Integrating XML Linked Time-stamps in OASIS Digital Signature Services

Ana Isabel González-Tablas[1] and Karel Wouters[2]

[1] Carlos III University (Madrid),
Computer Science Department - SeTI
aigonzal@inf.uc3m.es

[2] Katholieke Universiteit Leuven,
Department Electrical Engineering - ESAT, COSIC
karel.wouters@esat.kuleuven.be

Abstract. The technique of electronic time-stamping allows a client to get an electronic proof of the existence of a document at a specific point in time. A simple way to achieve this is to produce a digital signature over the pair (document,time). Linked time-stamps have an advantage over these simple time-stamps because they construct a verifiable link between time-stamps. In this paper, we discuss how to include linked time-stamps in the OASIS Digital Signature Services standard. We highlight the problem points when introducing a sub-profile of this standard, and we describe some additional structures that are needed to accommodate a broad range of linked time-stamping schemes.

Keywords. Linked time-stamping, Digital Signature Services, XML security.

1 Introduction

In recent years, time-stamping implementations have become popular in several countries. Studies, performed by the European Committee for Standardization [10], clearly identified the need for time-stamps when using electronic signatures. This has been followed by a range of standardization efforts [1,3]. With the rise of XML as a language to structure communications, XML formats for security protocols were proposed too. In 2002, the OASIS Digital Signature Services Technical Committee (DSS TC) was formed[8]. Its purpose is to develop techniques to support the processing of digital signatures, including the development of a profile for time-stamping, with a focus on XML formats. At the moment of this writing, the time-stamping profile includes support for so-called independent time-stamp tokens. In this paper, we investigate the possibility of including linked time-stamp tokens in the DSS standard, driven by the believe that this kind of time-stamp is more desirable in certain settings, despite its complexity. In the next two sections we give an overview of existing time-stamping schemes and the OASIS DSS standard. This is followed by a description of the structures for linked time-stamp requests and verifications, and its processing.

2 Time-stamping schemes

Digital time-stamping is a set of techniques that enables us to determine if a certain digital document has been created before a given time. A trusted third party – a Time-Stamping Authority (TSA) – implements this by creating time-stamps, the digital assertions that a given document was presented to the TSA at a given time. A common practise is to time-stamp documents that represent new inventions and discoveries (log files, financial audit reports). This helps to establish first-to-invent claims or document authenticity [13]. Time-stamping can also play a role in Public Key Infrastructures (PKI). In this context, time-stamps are used to extend the lifetime of digital signatures: a time-stamp on a digital signature can prove that the signature was generated before the certificate on the signature key-pair was revoked. We distinguish two classes of time-stamping schemes, which are described below.

2.1 Schemes producing independent time-stamps

Simple schemes generate time-stamps that are independent of other time-stamps. A classical example is the digital signature of a TSA on a pair `(time,document)`, which is standardised in RFC3161 [3] and in ISO/IEC FDIS 18014-2 [2]. A limitation of these schemes is that they assume a high level of trust in the TSA, and possible fraudulent behaviour of the TSA remains undetected. The time-stamping profile [9] of DSS is aimed at this kind of time-stamp.

2.2 Schemes producing linked time-stamps

Linking schemes limit the required trust in the TSA by including data from other time-stamps into the computation of the issued time-stamp, such that they depend on each other. Linking happens in three phases:
Aggregation: in the first step, all documents received by the TSA within a small time interval – the aggregation round – are considered as being submitted simultaneously. The output of the aggregation round is a binary string that securely depends on all the documents submitted in that round.
Linking: the output of the aggregation round is linked to previously computed aggregation round values. The resulting value cannot be computed without the existence of previous aggregation round values. This establishes a one-way order between aggregation round values, such that so-called *relative temporal authentication* is obtained: time-stamps of different aggregation rounds can be compared.
Publication: from time to time (e.g., each week), the TSA publishes the most recent time-stamp in a widely witnessed medium, such as a newspaper. By doing this, the TSA commits itself to all of the previously issued time-stamps. The published values are used for verifying time-stamps and they enable other parties to check if the TSA is behaving properly.

