents the computation $m = h(m_{12}, m_{34})$. Its first hash value is $m_{12} = h(m_1, m_2)$, representing the left subtree. The second hash value in turn refers to the step `m34`, i.e. the right subtree. This step represents the computation $m_{34} = h(m_3, m_4)$. The first value in this step is a reference to the concrete message M_3 (the file `m3datafile.dat`), whereas the second value is the digest $m_4 = h(M_4)$.

```
<HashChain xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    xmlns="http://www.example.org/hashchain">
    <DefaultTransforms>
        <ds:Transform
            Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
    </DefaultTransforms>
    <DefaultDigestMethod
        Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
    <HashStep id="m">
        <HashValue> <!-- Digest m12=h(m1, m2) -->
```

```
            <DigestValue>8dLS+STphqy...</DigestValue>
        </HashValue>
        <RefValue URI="#m34"/> <!-- Reference to digest m34 -->
    </HashStep>
    <HashStep id="m34">
        <!-- Reference to data file containing M3 -->
        <RefValue URI="/m3datafile.dat"/>
        <HashValue> <!-- Digest m4=h(M4)  -->
            <DigestValue>4kLtO//M3yc...</DigestValue>
        </HashValue>
    </HashStep>
</HashChain>
```

The second example shows the file representing the result of the Merkle tree computation. It contains the result of the hash step `m` and also a reference to the corresponding XML element. This file can be signed and verified by standard digital signature software. Additional software is needed to verify that the signed Merkle tree top refers to correct hash chain that proves validity of the message in the file `m3datafile.dat`.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- The URI attribute refers to step m in the hash chain -->
<hc:HashChainResult xmlns:hc="http://www.example.org/hashchain"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    URI="/hashchain3.xml#m">
    <ds:Transforms>
        <ds:Transform
            Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
    </ds:Transforms>
    <ds:DigestMethod
        Algorithm="http://www.w3.org/2001/04/xmlenc#sha256">
    </ds:DigestMethod>
    <ds:DigestValue>qiTak6MdcsN...</ds:DigestValue>
</hc:HashChainResult>
```