Journal of Web Engineering, Vol. 2, No. 3 (2004) 148–175

A WEB SERVICES BASED ARCHITECTURE FOR DIGITAL TIME STAMPING

A. Cilardo, A. Mazzeo, L. Romano,[a] G. P. Saggese
Dipartimento di Informatica e Sistemistica, Universita' degli Studi di Napoli Federico II, Via Claudio 21 Naples 80125, Italy
{*acilardo, mazzeo, lrom, saggese* }*@unina.it.*

G. Cattaneo
Finmatica SpA, Via Amato, 15 Salerno 84131 , Italy
g.cattaneo@finmatica.com

Received September 3, 2003
Revised Octomber 27, 2003

This paper describes the results of a research activity conducted cooperatively by an academic and an industrial party. It presents a practical solution for and an experience in the implementation of time stamping services and their exposition to the Internet. We present the main state-of-the-art algorithms for time stamping applications, thoroughly discuss pros and cons of each technique, and highlight the crucial issues raised by their practical implementation. Then we present an architecture which provides both relative temporal authentication, based on a linear linking scheme, and absolute temporal authentication, based on publishing mechanisms as well as on a trusted time source. In order to guarantee ubiquity and interoperability, the actual implementation of the proposed architecture relies on the emerging Web services technology for exposing time stamping functions to the Internet. Experimental tests have demonstrated the effectiveness of the proposed solution.

Keywords: Time Stamping, Cryptography, Web Services, Multi-Tier Architectures
Communicated by: M Gaedke, P Fraternali & G Rossi

1 Rationale and Contribution

Software systems have functional requirements (i.e., what services the system has to provide), and non functional requirements (i.e., the quality the system must guarantee in the delivery of such services). Typical functional requirements are business-specific services, and typical non functional requirements are security and performance.

Security has become a key requirement for the vast majority of current applications. As systems are being opened to the Internet, commercial traders, financial institutions, service providers, and consumers are exposed to a variety of potential damages, which are often referred to as electronic risks. These may include direct financial loss resulting from fraud, theft of valuable confidential information, loss of business opportunity through disruption

[a]Luigi Romano, Dipartimento di Informatica e Sistemistica, University of Naples, Via Claudio, 21 80125 Napoli - Italy. e-mail: lrom@unina.it Tel: +39.081.7683834 Fax: +39.081.7683816.

of service, unauthorized use of resources, loss of customer confidence or respect, and costs resulting from uncertainty. In order to mitigate risks and promulgate the deployment of information systems in open networked environments, applications must guarantee security features such as authentication, authorization, confidentiality, integrity, and non-repudiation. A particularly challenging issue is how to guarantee that long lived electronic documents can be verified over the years. To achieve this goal, it is crucial that reliable digital time stamping features be made available.

During the last years, especially in the context of legal regulation of digital signature techniques, the organizational and legal aspects of time stamping itself have become the subject of world-wide attention, both in academia and in the industry. Time stamping is a set of techniques enabling one to ascertain whether an electronic document was created or signed at a certain time. Without time stamping we neither can trust signed documents when the cryptographic primitives used for signing have become unreliable nor we can solve the cases when the signer himself repudiates the signing, claiming that he[a] has accidentally lost his signature key. Unfortunately, unlike physical objects, digital documents do not comprise the seal of time. Thus, the association of an electronic document uniquely with a certain moment of time is very complicated, if not impossible at all.

In the literature, time stamping schemes are classified into three distinct types [14]: simple, linking, and distributed schemes.

In the *simple* scheme, a time stamp is generated by a trusted third party (the Time Stamping Authority, TSA) in such a way that it does not involve data included in other time stamps. The main weakness of this scheme is that the TSA has to be unconditionally trusted: if the TSA fraudulently alters the time parameter of a certain time stamp, nobody can detect the alteration. Also, if a leakage of the signature key of the TSA has occurred, fake time stamps can be forged at will. Some ten years ago the only known time stamping methods were based on the use of linear schemes. Thus, applications which needed digital time stamping had no choice but resorting to TSAs, which they had to trust unconditionally. In 1991 the seminal publication [11] of Haber and Stornetta showed that the trust to the TSA can be greatly reduced by using the linking schemes or alternatively the distributed schemes. Several papers were published during the last years, which further improved the original schemes [3, 2, 12, 4].

The basic idea behind the *linking scheme* is to generate a time stamp which involves data included in other time stamps. A chain of time stamps is constructed, typically by using a one-way hash function. If an issuer is willing to deliberately alter or forge a certain time stamp, he has to alter all the related time stamps. For this reason it is more difficult for an issuer to manipulate a time stamp in the linking scheme than in the simple scheme.

Finally, in the *distributed scheme* multiple issuers independently generate a time stamp according to the simple scheme, each using his own key and time source. The set of issuers designated to sign the time stamp is chosen randomly, in such a way that the submitter cannot determine them a priori. If the number of signing issuers is less than a specific predetermined number, they cannot produce a correct time stamp. This scheme relies on the difficulty for

[a]Throughout this paper *he* and *she* will be used interchangeably.

the submitter to collude with a large enough number of issuers so to be able to complete the stamping process. However, the need for a large number of independent issuers (who could possibly be the users of the service themselves) makes the distributed scheme rather unpractical in most practical situations.

In conclusion, at the time of this writing, the algorithms used to provide time stamping services in real-world scenarios rely on linking schemes.

This paper presents a web-based architecture for implementing and exposing to the Internet time stamping services. The design, the implementation, and the validation of the architecture are the outcome of a years long research activity conducted cooperatively by an academic and an industrial party. Preliminary results of such an activity have already been described in [6, 7, 8].

The architecture uses a round-based linear linking scheme. Time stamping requests falling in a given time window are gathered in a tree-like structure and compressed into a single data item to be signed and linked with the results coming from the previous time windows. This technique improves performance and dramatically reduces the quantity of data to be signed and linked, as compared to the plain linear linking scheme which shows prohibitive trust and broadcast requirements. The width of the time window represents the time resolution by which relative temporal order between two distinct stamps can be established. For the structure of rounds we chose a tree-like scheme since it allows for an easy and efficient implementation. Intermediate results of the time stamping process are to be periodically spread via a variety of media so that the trustworthiness of the issuer is no longer needed for verifying correctness of stamps issued in the past. The architecture relies on a multi-tier structure. The back-end server is in charge of the bulk of the algorithmic issues. The middle-tier is in charge of leveraging services provided by the back-end, in order to satisfy interoperability requirements which arise from the heterogeneity of the service requestors. To implement the middle-tier, we resorted to the emerging Web services technology. Web services are a promising technology allowing for a flexible interaction between applications over the Web. Web services are to be thought of as self-contained, modular applications that can be described, published, located, and invoked over the Web. Through wrappering the underlying plumbing, services insulate the application programmer from the lower layers of the programming stack. This allows services to enable virtual enterprises to link their heterogeneous systems and to participate in single, administrative domain situations, where other communications mechanisms can provide a richer level of functionality.

The work presented in this paper makes three important contributions.

First, it provides a comprehensive overview of the main state-of-the-art algorithms for time stamping applications and thoroughly discusses pros and cons of each technique. Algorithms are analyzed in terms of security and efficiency, with respect to the actual implementation of both the issuing and the verification processes.

Second, it presents a practical solution for and an experience in the implementation of time stamping services and their exposition to the Internet. This makes a valuable resource for practitioners. That is especially true of a fields – such as computer security – where for most commercial product vendors do not provide details about implementation, performance,

and development and deployment costs. In particular, we address the integration problems which arise when a potentially large community of users, usually relying on heterogeneous technologies, are willing to access a remote third-party time stamping service.

Third, it provides valuable evidence from the field regarding best practices for wide scale deployment of time stamping services using Commercial Off-The-Shelf (COTS) software products.

The rest of the paper is organized as follows. Section 2 provides an overview of the Web services architecture. Section 3 contains a thorough analysis of the schemes used to provide time stamping services. Section 4 describes the architecture of the proposed system. Section 5 details the actual implementation of the architecture and system operation. Section 6 shows results of experimental tests and a case-study application. Section 7 presents some considerations on the design experience and lesson learned. Section 8 gives some examples of existing implementations of time stamping services. Section 9 concludes the paper with some final remarks.

2 The Web Services Framework

A *Web service* [16] is an interface that describes a collection of operations that are network-accessible through standardized eXtensible Mark-up Language (XML) messaging. A Web service is described using a standard, formal XML notion, called its *service description*, which covers all the details necessary to interact with the service, including message formats (that detail the operations), transport protocols, and location. The interface hides the implementation details of the service, allowing it to be used independently of the hardware or software platform on which it is implemented and also independently of the programming language in which it is written. This allows and encourages Web services-based applications to be loosely coupled, component-oriented, cross-technology implementations. Web services fulfil a specific task or a set of tasks. They can be used alone or with other Web services to carry out a complex aggregation or a business transaction.