Examples of linking schemes can be found in Bayer *et al.* [4], and Buldas *et al.* [6]. In these cases, the linking can be visualised by a graph and optimised in time-stamp size. In Benaloh *et al.* [5] and Merkle [12], some aggregation schemes are proposed.

3 OASIS Digital Signature Services Standard

In October 2002, the Digital Signature Services Technical Committee (DSS TC) was formed within OASIS. The purpose of this TC is to develop techniques (a standard) to support the processing of digital signatures. The core document specifies a simple client/server protocol on which the actual services are built. These services are specified by profiles. The core protocols support the creation and verification of signatures and time-stamps. The core document is aimed at XML Digital Signatures [7] and CMS Signatures [11]. The standard accommodates RFC3161 time-stamps [3] and a DSS XML time-stamp format [9]. XML elements, taken from the W3C XML Digital Signature standard are prefixed by `'ds:'` while elements from the OASIS DSS standard are not prefixed. The new elements, proposed in this paper, are prefixed by `'tsp:'`.

The DSS core protocol is composed of two operational types: one for signature generation and one for signature verification. A typical use of the protocols is submitting a document or its digest value to a DSS server through a <`SignRequest`> element. The DSS server will return a signature on the submitted values, in a <`SignResponse`> element. Later on, the signature can be submitted to the DSS server for verification, by sending a <`VerifyRequest`> element. The essence of the response is a valid/invalid indication, returned in a <`VerifyResponse`> element. The DSS core standard also specifies an XML structure for independent time-stamp tokens. The DSS <`TimeStamp`> children are <`ds:Signature`> and <`RFC3161TimeStampToken`>. The element <`RFC3161TimeStampToken`> allows for the inclusion of a base64-encoded RFC3161 time-stamp. In the <`ds:Signature`> case, a <`TstInfo`> element is placed in <`ds:Object`> and covered by the signature. This <`TstInfo`> element is a XML translation of the RFC3161 `TSTInfo` structure.

The *XML Timestamping Profile* of OASIS DSS [9] restricts the services of the DSS core to a time-stamping protocol. The main restriction is that only hash values (no documents) can be sent to the TSA. Furthermore, for the <`SignRequest/OptionalInputs`>, two values for <`SignatureType`> are proposed to identify the time-stamping schemes mentioned above. Finally, the TSA can include only a <`SigningTime`> optional output in the <`VerifyResponse`>.

The current DSS standard does not specify a structure to integrate linked time-stamps. The authors have knowledge of at least two large commercial TSAs that have deployed a linked time-stamp service. Furthermore, these time-stamping schemes have some advantages over simple schemes. Therefore, we think that linked time-stamps should not be ignored in the OASIS DSS standard. Moreover, as far as we know, there exists no other XML standard that allows for an easy integration of this class of time-stamping schemes. We should note that the X9.95 proposal by ANSI is based on an XML translation of ASN.1 structures; our approach differs by the fact that we start from a standard, based on XML Digital Signatures. In this paper, we present a possible path to include linked time-stamps in the DSS standard through a subprofile of the OASIS DSS XML time-stamping profile in [9]. Our subprofile defines a new XML *linked* time-stamp token, as a sibling of the <`ds:Signature`> and <`RFC3161TimeStampToken`>.

4 Integrating XML linked time-stamps in the XML time-stamping profile of OASIS DSS

4.1 Issuing Protocol

In this section, we describe the XML elements that are exchanged when a client wants to get a certain document time-stamped by the TSA. The main contribution is the definition of a linked time-stamp format, which allows to model most linked time-stamps. This definition is based on previous work in [15], in which an example XML fragment can be found.

Element <SignRequest>

- Element <SignRequest/OptionalInputs/SignatureType> The content of the optional input <SignatureType> should identify the requested linking scheme. An example for identifying an existing linking scheme could be ee:cyber:timestamp.
- Element <SignRequest/InputDocuments/DocumentHash> As described in [8], the digest value contained in each <DocumentHash> is the result of applying a digest method, specified in the same element, to one or more documents. If a client wants to time-stamp the same content with several digest algorithms, several <InputDocuments> elements can be included, each containing a <DocumentHash> element with a different digest algorithm. This is useful if one of the hash functions gets compromised as shown in [14].

