Authenticating Distributed Data using Web Services and XML Signatures*

Daniel J. Polivy, Roberto Tamassia

Department of Computer Science
Brown University
Providence, RI 02912-1910

{dpolivy, rt}@cs.brown.edu

ABSTRACT

As the need for digital data becomes more ubiquitous, so does the need to provide efficient mechanisms for distributing and verifying the authenticity of that data. We present an architecture for authenticating responses to queries from untrusted mirrors of authenticated dictionaries using Web Services and XML Signatures. We also describe an implementation of our scheme for the Secure Transaction Management System.

August 9, 2002

* Work supported in part by the Dynamic Coalitions Program of the Defense Advanced Research Projects Agency under grant F30602–00–2–0509.

1 INTRODUCTION

As the need for digital data becomes more ubiquitous, so does the need to provide efficient mechanisms for distributing and verifying the authenticity of that data. Take, for example, an Internet portal site that aggregates information from many different sources—such as weather, news headlines, and stock prices—and provides that information to its users. Currently, it is up to the user's discretion whether to trust the information provided by the portal; if they trust the portal, then they trust the data it provides. However, it is possible to individually verify the authenticity of each piece of data, without implicitly trusting the portal, using a distributed authenticated dictionary.

An *authenticated dictionary* is a data structure that supports authenticated membership queries; that is, in addition to returning the answer to a query, it also returns a cryptographic proof of that answer. A distributed authenticated dictionary architecture separates the data source, the host that maintains the data structure, from the responder, the host that answers queries on the data structure on behalf of the source. In this paper, we deal primarily with systems in which the client trusts only the source and not the responder, such as the Secure Transaction Management System (STMS) [23]. An overview of the basic message exchanges in a distributed authenticated dictionary is shown in Figure 1. At the end of a specified time quantum, the source reliably pushes incremental updates, as well as a digitally signed and time stamped fingerprint of the data structure, called the basis, to all responders. When a responder receives a query from a user, it returns the answer, a proof of that answer, and a copy of the signed basis from the source. Using these last two pieces of information, the user is then capable of verifying the authenticity of the response locally while trusting only the basis, which has been signed by the source.

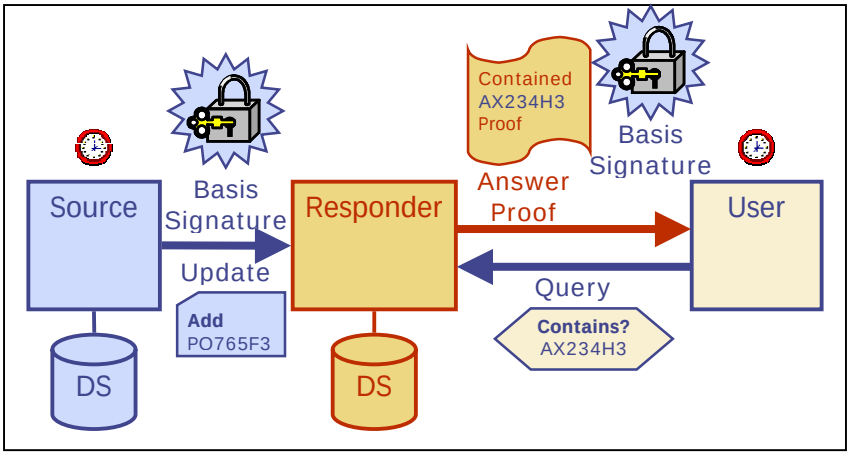

Figure 1 An overview of the protocol for authenticated distributed data systems. The source pushes updates containing a signed basis to the responder. The responder then answers user queries with a proof of the answer, and a copy of the signed basis from the source.

There are many applications where this type of authenticated data would be required. A user contacting a mirror site would need to cryptographically validate the information as being genuine; that is, as being the same information had the response come directly from the source. For example, a financial speculator that receives NASDAQ stock quotes from the *Yahoo! Finance* Web site would be well advised to obtain a proof of the authenticity of the data before making a large trade based on that information. Another application would be to guarantee the end-to-end integrity of dynamically generated web content from geographically distributed mirror sites. For all applications, however, we desire solutions with concise responses and low verification overhead.