The Web services architecture is based upon the interactions between three roles: *service provider*, *service registry*, and *service requestor*. The interactions involve the publish, find and bind operations. Together, these roles and operations act upon the Web services artifacts: the Web service software module and its description. In a typical scenario, a service provider hosts a network-accessible software module (an implementation of a Web service). The service provider defines a service description for the Web service and publishes it to a service requestor or service registry. The service requestor uses a find operation to retrieve the service description locally or from the service registry and uses the service description to bind with the service provider and invoke or interact with the Web service implementation. Service provider and service requestor roles are logical constructs and a service can exhibit characteristics of both. Figure 1 illustrates these operations, the components providing them, and their interactions.

The Web services architecture consists of several layers. To perform the three operations of publish, find and bind in an interoperable manner, there must be a Web services stack that embraces standards at each level.

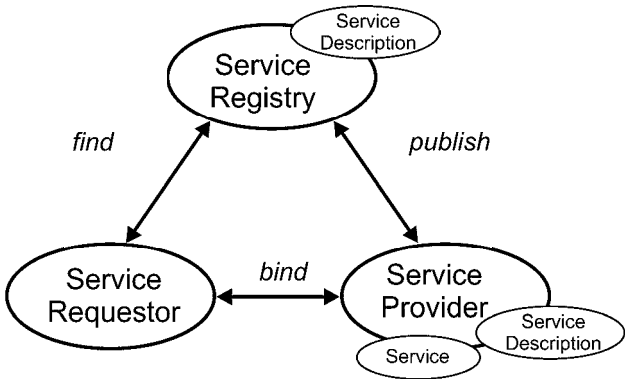

Fig. 1. The Service Oriented Architecture for Web services.

The foundation of the Web services stack is the network. Web services must be network-accessible to be invoked by a service requestor. Web services that are publicly available on the Internet use commonly deployed network protocols. Because of its ubiquity, the Hypertext Transfer Protocol (HTTP) is the *de facto* standard network protocol for Internet-available Web services. Other Internet protocols can be supported, including the Simple Mail Transfer Protocol (SMTP) and the File Transfer Protocol (FTP).

The next layer, XML-based messaging, represents the use of XML as the basis for the messaging protocol. The Simple Object Access Protocol (SOAP) is the chosen XML messaging protocol for many reasons:

- It is the standardized enveloping mechanism for communicating document-centric messages and remote procedure calls using XML.
- It is simple, since it is basically an HTTP POST with an XML envelope as payload.
- It is preferred over simple HTTP POST of XML because it defines a standard mechanism to incorporate orthogonal extensions to the message using SOAP headers and a standard encoding of operation or function.
- SOAP messages support the publish, find and bind operations of the Web services architecture.

The interoperable base stack – illustrated in Figure 2 – provides for interoperability and enables Web services to leverage the existing Internet infrastructure. This creates a *low cost of entry* to a ubiquitous environment. Flexibility is not compromised by the interoperability requirement, because additional support can be provided for alternative and value-add protocols and technologies.

The service description layer is actually a stack of description documents. First, the Web Service Definition Language (WSDL) is the *de facto* standard for XML-based service description. This is the minimum standard service description necessary to support interoperable Web services. WSDL defines the interface and mechanics of service interaction. Additional

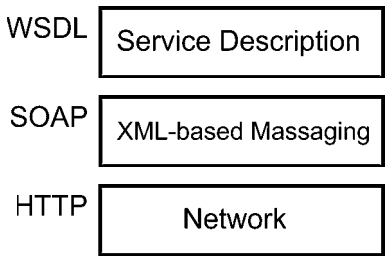

Fig. 2. Interoperable base Web services stack.

description is necessary to specify the business context, qualities of service, and service-to-service relationships. The WSDL document can be complemented by other service description documents to describe these higher level aspects of the Web service. For example, business context is described using the Universal Description, Discovery and Integration (UDDI) data structures in addition to the WSDL document. Service composition and flow are described in a Web Services Flow Language (WSFL) document.

Because a Web service is defined as being network-accessible via SOAP and represented by a service description, the first three layers of this stack are required to provide or use any Web service. The simplest stack would consist of HTTP for the network layer, the SOAP protocol for the XML messaging layer and WSDL for the service description layer. This is the interoperable base stack that all inter-enterprise, or public, Web services should support. Web services, especially intra-enterprise, or private, Web services, can support other network protocols and distributed computing technologies.

Two additional layers, namely service publication and service discovery, can be implemented on top of the interoperable basic stack. Any action that makes a WSDL document available to a service requestor, at any stage of the service requestors lifecycle, qualifies as service publication. The simplest, most static example at this layer is the service provider sending a WSDL document directly to a service requestor. This is called direct publication. E-mail is one vehicle for direct publication. Direct publication is useful for statically bound applications. Alternatively, the service provider can publish the WSDL document describing the service to a host local WSDL registry, private UDDI registry or the UDDI operator node.

Because a Web service cannot be discovered if it has not been published, service discovery depends upon service publication. The variety of discovery mechanisms at this layer parallels the set of publication mechanisms. Any mechanism that allows the service requestor to gain access to the service description and make it available to the application at runtime qualifies as service discovery. The simplest, most static example of discovery is static discovery wherein the service requestor retrieves a WSDL document from a local file. This is usually the WSDL document obtained through a direct publish or the results of a previous find operation. Alternatively, the service can be discovered at design time or runtime using a local WSDL registry, a private UDDI registry or the UDDI operator node.

Because a Web services implementation is a software module, it is natural to produce Web services by composing Web services. A composition of Web services could play one of several roles. Intra-enterprise Web services might collaborate to present a single Web service

interface to the public, or the Web services from different enterprises might collaborate to perform machine-to-machine, business-to-business transactions. Alternatively, a workflow manager might call each Web service as it participates in a business process. The topmost layer, service flow, describes how service-to-service communications, collaborations, and flows are performed. WSFL is used to describe these interactions.

For a Web services application to meet the stringent demands of todays e-businesses, enterprise-class infrastructure must be supplied, including security, management and quality of service. These vertical requirements must be addressed at each layer of the stack. The solutions at each layer can be independent of each other. More of these vertical requirements will emerge as the Web services paradigm is adopted and evolved.

The bottom layers of the stack, representing the base Web services stack, are relatively mature and more standardized than the layers higher in the stack. The maturation and adoption of Web services will drive the development and standardization of the higher levels of the stack and the vertical requirements.

3 Time Stamping Schemes

The main security objective of time stamping is *temporal authentication*, i.e. the ability to prove that a certain document has been created at a certain moment of time. As a definition applicable in legal situations, a time stamping system can be thought of as a set of supervisors, a time stamping server, and a triple $(S; V; A)$ of protocols [4]. The stamping protocol S allows each participant to submit a message. The verification protocol V is used by a supervisor having some time stamps to verify the absolute submission time of individual stamps or the relative temporal order of all stamps. The audit protocol A is used by a supervisor to verify whether the time stamping server carries out his duties. Additionally, nobody should be able to produce fake time stamps without being caught. A time stamping system should be able to handle time stamps which are anonymous and do not reveal any information about the content of the stamped data. The time stamping system is not required to identify the initiators of time stamping requests.

3.1 Simple Scheme

As we explained in the introductory section, the *simple* time stamping scheme relies only on a trustworthy third party (the Time Stamping Authority, TSA) which is in charge of certifying the time by signing the stamp. Thus, the stamping protocol S consists of a simple digital signature of the requestor's message associated with the submission time. More precisely, a time stamp is issued as follows.

1. The client sends a document x (or a hash value of it) to the TSA;
2. The TSA appends the current time t and the TSA identifier ID to the submitted document, and signs the composite document (ID, t, x);
3. The TSA returns the two values t and $s = sig_{TSA}(ID, t, x)$ to the client.

The verification protocol V checks the integrity of the digital signature, and establishes relative temporal order between two stamps by comparing their absolute time. The audit protocol A can only reveal an improper behaviour of the TSA by submitting trial time stamping

requests and checking the correctness of the certified time values. However, this obvious technique is unable to check a generic time stamp previously issued and correctly signed; one should consider such a stamp as a trustworthy proof even if nobody verified its correctness at issuance time.

As we have already pointed out, this time stamping system is rather weak since it relies on an unconditionally trusted third party, which is wholly liable for the issued stamps. Moreover, in the event that the secrecy of the issuer's signature key is compromised, there is no way to distinguish a genuine stamp from a forged one.

3.2 Linking Schemes

Linking schemes have been introduced to overcome these drawbacks. Basically, they are based on the following principle:

> Although the creation of a digital data item is an observable event in the physical world, the moment of its creation cannot be ascertained by observing the data itself; however, it is possible to check the relative temporal order of the created data items (i.e., prove the relative temporal authentication, RTA) using one-way dependencies defining the arrow of time.

All the proposed time stamping linking schemes realize one-way dependencies by means of the so-called *collision free one-way hash functions.* These include families of functions $h : \{0,1\}^* \rightarrow \{0,1\}^l$ compressing bit-strings of arbitrary length to bit-strings of a fixed length l, with the following properties:

- The functions h are easy to compute, and it is easy to pick a member of the family at random.
- It is computationally infeasible, given one of these functions h, to find a pair of distinct strings x, x' satisfying $h(x) = h(x')$. Such a pair is called a *collision* for h.