Element <SignResponse> The server must return a <Timestamp> signature object as defined in [9]. The new <tsp:LinkedTimestamp> element will be a child of the <SignResponse/SignatureObject/Timestamp> element. Its structure is presented in Figure 1 and we describe its key aspects below.

- Element <InputDocuments>. This element should be copied from the <SignRequest> element. In the DSS TST profile [9], these values are copied into the signature, so the resulting time-stamp contains them by default. As in our profile, it is optional to sign these values directly, we need to copy them somewhere else to reconstruct the input of the aggregation (or linking) operation.
- Element <TstInfo>. This element is optional and can contain values as discussed in the DSS core.
- Element <ds:Signature>. In our profile, the signature is optional and can only contain <ds:Reference> elements pointing to the <InputDocuments>, <tsp:BindingInfo> and/or <TstInfo> children of its <tsp:LinkedTimestamp> parent. This signature can be discarded from the time-stamp, once the linking round finishes. After that stage, the evidence value of the time-stamp lies in the binding information, rather than in the signature.
- Element <tsp:BindingInfo>. This element should contain the binding information of the linked time-stamp. This element is used as follows:

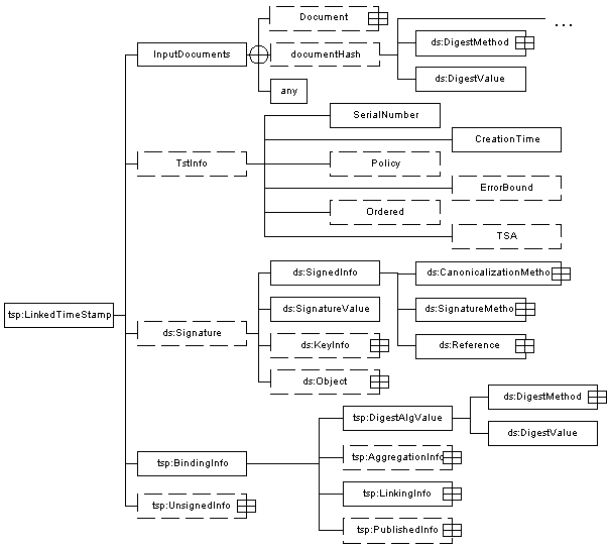

Fig. 1. tsp:LinkedTimestamp element

- The <`tsp:DigestAlgValue`> element contains the digest value that is passed on to the linking scheme. This value is obtained as follows: First, we build a node set using an XPath expression which selects the `descendant-or-self` elements and attributes of the <`DigestMethod`> and the <`DigestValue`> in the <`DocumentHash`> elements that have been copied into <`tsp:LinkedTimestamp/dss:InputDocuments`> element. Optionally, we can also attach <`TstInfo`> to the node set. Next, we take the `excl-CN14` transform of this node set which should result in an octet string. This octet string is hashed using the specified digest method in <`tsp:DigestAlgValue`>.
- The <`tsp:AggregationInfo`> element, if present, specifies the aggregation algorithm and the necessary data to compute the output of the aggregation round using the <`tsp:DigestAlgValue`> element.
- The <`tsp:LinkingInfo`> element contains the algorithm and data to compute the value of the linking round, given the output of the aggregation round.
  - <`tsp:Head`> contains linking information from time-stamps issued before this one.
  - <`tsp:Tail`> contains information from time-stamps after this one. It is computed by the TSA at the end of the linking round. In most cases, this element will not be present in the <`SignResponse`> element, as the necessary information is not available at the time of generating the time-stamp. This information will be added in the

verifying protocol. How this affects the signature, is discussed in the next section.
    * <ds:Object> contains information that is 'unnatural' to include directly into <tsp:Head> or <tsp:Tail>, but is used in some linking schemes. It can be referenced from within these elements.
  - The <tsp:PublishedInfo> contains round values for linking rounds, plus the location where they can be retrieved or verified.
- Element <tsp:UnsignedInfo>. If the signature on an existing time-stamp should remain valid after completion of the time-stamp with values generated after this time-stamp, additional information can be placed here. As it is not covered by the signature, adding things here will not break the signature. This element will not be present in the <SignResponse> element, but it will be added in the verifying protocol.