The main goals of this project are to obtain greater interoperability with client platforms, reduce the amount of "reinventing the wheel" (i.e., developing proprietary mechanisms when

standards are available) and research new ways of simplifying proof verification. To achieve these goals, we contribute a design and implementation of a Web Service interface for STMS using SOAP [5] and a design and implementation of a methodology for verifying STMS proofs via a standard XML Signature [2]. While both of these techniques have been deployed within the STMS system, the XML Signature approach to proof verification is a unique contribution that is applicable to the greater domain of distributed data authentication.

The main drawback of previous versions of STMS was the fact that clients had to access it using a bulky, proprietary interface. Not only did this require special knowledge on the part of the application developer, but it also required a separate toolkit to be provided for each target operating system and programming language. Additionally, such toolkits used proprietary communications protocols, canonicalizers, and serializers. By building a Web Services interface to STMS, we are better able to utilize existing standards to achieve greater levels of portability, interoperability, and ease of use for the client. This new interface also reduces the size of the client toolkit by delegating the communication, canonicalization, and serialization code to existing implementations. The contribution of a Web Services interface provides a framework for performing remote procedure calls using standard interoperable technologies such as XML [3], SOAP [5] and HTTP [11]. Previous STMS implementations used only Java RMI as the communications protocol; replacing this with a SOAP-based protocol allows compatibility with many different programming languages, as well as better scalability properties when deployed across the Internet.

In the untrusted responder model for distributed authenticated dictionaries, the verification of a response requires two distinct steps. The first step involves verifying a signed fingerprint of the source, and the second step involves computing a proof and comparing the result to the signed fingerprint. The verification of the fingerprint is a standard digital signature verification (using a custom byte canonicalization), while the verification of the proof is dependent upon the implementation of the data structure, but usually involves some sort of hash computation. By requiring two separate validations to ensure authenticity, the untrusted nature of the responder makes itself known.

The primary contribution of this paper is a methodology for making the responder "invisible" to the client when performing proof verification via XML Signatures [2]. The XML Signature standard provides the rules and syntax for encoding standard digital signatures of data in XML. Through the use of a custom transform, the two-step verification process is reduced to the simple verification of a standard XML Signature, while still maintaining the original security guarantees. Before data is signed or verified in the XML Signature model, specified transforms are applied in series to produce the final, signed data. By implementing the proof authentication as a type of transform, this step is integrated into the verification of the basis. Thus, clients can now verify the authenticity of data using standard cryptographic APIs, without regard to how the data was distributed. When combined with the Web Services interface, the result is a fully interoperable, standards-based architecture for distributed data authentication.

We also present an implementation of this architecture in Java and .NET, as well as experimental results from this implementation.

2 RELATED WORK

Previous work is mostly related to the authentication of membership queries, where a query is of the type "Is element *e* in set *S*?" *Authenticated dictionaries* are data structures that provide authenticated answers to such queries, and were initially motivated by research into the certificate revocation problem.

The *hash tree* scheme introduced by Merkle [16][17] can be used to implement a static authenticated dictionary that supports query operations in logarithmic time. Kocher [15] also advocates a static hash tree approach for realizing an authenticated dictionary, although with a simplified verification process.

Using techniques from incremental cryptography, Naor and Nissim [18] develop dynamic hash trees that additionally support insertion and deletion of elements. In their scheme, the data structure is implemented by 2–3 trees.

Goodrich and Tamassia [12] present an authenticated dictionary data structure based on skip lists [20]. They introduce the notion of commutative hashing and show how to embed in the nodes of a skip list a computational DAG (directed acyclic graph) of cryptographic computations based on commutative hashing. They match the asymptotic performance of the Naor-Nissim approach [18], while simplifying the details of an actual implementation. Goodrich, Tamassia and Schwerin [13] present the software architecture and implementation of an authenticated dictionary based on the above approach. An implementation of a one-way accumulator to realize an efficient and dynamic authenticated dictionary is presented in [14]. An efficient data structure for *persistent authenticated dictionaries*, where the user can issue historical queries of the type, "was element *e* in set *S* at time *t*?" is introduced in [1]. An approach to signing XML documents which allows untrusted servers to answer certain types of path queries and selection queries over XML documents is described in [19].