Hash functions have a number of good properties well suited for all kinds of time stamping schemes. One can hash the document x to be time stamped, and submit only the hash value $y = h(x)$ to the time stamping authority. For the purpose of authentication, stamping y is equivalent to stamping x. Resorting to hash functions solves a privacy issue, since the content of the document to be time stamped need not to be revealed. The originator of the document computes the hash value himself, and sends it to the time stamping service. The plain document is only needed for verifying the time stamp. This is very useful for many reasons (such as protecting something that one might want to patent). Furthermore, only a small, fixed-length message is to be submitted to the time stamping authority, greatly reducing the bandwidth problems which would arise if the whole document x were to be processed. With a secure signature scheme available, when the time stamping authority receives the hash value, it builds the time stamp token, then signs this response and sends it to the client. By checking the signature, the client is assured that the time stamping authority actually did process the request and that the hash value was correctly received. Depending on the design goal for an implementation of time stamping, there may be a single hash function used by everybody, or different hash functions for different users.

In the time stamping linking scheme, hash functions are mainly used to produce time dependencies between issued time stamps, based on the following consideration: if h is a collision-free one-way hash function, and the values $h(x)$ and x are known to a supervisor P at a moment t, then someone (possibly P himself) used x to compute $h(x)$ at a moment prior to t.

The general stamping protocol S followed by a TSA in the linking scheme can be summarized as follows:

1. Clients send documents (or hash values) to the TSA;
2. The TSA combines requests from individual clients which arrive within a given time window, along with certain values related to stamps issued in the past;
3. The TSA signs a composite document which is a function of a number of issued stamps;
4. The TSA returns the time stamps to the clients.

Obviously, with such a scheme it is hard to produce fake time stamps because forging a single stamp means forging all verifiable dependencies.

Moreover, the need for a trusted TSA can be greatly reduced with the linking schemes by periodically publishing the values used to create the dependencies. Actually, if one can demonstrate the dependency of a stamp on some widely accepted data (for example, a hash value weekly published on a printed newspaper), the TSA is no longer involved in the verification process. The verification protocol V can thus follow the dependency path from the stamp in question to the closest published piece of data. In addition, the dependencies can be used to establish the relative temporal order between two stamps. The audit protocol A consists in checking the integrity of the linking scheme, based on the published values. The audit can be accomplished at any time and allows the verifier to demonstrate not only whether, but also *when* the linking process was altered and thus to distinguish genuine stamps from unreliable ones (with a certain time resolution depending on the publishing rate).

3.3 Distributed Scheme

In [11] an alternative approach is proposed based on *distributed trust*. This scheme needs not use a centralized time stamping server at all. It is assumed that there is a secure signature scheme so that each user can sign messages, and that a standard secure pseudorandom generator G is available to all users. A *pseudorandom generator* is an algorithm that stretches short input *seeds* to output sequences that are indistinguishable by any feasible algorithm from random sequences; in particular, they are unpredictable.

The stamping protocol S followed by a user in the distributes scheme is as follows:

1. Given the hash value y of the document to be time stamped, the pseudorandom generator is used to derive a k-tuple of client identification numbers:

$$G(y) = (ID_1, ID_2, \ldots, ID_k).$$

   Each of the clients corresponding to $ID_1, ID_2, \ldots, ID_k$ receives the time stamping request along with y and the submitting identifier ID.

2. Each of the clients designated by means of $G(y)$ signs the message (t, ID, y) including the time t, and returns the signed message s_j. The time stamp consists of $[(y, ID), (s_1, \ldots, s_k)]$.

As in the simple scheme, the verification protocol V checks the integrity of the digital signature of all responses, while the audit protocol A can detect breaches in the system by submitting trial time stamping requests and checking the correctness and the consistency of all returned time values. As pointed out above, this technique is unable to check a generic time stamp previously issued and correctly signed.

The strength of this scheme is that the submitter should be able to collude with all selected clients $ID_1 \ldots ID_k$. The properties of the pseudorandom generator G ensure it is computationally impossible to find a document y which produces some predetermined $ID_1 \ldots ID_k$. To obtain a fake stamp, the submitter should collude with k random chosen clients. k could be set so that the probability of such a collusion is kept under a certain level.

The distributed scheme has a number of drawbacks. First, the set of possible clients should be established a priori, and the pseudorandom generator should determine only existing clients. Even if this condition is assured, availability of random chosen clients could easily determine a weakness in the system. Moreover, the bandwidth required for a single time stamp issuance would be k times larger than that for the simple scheme. Further problems are related to the synchronization of many distinct time sources and to the legal definition of liability for the certified time. In fact, the distributed scheme, proposed along with the linking scheme in 1991, has not been studied afterwards.

3.4 Evolution of Linking Schemes

The most widely known linking schemes are the Haber and Stornetta's ones. The first scheme proposed by Haber and Stornetta is referred to as *linear* linking scheme [11]. In order to diminish the need for trust, the time stamping authority links all time stamps together into a chain using the collision-free hash function h. In this case the time stamp for the nth submitted document y_n is $s = sig_{TSS}(n, t_n, ID_n, y_n, L_n)$, where t_n is the current time, ID_n is the identifier of the submitter and L_n is the linking information defined by the recursive equation

$$L_n := (t_{n-1}, ID_{n-1}, y_{n-1}, h(L_{n-1})).$$

There are several complications with the practical implementation of this scheme. First, the number of steps needed to verify the one-way relationship between two time stamps is linear with respect to the number of time stamps between them. Hence, a single verification may be as costly as it was to create the whole chain. It has been shown that this solution has impractical trust and broadcast requirements. Haber and Stornetta proposed a modification where every time stamp is linked with $k > 1$ time stamps directly preceding it. This variation decreases the requirements for broadcast by increasing the space needed to store individual time stamps.

In reference [12] Haber and Stornetta proposed a *tree-like* scheme, based on Merkle's authentication trees [18, 19], in which the time stamping procedure is divided into rounds. Every participant P_i who wants to time stamp at least one document in this round, submits to the time stamping authority a hash $y_{r,i}$ which is a hash of all the documents he wants to time

stamp in this round. The global stamp R_r for the whole round r is a cumulative hash of the time stamp R_{r-1} for round $r-1$ and of all the documents submitted to the TSA during the round r. After the end of the rth round a binary tree T_r is built. The leaves of T_r are labeled by different $y_{r,i}$. Each inner node k of T_r is recursively labeled by $h_k := h(h_{k\mathrm{L}}, h_{k\mathrm{R}})$, where $k\mathrm{L}$ and $k\mathrm{R}$ are correspondingly the left and the right child nodes of k (see Figure 3), and h is a collision-free hash function. The TSA has to store only the time stamps R_r for rounds. All the remaining information, required to verify whether a certain document was time stamped during a fixed round, is included into the individual time stamp of the document.

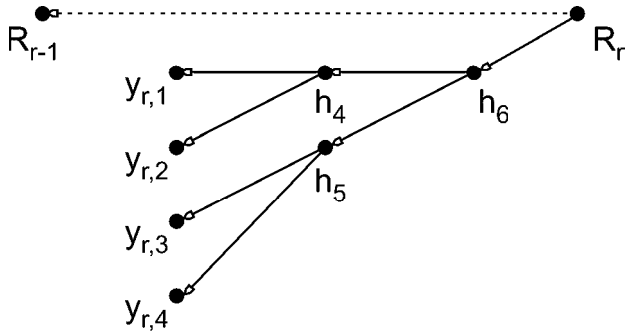

Fig. 3. An example of time stamp for round r by the Haber and Stornetta scheme.

The tree-like scheme is feasible but provides the relative temporal authentication (RTA) for the documents issued during the same round only if we unconditionally trust the TSA to maintain the order of time stamps in T_r. Therefore, this method either increases the need for trust or otherwise limits the maximum temporal duration of rounds to the insignificant units of time.

In the linear linking scheme, the challenger of a time stamp is satisfied by following the linked chain from the document in question to a time stamp certificate that the challenger considers trustworthy. If a trustworthy certificate occurs about every N documents then the verification process may require as many as N steps. By using the tree-like scheme, an exponential increase in the publicity obtained for each time stamping event is achieved, reducing the storage and the computation required in order to validate a given certificate, as the cost from this operation is reduced from N to $\log N$.

In reference [4] Buldas et al. proposed a practical linking scheme which allows the intra-round stamps to be kept ordered, regardless of the trustworthiness of the time stamping service provider. Basically, this scheme permits, given any two submitted hash values y_1 and y_2, building a hash chain leading from y_1 to y_2 or viceversa, even if both requests fall in the same round. Moreover, the scheme has a logarithmic upper bound to the length of the shortest verifying chain between any two time stamps. Mathematically speaking, a binary linking scheme can be defined as a directed countable graph which is connected, contains no cycles and where all the vertices have two outgoing edges. A general process for constructing an infinite family T_k of such graphs is as follows (See Figure 4):

- T_1 consists of a single vertex which is labeled with number 1. This vertex is both the

source and the *sink* of the graph T_1;

- Let T_k be already constructed and its sink be labeled by $2^k - 1$. The graph T_{k+1} consists of two copies of T_k, where the sink of the second copy is linked to the source of the first copy, and an additional vertex labeled by $2^{k+1} - 1$ which is linked to the source of the second copy. Labels of the second copy are increased by $2^k + 1$. The sink of T_{k+1} is equal to the sink of the first copy, the source of T_{k+1} is equal to the vertex labeled by $2^{k+1} - 1$. Thereafter, link all the vertices of the second copy which have less than two outgoing links, to the sources of the first copy. Note that there is now a double link from the sink of the second copy to the source of the first copy.