4.2 Verifying protocol

Here, we describe the XML elements used in the verification of linked time-stamps. The biggest challenge here is enabling its comparison and extension within the DSS approach. We give a short description of these operations, and describe how the DSS elements can be used to realise them.

- **Verify a time-stamp TS1 against another time-stamp TS2**.
  Upon this request, the verifier should get a response from the server indicating one of the following cases: (a) TS1 was issued before TS2 ('earlier'), (b) TS1 was issued after TS2 ('later'), (c) an error. The server can determine the response using the times in the signed information, or using the linking information between the two time-stamps directly. Our protocol also allows the verification of a timestamp TS1 alone, not against another one.
- **Update a time-stamp TS1 to a published value PV or to an arbitrary second time-stamp TS2**.
  Updating a time-stamp means that the linking information in that time-stamp is updated such that the link between the time-stamp and a certain other value can be computed with the information held in the time-stamp. Updating TS1 to a PV means one of the following cases: (a) the completion of the time-stamp within its same round, or (b) the extension to the last published value. Depending on the time when the update is requested, a completion (a) or an extension (b) should be returned. For most linking schemes, the extension to the PV should include the completion of an (incomplete) time-stamp.
  If a completion is done, the signed <tsp:BindingInfo> is replaced by a new <tsp:BindingInfo>. If an extension is done, the new linking information can be placed in <tsp:UnsignedInfo> or in <tsp:BindingInfo>. In the last case, a new <ds:Signature> has to be computed to replace the one present in the time-stamp, if the signature is still needed.
  Extending the time-stamp TS1 to other time-stamp TS2 means that the server should include in the response enough information (chain of digest

values and optionally published values) to allow the client to verify the temporal relationship between TS1 and TS2. In most cases this means building the hash chain that passes through both time-stamps TS1 and TS2. As in the case of updating to a PV, this new information can be signed or not.
It is assumed in the protocol that TS1 has been issued earlier than TS2. If it is not this case, the server should return a response indicating that the client can change their order and make a new request for the extension.

Next, we describe the elements that allow the functionality above.

Element <VerifyRequest> Important modified children of this element are:

- Element <OptionalInputs>. Contrary to [9], in this protocol the following optional inputs are allowed.
  - Element <tsp:RelativeTimestamp> can contain a <Timestamp> child, a second time-stamp TS2. Alternatively, its URI attribute can refer to TS2 (using, for example, the serial number). This is an optional element, but is mandatory if the optional input <tsp:CompareLinkedTimestamp> is present in the verify request. The time-stamp TS1 in the <SignatureObject> element will be updated or compared to TS2.
  - Element <tsp:CompareLinkedTimestamp> is an empty element which indicates that time-stamp TS1 must be compared to the time-stamp TS2 contained in the <tsp:RelativeTimestamp> optional input, that must be present in this case.
  - Element <UpdateLinkedTimestamp> indicates that the client wants to update his time-stamp TS1. This optional element has an URI attribute that can contain one of the following items: a local reference to the ID attribute of the <tsp:RelativeTimestamp> optional input, a reference to a published value, or an empty reference. If in the last case, TS1 should be updated to the most recent published value.
- Element <SignatureObject>. The client sends a <Timestamp> element containing the time-stamp TS1.
- Element <InputDocuments>. The client must only send <DocumentHash> elements; <Document> elements are not allowed.

Element <VerifyResponse> This element contains the TSA's response. It holds a status code and optionally and updated time-stamp.

- Element <Result>. Our profile defines additional <ResultMinor> children of <Result>, all of them prefixed with urn:oasis:names:tc:dss:1.0:resultminor:. If the verification is successful, the server returns:
  - ValidLinkedTimestamp_Earlier: TS1 has been found earlier than TS2.
  - ValidLinkedTimestamp_Later: TS1 has been found later than TS2.
  - LinkedTimestamp_Updated: TS1 was updated.