Previous work has also been completed in the area of authentication through digital signatures. A *digital signature* is a piece of information that guarantees the sender of a message is indeed who he or she claims to be (e.g., see [21]). The Internet standard for X.509 Certificates [24] provides a standard method for encoding and verifying signed statements containing information about an entity, including its public key. Certificates are useful as a way for securely distributing public keys for digital signature verification.

The recently developed XML Key Management Specification (XKMS) [10] is a protocol specification for distributing and registering public keys that can be used in conjunction with XML Signature [2]. XKMS is mainly concerned with the problem of key distribution and location in a distributed online environment. The XML Signature specification [2] is published by the W3C as a set of rules and XML syntax for encoding digital signatures.

3 PRELIMINARIES

In this section, we review concepts and terminology that are used in the rest of the paper.

3.1 STMS

The Secure Transaction Management System (STMS) is a distributed system for data authentication [13][23]. At its core is an authenticated data structure maintained by a trusted source. This data structure is replicated by untrusted responders, which answer queries about the data structure on behalf of the source and provide a cryptographic proof of the answer. This approach has many advantages, including centralized trust (the users have to trust only the source), distributed service (the responders are geographically distributed closer to clients), low deployment cost (the responders do not require secure installations), and resiliency to denial-of-service attacks (the source does not answer queries itself). Applications of this technology could exist in many domains, including content delivery networks, telecommunications, financial services, and health care.

A schematic illustration of the STMS architecture is shown in Figure 1. In STMS, the Basis is a piece of data analogous to a fingerprint of the data structure. It is updated once per time quantum, digitally signed by the source (the to-be-signed basis, complete with timestamps and signing information, is called a RichBasis), and reliably pushed to all the responders. An AuthenticResponse is an object returned as the result of a query on an authenticated data structure; in addition to the answer, it contains a proof that can be verified against the basis to guarantee integrity of the answer. The basis can be informally viewed as a hash of the entire data structure while the proof can be viewed as the hash of the data structure minus the query key. A client verifies that the answer is authentic by hashing the query key and "combining" it with the proof; the result should be equivalent to the signed hash of the entire data structure.

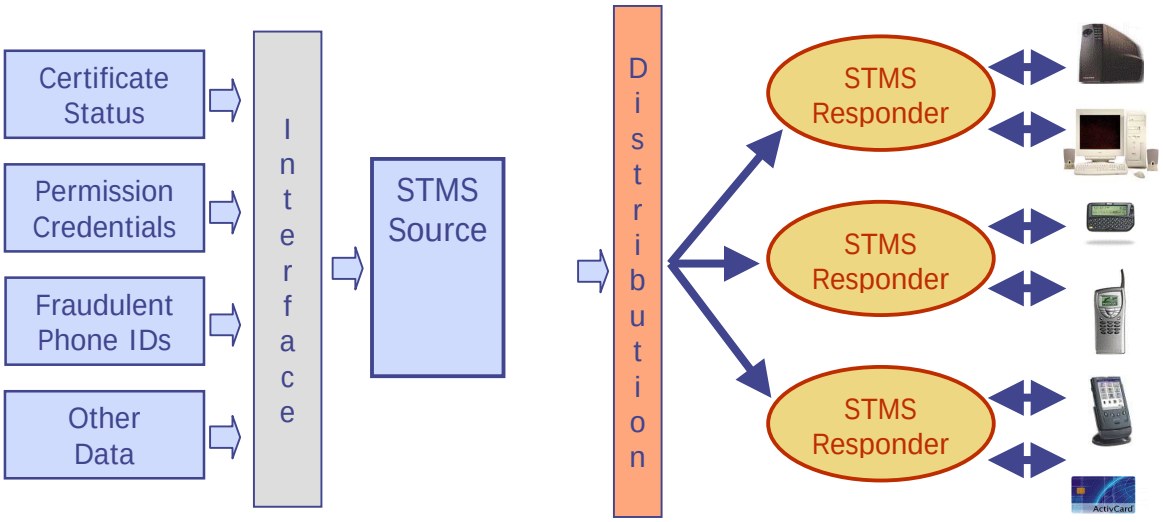

Figure 2 Overview of the STMS architecture, showing possible deployment models.