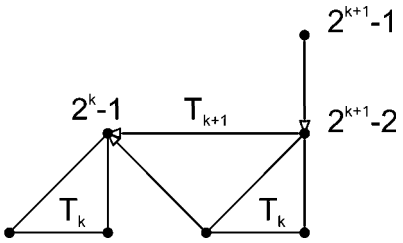

Fig. 4. Demonstration of the construction of a binary linking scheme.

The sequence (T_k) defines a binary linking scheme with the vertices labeled by numbers which contains each scheme T_k as its initial segment. After the construction of this binary linking scheme, links from the sources of any such initial segment to a special vertex labeled by 0 must be added (See Figure 5).

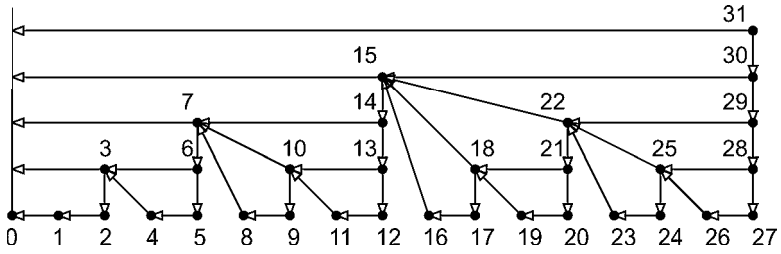

Fig. 5. Binary Linking Scheme.

As Figure 5 shows, given any two nodes labeled with i and j, with $i < j$, it is always possible to connect i to j with a one-way hash chain starting from j. A major drawback of the Buldas' approach is related to the algorithms for traversing the scheme, which are all rather irregular and complex.

4 System Architecture

Our objective was designing an architecture for providing time stamping services with emphasis on ubiquity and interoperability. Ubiquity is a key requirement due to the ever increasing number of "on the go" users. Interoperability is also a crucial requirement since the delivery of time stamping services involves a potentially large community of users, who typically rely on heterogeneous technologies. In order to achieve these goals, we adopted a multi-tier architecture, whose characteristics are described in the following. This facilitated a development approach exploiting clean separation of responsibilities, and made the proposed solution flexible, since the architecture relies on modular components consisting of re-configurable software modules. The innermost tier of the architecture, i.e. the back-end, implements a complex infrastructure which is in charge of the construction of rounds and time stamp chains, of the verification process, and of the publication. Implementation details and schemes are hidden to the external world. The middle-tier of the architecture addresses issues related to the heterogeneity of users and guarantees ubiquity and interoperability (as an example, it makes system services available to applications running on devices with limited computing power).

The overall organization of the system is depicted in Figure 6.

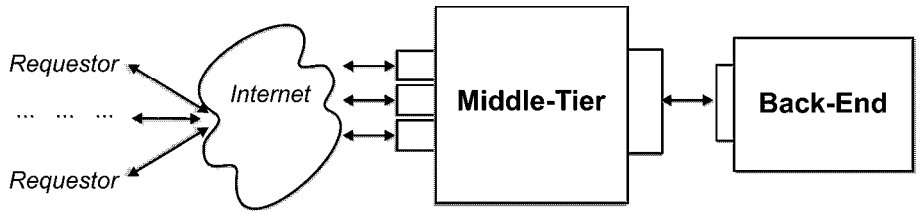

Fig. 6. Overall system architecture.

As shown in the figure, the clients, the middle-tier components, and the back-end servers are located in the first, second, and third tier of a multi-tier architecture, respectively.

The rest of this section is organized as follows: Subsection 4.1 thoroughly explains the back-end structure; Subsection 4.2 provides details on the middle-tier.

4.1 The Back-End

The back-end provides time stamping functions, thus satisfying the application functional requirements. The overall structure of the back-end server is depicted in Figure 7.

Basically, the functions the back-end time stamping system provides are the following:

- time stamp creation, i.e. creating a time stamp in response to a time stamp request;
- time stamp verification, i.e. verifying the validity of an existing time stamp;
- time stamp publication, i.e. making global hash values widely available for auditing and independent verification purposes.

As far as the time stamping scheme is concerned, we chose the linear linking scheme divided into rounds since it provides the best compromise in terms of security and performance. In fact, as it was pointed out in section 3, the actual implementation of the simple linear linking

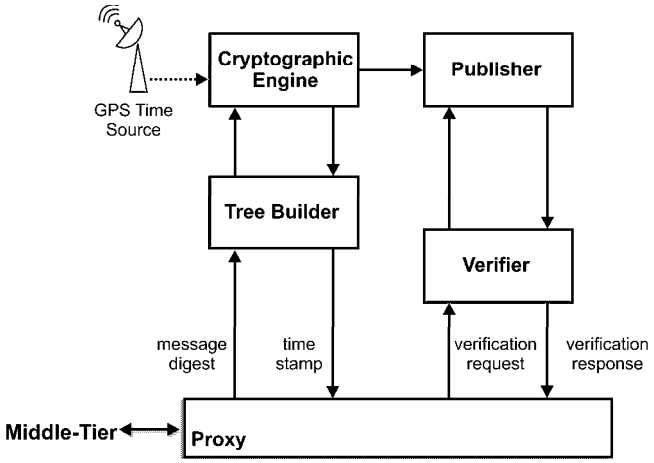

Fig. 7. The organization of the back-end.

scheme shows impractical requirements, mainly for the number of steps involved in verifying the one-way relationship between time stamps. The structure of the single round of the time stamping process is basically a tree in our implementation. The tree-like structure does not provide relative temporal order between stamps falling in the same round, while allowing for an easy and efficient implementation. As discussed in section 3, binary linking schemes are hard to build and to traverse, and they thus lead to a complex and inefficient implementation. So, we preferred to use a simple round structure, which is far more efficient.

In the following we briefly describe each of the back-end components.

The *Proxy component* is in charge of receiving the messages delivered by the middle-tier in a common format which does not depend on the particular implementation of the time stamping servers. The proxy builds messages to be exchanged with the underlying back-end server in a suitable format.

The *Tree Builder* component accepts a sequence of input time stamping requests and forms one output time stamping request to be submitted to the time stamping server for the linking process. The purpose of building tree structures is improving the system performance by reducing the server load as the number of requests sent directly to the server is decreased. This aggregation process is considerably less resource-consuming than linking and signing the time stamps. It allows to reduce the workload (since this does not scale any more with the number of time stamping requests) of the time stamping server, at the price of a moderate increase in the size of individual time stamps. The potential loss of precision due to the introduction of rounds can be made negligible by setting the length of the round to the time resolution needed by the target applications. Note that the Tree Builder component does not introduce any security risks. Hence, it is not necessary to authenticate it separately. The Tree Builder basically uses the Haber and Stornetta's tree-like scheme based on Merkle's authentication trees [18, 19]. Each tree corresponds to a round of the time stamping process and includes time stamping requests falling in a time window whose length T_w is taken as

a system parameter. Basically, Merkle's authentication tree is a method of providing short proofs (logarithmic in the number of inputs) that a bitstring x belongs to the set of bitstrings $\{x_1, x_2, \ldots, x_n\}$. During a round a binary tree is constructed as follows (see Figure 8): The leaves are labeled with message digests extracted from the time stamping requests obtained during the round, every non-leaf is labeled with a message digest computed using a hash function h over a concatenation of the labels of its children.

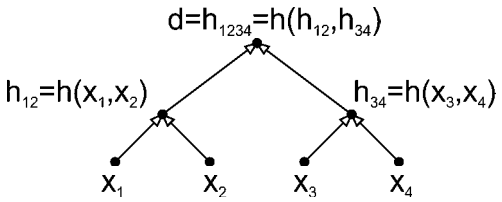

Fig. 8. Merkle's authentication tree.

Hence, the label of the root of the tree depends on all the leaves, i.e. aggregated bitstrings. For every leaf it is possible to prove this dependence by exhibiting some more vertices of the tree; the minimal collection of such extra nodes is called an *authentication path.*

Figure 9 shows an example of a Merkle's authentication tree where in order to prove that the root value d is dependent on the input x_1 it is enough to add vertices x_2 and h_{34} and compute $h_{12} = h(x_1, x_2)$ and $d = h_{1234} = h(h_{12}, x_{34})$.

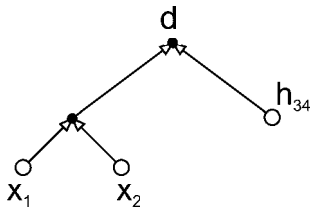

Fig. 9. Authentication path for bitstring x_1.