  Otherwise, if verification has failed, the following <ResultMinor> codes may be returned:

- `IncorrectTimestamp`: The time-stamp fails to verify, indicating that the time-stamp was modified, or that the time-stamp has been computed incorrectly.
- `IncomparableTimestamps`: TS1 cannot be compared to TS2. A possible reason might be that they are in the same aggregation round.
- `IncorrectOrder`: Updating TS1 to a certain value V failed because V existed prior to TS1. We only allow forward extensions in our protocol.
- `NoPublishedValue`: There is no new published value yet.

– Element <`OptionalOutputs`>. Our profile defines <`tsp:UpdatedLinkedTimestamp`>, as an optional child of <`OptionalOutputs`>. This element shall contain the original linked time-stamp TS1 with some additional information added to it (completion or extension information). Optionally, the added information can be signed, depending on where the information is added, as explained above.

4.3 Processing of XML linked time-stamp tokens

Signing protocol Upon receiving a <`SignRequest`> a DSS server will form a <`SignatureObject/Timestamp/tsp:LinkedTimeStamp`> as follows: First, it copies <`SignRequest/InputDocuments`> into a <`tsp:LinkedTimeStamp`> element. Then, optionally, the server computes a <`TstInfo`> element and enters it as the second child of <`tsp:LinkedTimeStamp`>. Next, the server computes the <`tsp:BindingInfo`> as it is described in section 4.1 and a <`ds:Signature`> element according to [7], which are entered as the third and fourth child of <`tsp:LinkedTimeStamp`>. The server may include <`ds:Signature/SignedInfo/Reference`> elements pointing to <`InputDocuments`>, <`TstInfo`> (optional) and <`tsp:BindingInfo`> elements. Then, an appropriate <`Result`> element is generated, depending on if the previous steps were successful. Finally, the <`Result`> and the <`SignatureObject`> elements are entered into a <`SignResponse`>, and returned to the requester.

Verifying protocol Upon receiving a <`VerifyRequest`>, a DSS server will perform the following steps. If this fails, the appropriate error will be returned in the <`Result`> element.

– **Verification.**
There must exist a <`SignatureObject/Timestamp/tsp:LinkedTimeStamp`> element (TS1) present as a child of the <`VerifyRequest`>. First, the server gets the <`TstInfo`> element if present, and verifies that the <`Policy`> contained in it is acceptable according to the relying party's policy. Then, the server gets the <`tsp:BindingInfo`> element and verifies that the <`tsp:BindingInfo/tsp:DigestAlgValue`> element value has been computed taking as inputs the children of <`InputDocuments`> and, optionally, the <`TstInfo`> element. After that, the server must verify the <`tsp:AggregationInfo`> (if present) and <`tsp:LinkingInfo`> elements as specified by the <`tsp:BindingInfo`> algorithm. This includes retrieving the trust anchors (published

reference values) needed to check the time-stamp and comparing them to the reference values stored in <tsp:Head>, <tsp:Tail> and <tsp:Published-Info>. If the element <ds:Signature> is present, the server performs the standard checks on the signature keys as specified in the OASIS DSS standard. After checking that the <ds:Reference> elements point to <Input-Documents>, <TstInfo> and <tsp:BindingInfo> elements, the server must verify all digests and the signature according to [7].

- **Comparison.**
  If there exists a <tsp:CompareLinkedTimeStamp> optional input, the server should verify that there is also a <tsp:RelativeTimestamp> optional input. In this case, the server must retrieve the <tsp:LinkedTimestamp> element TS2 from the element <tsp:RelativeTimestamp> and verify it as described above (*Verification* step). Then, the server builds the chain of digests between the two time-stamps according to the binding algorithm and the time-stamping policy. This step will determine the temporal relation between the time-stamps.
- **Update.**
  If there exists a <tsp:UpdateLinkedTimeStamp> optional input which contains an URI attribute pointing to <tsp:RelativeTimestamp> optional input, and it has not been verified during the *Comparison* step, the server must verify it. If the URI attribute points to a published value PV or another time-stamp TS3, the server should verify that it is a correct identifier, and retrieve and verify PV or TS3. Then, the server should build the chain of digests that passes through the time-stamp TS1 and the requested extension point (TS3, PV or most recent PV). If the linked time-stamp can be extended to that extension point, the extension information is placed in <tsp:BindingInfo> or in <tsp:UnsignedInfo>, depending on the binding algorithm and the time-stamping policy.