The *Cryptographic Engine* performs two distinct actions:

- It freezes the time stamping request sequence by linking the consecutive rounds based on a linear scheme. The direct dependencies are created by computing message digests using hash functions. This process is shown in Figure 10, where d_{13}, d_{14}, d_{15}, and d_{16} are the elements of the linear chain.

- It adds the time value to the time stamp to be formed. The absolute time is to be thought of as an additional piece of information provided by the system. It allows users to gain fine-grain information on the time period comprised between two published root values. In others word, it provides fine-grain absolute temporal authentication, in addition to the fine-grain relative temporal authentication provided by the linking scheme and the coarse-grain absolute temporal authentication provided by the mechanism of root value

publication. Obviously, this additional feature relies on the security of the system private key. However, erroneous or fraudulent behaviors would be immediately detected and proved by means of the linking process.

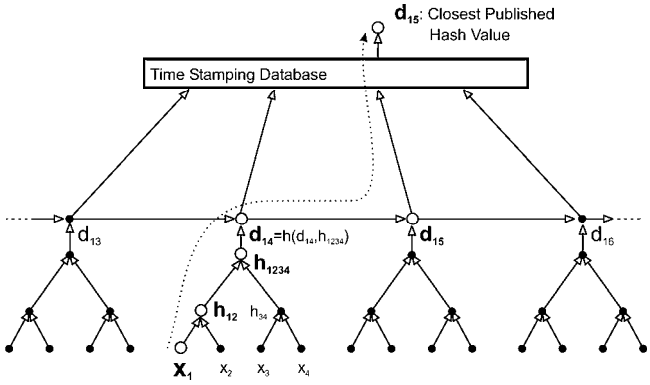

Fig. 10. Linking of round hash values and a complete hash chain.

Issued time stamps are also signed by the server assuring their authenticity and instant verifiability.

The *Publisher* component is used to make message digests daily available by some public and auditable media. Publishing is one of the central means ensuring the auditability of the service and retaining the long-term proof value of the time stamps, regardless of the trustworthiness of the time stamping service provider.

The *Verifier* component allows any external verifier to check the validity of an existing time stamp. An instant verification on time stamps is always possible without interacting with the time stamping server by simply checking the digital signature it provided with the stamp. However, as we explained above, this method requires an unconditionally trusted time stamping server and critically relies on the security of the provider's private key. On the other hand, the Verifier component provides the verifier with all the pieces of data needed to reconstruct a hash chain from the time stamp in question to the closest published hash value, so that the time stamping provider is not required to be trusted and the verification process is as trustworthy as the publishing media are. Figure 10 shows an example of an authentication path needed for such a verification process: the message to verify is x_1, while the chain leading to the closest published value consists of x_1, h_{12}, h_{1234}, d_{14}, d_{15}.

4.2 The Middle-Tier

The middle-tier is in charge of leveraging services provided by the back-end server, in order to satisfy interoperability requirements which arise from the heterogeneity of the clients.

Interactions between the middle-tier and the back-end take place via the Gateway (at the middle-tier side) and the Proxy (at the back-end side).

The middle-tier also decouples service implementation (i.e., the back-end) from its interface. Furthermore, the middle-tier is responsible for anonymity of requestors. Actually, an

important property of a time stamping service is that it should never associate clients to time stamp requests. The structure of the middle-tier is shown in Figure 11.

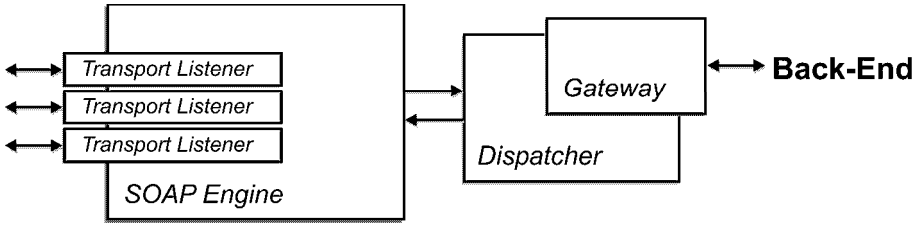

Fig. 11. The organization of the middle-tier.

On the client side the system appears as a Web service provider. The use of the Web services technology provided the following benefits:

- Promoted *interoperability* - The use of XML-based interfaces and protocols minimized the requirements for shared understanding between the service provider and a service requestor (which could be limited to collaboration and negotiation).
- Enabled *just-in-time integration* - Collaborations in Web services could be bound dynamically at runtime. Dynamic service discovery and invocation (publish, find, bind) and message-oriented collaboration yielded applications with looser coupling.
- Reduced complexity and improved flexibility and extensibility - These advantages were achieved via *encapsulation*. All components were seen as services. What was important was the type of behavior a service provided, not how it was implemented.

All these characteristics enabled heterogeneous client applications to easily interface with the time stamping server, and exploit its functions.

The service requestor can access the middle-tier via any transport protocol available over the Internet, provided that a protocol listener for the specific protocol has been implemented (in particular, we provided an HTTP listener and an FTP listener, which are two of the transports recommended in RFC3161 [1]). Requests and responses are exchanged through SOAP messages. The entry point for requestors' messages consists thus of a SOAP engine (see Figure 11) which coordinates the SOAP message flow through the subsequent components. The engine is also responsible for ensuring that the SOAP semantics are followed. Clients are not aware of the implementation details of the service provided by the back-end components. All they know is which services are available and what their interface is like.

The *Dispatcher* component is responsible for acting as a bridge between the SOAP processor and the functional components. It identifies the Java objects to which to delegate the execution of individual activities, and invokes the appropriate methods on such objects.

The *Gateway* component is in charge of data conversion from XML and SOAP data structures to Abstract Syntax Notation One (ASN.1) data structure (and viceversa). ASN.1 data structures are compliant with indications contained in [1].

All layers of the presented multi-tier architecture work in a pipelined fashion to achieve high throughput.

5 Implementation Details

Figure 12 shows the sequence diagram for the time stamp issuance process.

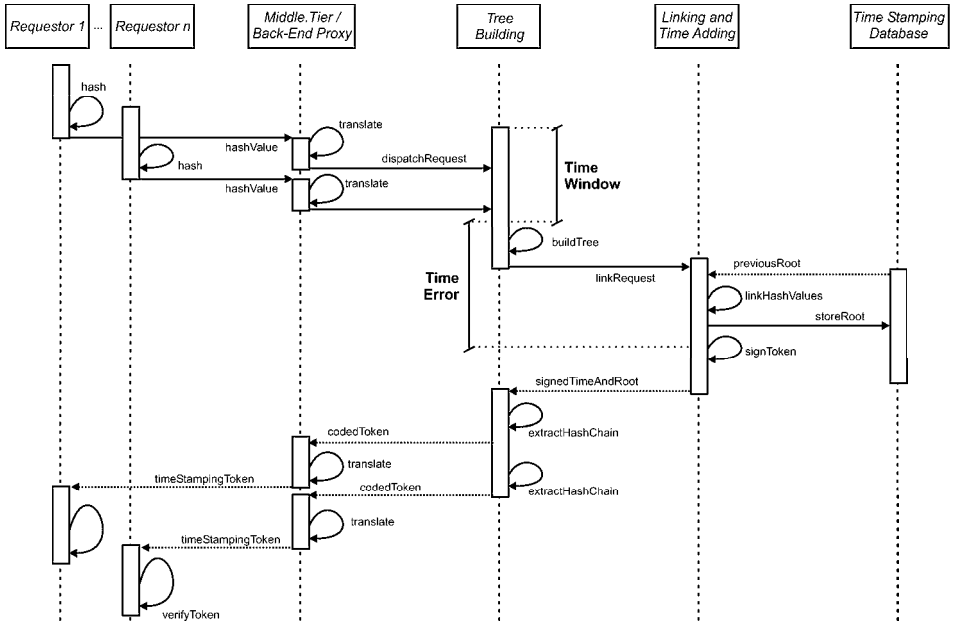

Fig. 12. The sequence diagram for the time stamp issuance process.

Clients who are willing to obtain a stamp for their documents have to produce the hash value of the document locally. The adopted hash function for our system is obtained as a concatenation of two well-known hash functions, i.e. SHA1 [20] and RIPEMD160 [10], that is:

$$h(x) = \text{SHA1}(x) \,\&\, \text{RIPEMD160}(x)$$

We chose a compound hash function in order to strengthen the security of our time stamping server. Actually, in the event that one of the two hash functions is broken, the system will not be compromised and old time stamps will remain valid as long as the other hash function is secure.

The Middle-Tier gathers clients' requests and dispatches them to the Back-end through the Back-End proxy, in order for the submitted hash values to be put into the tree-like scheme corresponding to the current time window. Once the tree is built, the root value is linked to the previous one, according to the linear linking scheme. The current time is also attached to the root value and the resulting message is signed with the private key of the system. This step relies on a trusted time source, namely three GPS time sources synchronized by means of a triangulation process and a network oriented protocol. The time source outputs universal coordinated time (UTC) with some error depending on its properties.

The requestors receive immediately the root value for the relevant time window tree, along with the hash chain covering the path between their hash value and the root value. Once the next publication will have taken place, they will get the full hash chain leading to the published root value.

Note that the certified time refers to the time window controlled by the tree building component. Submitted hash values are associated with the time window they fall in, regardless of the time of their submission. It takes some time to complete the tree building, to link the root value to the previous one, and to sign the resulting value along with the absolute time value. This time interval is referred to as time error in Figure 12. Obviously, the time error deteriorates the time resolution of the stamp issued. In our implementation, the error is guaranteed to be less than 1 second. The maximum value for the error is included in the time stamp together with the time value, and takes into account both the intrinsic time source error and round resolution error.

The width of a time window T_w, plus the time error (1 second at most), corresponds to the granularity of the time stamping token, as far as the absolute temporal authentication is concerned. The relative temporal authentication is provided as long as the two documents fall in two distinct rounds. Thus, the minimum temporal distance between two documents has to be greater than T_w seconds in order for their temporal order to be established. In section 6 we detail the procedure followed to establish the optimal value for T_w and the depth of the tree.

As far as protocols are concerned, to date there exists no consolidated standard describing all messages to be exchanged with a time stamping server based on a linking scheme. RFC3161 [1], the most widely accepted protocol for time stamping operations, refers to trusted time stamping authority schemes. Thus, it defines a simple protocol, which includes only request/response messages, while it does not provide any indications as to verification requests.

Figures 13 and 14 show a simplified representation of the RFC3161 structures for the time stamping request and response, respectively.

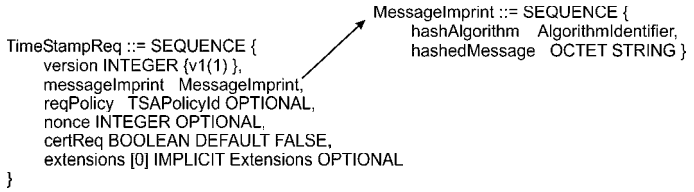

Fig. 13. A representation of the RFC3161 structure for the time stamping request.

The request structure is rather simple and contains only the document digest and a large random integer (called *nonce*) that will be put into the response by the time stamping server in order to allow the requestor to verify the timeliness of the response.

The response structure is much more complicated. It is based on a structure including the time stamp token and some information for the request status. The token is a general structure defined by the Cryptographic Message Syntax (CMS) (RFC2630 [13]), here specifically used for representing signed data and including digital certificates. According to RFC3161,

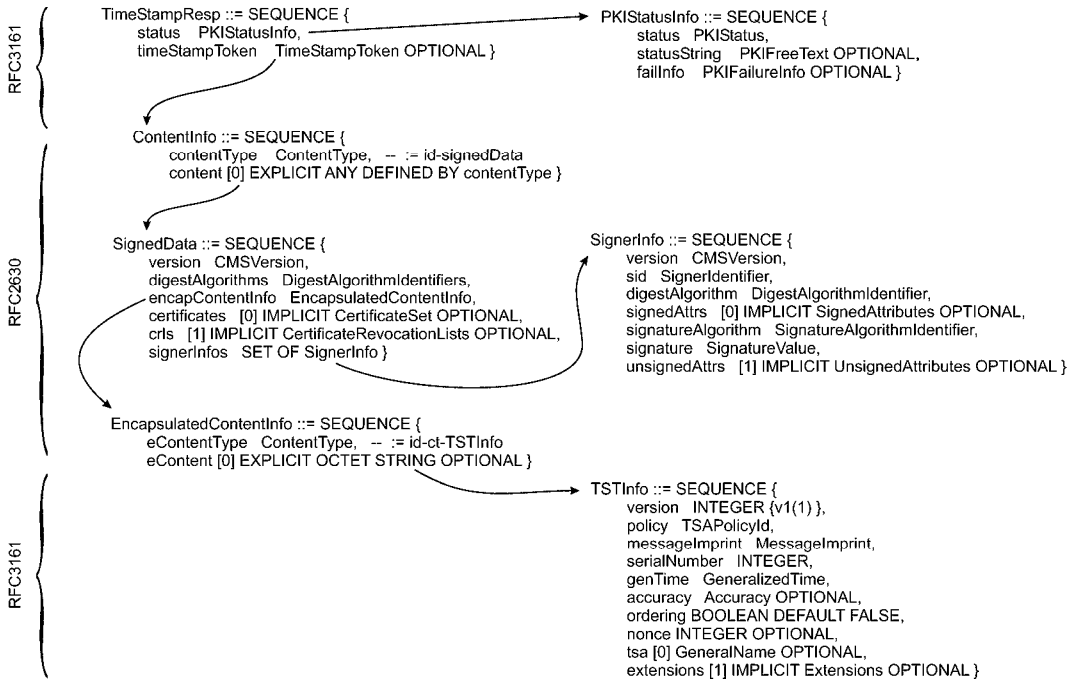

Fig. 14. A simplified representation of the RFC3161 structure for the time stamping response.

this signed data structure must encapsulate a time stamping structure that includes request specific data, such as the serial number, the time, the submitted digest, the nonce, etc. Note that this structure has a field aimed at containing extensions to the time stamping token.

For the time stamp issuance process, we followed the RFC3161 rules. In particular, we exploit the extension field within the token structure for returning the hash chain values linking the token to the relevant round root value. For the verification process, an ad hoc RFC3161-like protocol was employed.

Figure 15 shows the WSDL description of the SOAP messages involved in time the stamp issuance process, whose execution is shown in Section 6.

6 System Deployment and Experimental Tests

A prototype system has been built to actually make available the time stamping functions over the Internet and conduct some case-study experiments.

The system Back-End code runs on a IBM Server X-370 with four Pentium Xeon 900 MHz processors, and Linux kernel 2.4.12. The code is written in C++ language and is based on Crypto++ [9] library for cryptographic functions and on Snacc-based [27] efficient C++ routines and data structures to support Basic Encoding Rules (BER) for encoding and decoding of ASN.1 data structures. We used Java 2 Platform, Standard Edition (J2SE) [15] to provide connection to the service via Secure Sockets Layer (SSL) [26].

The Middle-Tier, relying on the XML, SOAP, WSDL and UDDI open standards, over an

```
<?xml version="1.0" encoding="UTF-8" ?>
- <wsdl:definitions targetNamespace="http://143.225.229.250:9080/TSS/wsdl/TS-service.wsdl" xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:apachesoap="http://xml.apache.org/xml-soap" xmlns:impl="http://143.225.229.250:9080/TSS/wsdl/TS-service.wsdl"
    xmlns:intf="http://143.225.229.250:9080/TSS/wsdl/TS-service.wsdl" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:tns1="http://services" xmlns:tns2="http://PckProjectService" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  - <wsdl:types>
    - <schema targetNamespace="http://services" xmlns="http://www.w3.org/2001/XMLSchema">
        <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
      - <complexType name="TSSResponse">
        - <sequence>
            <element name="poolRef" type="xsd:int" />
            <element name="timeToCheckBack" type="xsd:int" />
          </sequence>
        </complexType>
        <element name="TSSResponse" nillable="true" type="tns1:TSSResponse" />
      </schema>
    </wsdl:types>
  + <wsdl:message name="requestAdvancedMarkForHashAXRequest">
  + <wsdl:message name="requestAdvancedMarkForHashRequest">
  + <wsdl:message name="getSimpleMarkAXResponse">
  + <wsdl:message name="requestAdvancedMarkForStringResponse">
  + <wsdl:message name="requestAdvancedMarkForHashAXResponse">
  + <wsdl:message name="requestAdvancedMarkForStringRequest">
  + <wsdl:message name="retriveMarkResponse">
  + <wsdl:message name="getSimpleMarkForHashRequest">
  - <wsdl:message name="getSimpleMarkForStringResponse">
      <wsdl:part name="getSimpleMarkForStringReturn" type="xsd:base64Binary" />
    </wsdl:message>
  + <wsdl:message name="getSimpleMarkForHashResponse">
  + <wsdl:message name="retriveMarkRequest">
  + <wsdl:message name="getSimpleMarkForStringRequest">
  + <wsdl:message name="getSimpleMarkAXRequest">
  + <wsdl:message name="requestAdvancedMarkForHashResponse">
  + <wsdl:portType name="TS">
  + <wsdl:binding name="rpcrouterSoapBinding" type="impl:TS">
  - <wsdl:service name="TSService">
    - <wsdl:port binding="impl:rpcrouterSoapBinding" name="rpcrouter">
        <wsdlsoap:address location="http://143.225.225.250:9080/servlet/rpcrouter" />
      </wsdl:port>
    </wsdl:service>
  </wsdl:definitions>
```

Fig. 15. WSDL description of the SOAP messages involved in time stamp issuance.

Internet protocol backbone is implemented with the Application Server IBM WebSphere 4, running on Dell PowerEdge 1400SC with two 1400MHz Pentium III processors, running a Linux Red Hat kernel 2.4.18-3 with dual processor support.