5 Implementation

To implement the scheme, we can start from a standard DSS implementation, with adjustments to handle linked time-stamp requests. The implementation will have a first presentation tier which receives the request, verifies the syntax and determines to which service module it has to be dispatched.

The service modules in the second tier will determine the nature of the request (time-stamp generation/verification), and will pass the request to a suitable plug-in component in a third tier. If the server supports several linking schemes, the element <SignRequest/OptionalInputs/SignatureType> indicates the specific linked time-stamp scheme asked by the client and this should help the server to allocate the requests; otherwise the server should know which one should be applied. In the case of <VerificationRequest>s, there is no specific indication of the applied linked time-stamp in the main body, but the <LinkedTimeStamp> element should carry enough information to allow the server to identify which kind of linked time-stamp contains, and therefore, which linked time-stamp module it may be assigned to.

6 Conclusions

In this paper, we sketched a path to include linked time-stamp tokens in the OASIS DSS standard [8] by making a sub-profile of the existing time-stamping profile [9] for this standard. We discussed several points at which our subprofile collides with the original profile, and we think that some changes to the standard could help the integration of linked time-stamps. We hope to have an impact in the DSS standardization body to which we will present a fully elaborated version of this paper.

References

1. ISO/IEC 18014-1. Information technology – Security techniques – Time-stamping services – Part 1: Framework, 2002.
2. ISO/IEC 18014-2. Information technology – Security techniques – Time-stamping services – Part 2: Mechanisms producing independent tokens, 2003.
3. C. Adams, P. Cain, D. Pinkas, and R. Zuccherato. Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP). `www.ietf.org/html.charters/pkix-charter.html`, April 2002.
4. Dave Bayer, Stuart Haber, and W. Scott Stornetta. Improving the Efficiency and Reliability of Digital Time-Stamping. In *Sequences II: Methods in Communication, Security and Computer Science*, pages 329–334. Springer-Verlag, 1993.
5. J. Benaloh and M. de Mare. One-way Accumulators: A Decentralized Alternative to Digital Signatures. In T. Helleseth, editor, *Advances in Cryptology - Proceedings of EuroCrypt '93*, volume 765 of *Lecture Notes in Computer Science*, pages 274–285, Lofthus, Norway, May 1993. Springer-Verlag.
6. Ahto Buldas, Helger Lipmaa, and Berry Schoenmakers. Optimally Efficient Accountable Time-Stamping. In *Public Key Cryptography - PKC'2000*, number 1751 in Lecture Notes in Computer Science, pages 293–305. Springer-Verlag, 2000.
7. D. Eastlake, J. Reagle, and D. Solo. XML-Signature Syntax and Processing. `www.w3.org/Signature`, February 2002.
8. T. Perrin et *al.* OASIS Digital Signature Services TC. Digital Signature Service (DSS) Core Protocols, Elements and Bindings, Working Draft 26. `www.oasis-open.org`, June 2004.
9. T. Perrin et *al.* OASIS Digital Signature Services TC. XML Timestamping DSS Profile, Working Draft 06. `www.oasis-open.org`, June 2004.
10. European Committee for Standardization CEN. CWA 14171: Procedures for Electronic Signature Verification. `www.cen.eu.org`, 2001.
11. R. Housley. Cryptographic Message Syntax. `http://www.ietf.org/html.charters/smime-charter.html`, April 2002.
12. Ralph C. Merkle. Protocols for public key cryptosystems. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 122–134, 1980.
13. Surety. Surety AbsoluteProof Solution Suite . `www.surety.com`.
14. Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Hongbo Yu. Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD. Rump session of CRYPTO 2004, available at `http://eprint.iacr.org/2004/199.pdf`, August 2004.
15. Karel Wouters, Bart Preneel, Ana Isabel González-Tablas, and Arturo Ribagorda. Towards an XML Format for Time-Stamps. In *ACM Workshop on XML Security 2002*. ACM, ACM, November 2002.