On this testbed, we conducted a massive experimental campaign, for different values of key system parameters, and in particular of the time window T_w and of the tree depth d. This allowed us to identify the parameter configuration which delivered the best performance for a given time resolution. Since our objective was to achieve a time resolution of 10 seconds ($T_w = 10s$) we set d to 13. These values lead to a throughput of $2^{13} = 8192$ documents every 10 seconds, i.e. 2949120 documents per hour. Experimental tests showed that the system was able to achieve this throughput with a CPU utilization and a memory occupation which never exceeded 60 and 80 percent, respectively.

We also tested the functions of the time stamping server from a thin-client browser running on a java-enabled PDA. We used a Compaq IPAQ 3870 handheld device as the experimental client system. This is a challenging but nevertheless more and more likely scenario, due to the astonishing increase in the popularity of handheld devices.

The PDA runs the Linux Familiar Operating System 2.4.18. Note that the Java virtual machines (JVMs) available for PocketPc-based devices are compliant with J2ME/MIDP specifications; these JVMs lack standard XML APIs and support for the SSL protocol. Moreover, these devices are equipped with web browsers that are not capable of managing Java applets.

Hence third party J2ME/CLDC (Connected Limited Device Configuration) libraries have been used in order to handle XML, especially Web services-specific XML protocols. KXML and KSOAP libraries contain XML parsing and SOAP-based communication primitives for J2ME applications.

Figure 16 shows the execution of a client application written in java, which sends a time stamp request to the time stamping server.

```
usr
xlibs_4.1-5_arm.ipk
/mnt/hda # cd sintel
/mnt/hda/sintel # ls
certlogcli   certlogjars  sharedjars   tsxcli       tsxjars
/mnt/hda/sintel # cd tsxcli/
/mnt/hda/sintel/tsxcli # ls
MyFile.ts       TestTSX.class   TSProxy.class   services
MyFile.txt      TestTSX.java    TSProxy.java
/mnt/hda/sintel/tsxcli # java TestTSX S MyFile.txt MyFile.ts
Service end point : http://143.225.229.250:9080/TSS/servlet/rpcrouter
File read
Proxy instanced
Pool ref received
time to check back received
Waiting, for 60 sec.....................................................
.................

stamp received
Opening file for writing
Time stamp token saved
Done
/mnt/hda/sintel/tsxcli # uname -a
Linux familiar 2.4.18-rmk3 #1 Tue Oct 15 14:38:20 EDT 2002 armv41 unknown
/mnt/hda/sintel/tsxcli # _
```

Fig. 16. Execution of the case-study application.

The server returns an XML structure (called *TSSResponse*) which contains an ID num-

ber (*poolRef*) associated with the particular time stamping request, and an integer value (*timeToCheckBack*) representing the (minimum) time the client has to wait. When *timeToCheckBack* seconds have elapsed, the client gets the time stamp identified by *poolRef* (see figure 12). The PDA was able to receive and store a time stamped token of a file containing the data of an electronic transaction. The experiment was successful, in that this goal was achieved in a seamless way. We emphasize that this would not have been the case if traditional technologies had been used.

7 Lessons Learned

In this section we present some of the problems we had to cope with during the actual implementation of the system, and the lessons learned from these difficulties.

A major problem concerned the protocol used for codifying the time stamping token. We followed – whenever possible – the recommendations of RFC3161. However, RFC3161 refers to the information exchange between the clients and the server of a time stamping system which is based on a simple scheme. As already mentioned in subsection 3.2, simple schemes allow a malicious TSA to fraudulently alter the time parameter of a certain time stamp without being caught. As a consequence, simple schemes are inadequate for most real-world scenarios. We thus had to come up with some workarounds, in order to develop a time stamping service which was based on a linking schemes while still being compliant to RFC3161. In particular, in linking schemes an additional piece of information (as compared to simple schemes) is returned to the clients, i.e. the hash chain from the current hash value to the root of the round. Such an additional piece of information is retrieved immediately, along with the date/time information. Additionally, in the linking scheme the client requests a hash chain extension leading to the closest published value. This request must take place at a later time, and is always initiated by the client. It is impossible to accomplish all these operations while complying with RFC3161, simply because this standard – being based on simple schemes – assumes that no such additional data item be returned, nor it does specify any interaction between the server and the client after the time stamping token is delivered to the client. To allow the retrieval of the round hash chain, we exploited the extension field included in the RFC3161. For returning the hash chain to the published value we resorted to a proprietary protocol which closely follows the RFC3161 (a further field is needed to indicate the operation type). From this experience, we can thus conclude that the RFC3161 is an incomplete standard, as far as time stamping services based on linking schemes are concerned. Given the ever increasing interest that linking schemes are gaining in open networked environments, it appears that there is an urging need for a more mature protocol.

Another key issue we had to address was related to the compliance of the WSDL code for the service – which we generated by means of Commercial-off-the-Shelf development tools – to the recommendations of the World Wide Web consortium for Web Services interoperability. We found that currently available development kits exhibit several interoperability problems, especially for aspects related to data type serialization and name spaces. In the following, we give an example of an actual problem experienced during the development of the middle-tier.

We used the development kit shipped with WebSphere 4 to generate the WSDL code for the time stamping service and verified the validity of such code using xmlspy [30]. Figure 17 shows the output of the verification process.

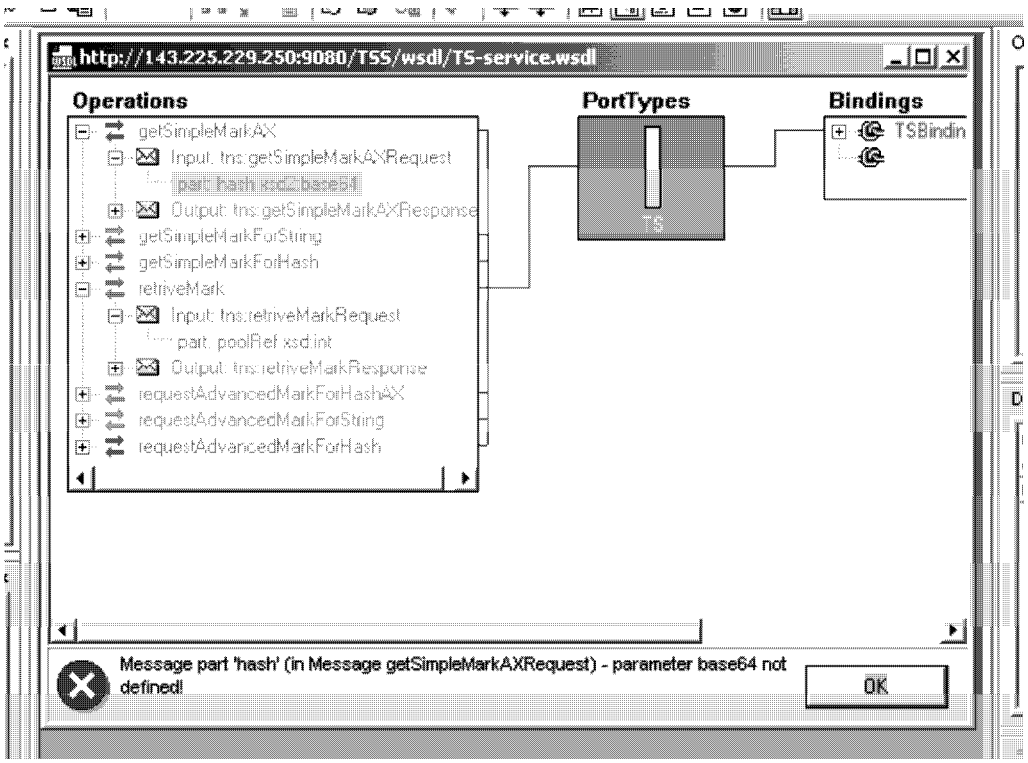

Fig. 17. An xmlspy screenshot showing the flawed piece of WSDL code.

The xmlspy verifier detected problems with several datatypes and namespace definitions. Some of the problems were fixed by re-generating the code using a more recent version of the WSDL Toolkit (which we downloaded from the IBM alphaworks site). In some other cases, we had to manually hack the WSDL code. From this experience we can conclude that there is still a long way to go, as far as seamless interoperability of Web Services technologies is concerned.

8 Related Work

In this section, we provide an overview of relevant projects/products for implementing time stamping services. We comment individual solutions and contrast them to our work.

Among commercial solutions, the most relevant is the one proposed by Surety [29], the American company founded in 1994 by Haber and Stornetta, the pioneers of time stamping. The vendor does not provide details about implementation, performance, and development and deployment costs of this solution. According to the (scarce) technical documents available, Surety builds a time stamping infrastructure based on a linking scheme. The central

system manages the construction of the time stamp chain, the stamp repository, and even the verification process. Our solution is based on a linking scheme, too. As such, both solutions do not require that the TSA be unconditionally trusted. This is a key feature for a time stamping service which is to be deployed in a real-world scenario, as discussed in section 1. As to the throughput, both systems are able to satisfy the requirements of a wide scale environment. However, users of the service provided by Surety are tightly tied to a proprietary system protocol, in particular in the verification phase. Our solution relies instead on open and widely adopted standard protocols, namely the Web Services technologies. As such, our solution is ideally suited for deployment in an Internet-based scenario, where challenging interoperability issues arise from the fact that the service must be made available to a potentially large community of users, which typically rely on a whole variety of heterogeneous hardware/software platforms.

Another interesting commercial service is provided by Cybernetica [5]. The system infrastructure is based on a binary linking scheme. To use the service, a proprietary protocol and ASN.1 data structures are defined and made publicly available. Software programs which comply with this set of rules can interact with the system have. Our solution is based on a linking scheme, which may not be as efficient as a binary linking scheme. We are not able to provide a quantitative estimate of this aspect, since the Cybernetica system has been set up only recently and data on its ability to cope with the scalability requirements of real-world scenarios is not available at the time of this writing. As to interoperability, our solution appears to be better, since – as already mentioned – it relies on the Web Services technology, which is based on open and widely adopted standard protocols.

Among open-source implementations, the most active project is OpenTSA [21], whose main aim is to develop an RFC 3161 [1] compliant, stable, and free time stamping authority client and server application. The time stamp request creation, response generation and response verification functionality is implemented as an extension to the latest stable version of OpenSSL [22]. The major drawback of this project is that it is based on a simple scheme. As discussed in subsection 3.2, simple schemes allow a malicious TSA to fraudulently alter the time parameter of a certain time stamp without being caught. As a consequence, the solution proposed by OpenTSA is inadequate for most real-world scenarios.

Another example of open-source, free digital time stamping service is the PGP Digital Timestamping Service [23]. PGP Digital Timestamping provides an automatic service for corroborating the date at which a user signs a document with PGP. Every signature made by the system has a unique serial number, which automatically increments by one every time a document is signed. The system also stamps summaries of its own signatures from the previous day. This system does not comply with any existing time stamping protocol. Also, PGP is known to have major scalability problems. The PGP Digital Timestamping Service is thus not suitable for an Internet-based environment.

9 Conclusions

This paper described the results of a year long research activity conducted cooperatively by an academic and an industrial party, aimed at designing and implementing a practical solution for providing time stamping services and exposing them to the Internet. The paper provides practitioners of computer security with field experience and lessons learned about best practices for wide scale deployment of time stamping services using Commercial Off-The-Shelf (COTS) software products.

The paper provided a comprehensive overview of the main state-of-the-art algorithms for time stamping applications and thoroughly discussed pros and cons of each technique. In particular, it highlighted the crucial issues raised by the practical implementation of time stamping algorithms.

A practical architecture for time stamping services was then introduced, which provided both relative temporal authentication, based on a linear linking scheme, and absolute temporal authentication, based on publishing mechanisms as well as on a trusted time source. The emerging Web services technology was used to expose the time stamping services to the Internet. This choice allowed us to cope with the challenging ubiquity and interoperability requirements, which arise from the large number of users and heterogeneous technologies involved in emerging Internet-based scenarios.

The problems we experienced with the actual implementation of the system allowed us to emphasize some of the key open issues related to the provision of time stamping services in open networked environments, and in particular: i) the lack of a convenient agreed-upon standard for time stamping services which are based on linking schemes, and ii) some yet unresolved interoperability problems of Commercial-Off-The-Sheld (COTS) products for the development of Web-Service based applications.

Experimental tests were conducted on a distributed prototype – which included a Compaq IPAQ 3870 handheld device as the client system. Such tests demonstrated the effectiveness of the proposed solution even for wired/wireless internetworked computing infrastructures.

Acknowledgements

Authors are grateful to Annalisa Caso, Sonia Del Vacchio, Francesco Festa, and Gerardo Maiorano for the numerous fruitful technical discussions, and for developing the bulk of the code. This work has been carried out under the financial support of the University of Naples Federico II, Sintel SPA, the Consorzio Interuniversitario Nazionale per l'Informatica (CINI), the National Research Council (CNR), the Ministero dell'Istruzione, dell'Universita' e della Ricerca (MIUR), and the Regione Campania, within the framework of projects: SP1 Sicurezza dei documenti elettronici, Oltre la Firma Digitale(OFD), Gestione in sicurezza dei flussi documentali associati ad applicazioni di commercio elettronico, FIRB - Middleware for advanced services over large-scale wired-wireless distributed systems (WEB-MINDS), and Telemedicina.

References

1. C. Adams, P. Cain, D. Pinkas, R. Zuccherato (2001), *Internet X.509 Public Key Infrastructure Time Stamp Protocol (TSP)*, available at http://www.ietf.org/rfc/rfc3161.txt

2. D. Bayer, S. Haber, W.S. Stornetta (1993), *Improving the efficiency and reliability of digital timestamping*, in Proceedings of Methods in Communication, Security, and Computer Science, Springer-Verlag, pp. 329-334.
3. J. Benaloh, M. de Mare (1991), *Efficient Broadcast time stamping*, Technical Report 1, Clarkson University Department of Mathematics and Computer Science.
4. A. Buldas, P. Laud, H. Lipmaa, J. Villemson (1998), *Time Stamping with Binary Linking Schemes*, in Proceedings of Advances in Cryptology CRYPTO '98.
5. Cybernetica web site - http://www.timestamp.cyber.ee/
6. A. Cilardo, A. Mazzeo, L. Romano, G.P. Saggese, G. Cattaneo (2003), *Providing Interoperable Time Stamping Services*, in Proceedings of International Conference on Advances in Infrastructure for Electronic Business, Education, Science, Medicine, and Mobile Technologies on the Internet, L'Aquila, Italy.
7. A. Cilardo, A. Mazzeo, L. Romano, G.P. Saggese, G. Cattaneo (2003), *Using Web Services Technology for Inter-Enterprise Integration of Digital Time Stamping*, in Proc. of Workshop on Reliable and Secure Middleware (WRSM 2003), LNCS 2889, Springer-Verlag. pp. 960-974.
8. D. Cotroneo, C. di Flora, A. Mazzeo, L. Romano, S. Russo, G. P. Saggese, *Providing Digital Time Stamping Services to Mobile Devices*, to appear in: Proc. of the 9-th IEEE international Workshop on Object-oriented Real-time Dependable Systems - Fall (WORDS 2003F), IEEE CS Press, pp. 94-100.
9. Crypto++ Project: http://www.eskimo.com/ weidai/cryptlib.html
10. H. Dobbertin, A. Bosselaers, B. Preneel (1996), *RIPEMD-160, a strengthened version of RIPEMD*, in Fast Software Encryption, LNCS 1039, D. Gollmann, Ed., Springer-Verlag, pp. 71-82.
11. S. Haber, W.S. Stornetta (1991), *How to timestamp a digital document*, in Journal of Cryptology, Vol. 3, pp. 99-111.
12. S. Haber, W.S. Stornetta (1997), *Secure Names for Bit-Strings*, in Proceedings of the 4th ACM Conference on Computer and Communications Security, pp. 28-35.
13. R. Housley (1999), *Cryptographic Message Syntax (CMS)*, available at http://www.ietf.org/rfc/rfc2630.txt
14. International Organization for Standardization and International Electrotechnical Commission (2002), *ISO/IEC Standard 18014: Information technology - Security techniques -Time stamping services*
15. Java 2 Platform, Standard Edition: http://java.sun.com/j2se
16. H. Kreger (2001), *Web Services Conceptual Architecture*, IBM Software Group, available at http://www-3.ibm.com/software/solutions/webservices/pdf/WSCA.pdf
17. A. J. Menezes, P. C. Van Oorschot, S. A. Vanstone (1996), *Handbook of Applied Cryptography* , CRC Press.
18. R. C. Merkle (1980), *Protocols for public key cryptosystems*, in Proceedings of the 1980 IEEE symposium on security and privacy, pp. 122134, IEEE Computer Society Press.
19. R. C. Merkle (1989), *A certified digital signature*, in Proceedings of Advances in Cryptology Crypto89, vol. 435, pp. 218238, Springer-Verlag.
20. National Institute of Standards and Technology (1995), *Secure Hash Standard, FIPS PUB 180-1*, Federal Information Processing Standards Publication (available on-line at http://www.itl.nist.gov/fipspubs/fips180-1.htm.)
21. Open TSA project web site - http://www.opentsa.org/
22. Open SSL project web site - http://www.openssl.org/
23. PGP Digital Timestamping Service web site - http://www.itconsult.co.uk/stamper.htm
24. R. Rivest (1992), *The MD5 Message-Digest Algorithm*, RFC 1321, MIT LCS & RSA Data Security Inc.
25. B. Schneier (1995), *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, John Wiley & Sons.
26. Secure Sockets Layer (SSL): http://wp.netscape.com/eng/ssl3/draft302.txt
27. Snacc website: http://www.fokus.gmd.de/ovma/freeware/snacc/entry.html

28. H.M. Sneed (1996), *Encapsulating legacy software for use in client/server systems*, in Proceedings of Working Conference on Reverse Engineering, pp. 104–119.
29. Surety web site - http://www.surety.com
30. xmlspy web page - http://www.xmlspy.com/