3.2 Authenticated Skip Lists

An Authenticated Skip List (ASL) [12] is the particular implementation of an authenticated dictionary used in STMS. It supports logarithmic query and update times while using linear space. A hierarchical commutative hashing over a directed acyclic graph provides authentication; the hash of the start node is a digest for the entire data structure, and a proof is the sequence of hashes of the neighbors of the nodes on the search path. The definition of a Commutative Hash function as found in [12] is a cryptographic hash function h such that $h(x, y) = h(y,x)$, where x and y are integers with the same bit length. The Authenticated Skip List is the primary implementation dealt with in this research.

3.3 Web Services and SOAP Overview

The term "Web Services" is generally used to describe a collection of protocols and standards that are used to facilitate interoperability between applications. One of the major factors for their success is the fact that they are built upon existing Internet standards such as XML [3] and HTTP [11]. This allows for high levels of scalability and interoperability that previous distributed architectures could not provide. One of the main enabling technologies for performing remote procedure calls (RPCs) using Web Services is SOAP, the Simple Object Access Protocol. SOAP is an XML-based protocol for packaging messages and facilitating RPC-style communication between clients and servers (and is capable of performing many other tasks as well). For a full description of SOAP, see [5].

SOAP's use in STMS is to provide a protocol- and platform-agnostic format for encoding objects in RPC-style communication. While SOAP defines a set of built-in types that roughly correspond with those defined in the XML Schema specification, it is also capable of handling more complex custom types composed of combinations of primitive and other complex types. As is explained later, this feature is exploited in order to transfer STMS objects from responder to client.

3.4 XML Signature Overview

The XML Signature standard [2] provides a set of rules and XML syntax for encoding, computing, and verifying digital signatures of arbitrary (but often XML) data. In addition to providing authentication, data integrity, and support for non-repudiation to the data that they sign, XML Signature has been designed to take advantage of the Internet and XML. A fundamental feature of XML Signature is the ability to sign only specific portions of the XML tree rather than the complete document; this becomes useful when documents aggregate many pieces of information from different sources, each with their own proof of authenticity.

Validation of a signature requires that the signed data be accessible by some sort of reference. This reference can be a URI, a part of the same resource as the signature, embedded within the signature, or embed the signature within itself. The basic signature algorithm is as follows: any number of Transforms is applied to the data to be signed. The output of the final transform is then digested, and the resulting value is placed in an element (DigestValue) along with other information. This other information (the SignedInfo element), is then canonicalized using a standard XML Canonicalization algorithm, digested, and cryptographically signed. The result is stored in the SignatureValue element. Verification follows a similar algorithm.

4 ARCHITECTURE AND ALGORITHMS

In this section, we discuss the general architecture and algorithms of the Web Services interface and XML Signature validation. Section 4.1 gives an overview of the XML representations of the main STMS data objects. Section 4.2 discusses the cornerstone of the XML Signature validation research: the custom transform. This algorithm enables XML Signatures to be used to in conjunction with distributed data systems. Sections 4.3-4.5 discuss the basic algorithms for signing the basis, and creating and verifying the response. Finally, in Section 4.6, the Web Service interface is introduced, and everything is integrated together.

4.1 XML Serialization of STMS Objects

In order to transfer STMS data using Web Services, they must have a platform-agnostic representation that can easily be created, understood, and transmitted. The enabling technology for this serialization is XML, the extensible markup language [3]. When dealing with XML Signatures, both the proof and basis need to be serialized using a custom schema.

The XML representation of the basis (see Figure 3) contains important information about the data structure: its name, algorithm, fingerprint, and timestamp/expiration date. The timestamp and expiration date are attributes of the BasisData element, and the algorithm is an attribute of the Repository tag. This XML encoding of the basis is what is signed and distributed by the source at the expiration of each time quantum.

The notion of a proof is represented by the Proof element, which has an Algorithm attribute, specifying the underlying algorithm of the repository's data structure (and thus the structure and content of the children tags). In the case of the Authenticated Skip List, three children are present: the QueryKey, DigestMethod, and HashList elements. The QueryKey contains the base 64 encoding of the query key as its value, as well as an attribute specifying whether the key is contained. In the case where the key is not contained, the QueryKey element has the SmallerValue and GreaterValue elements as children. The DigestMethod tag specifies as an attribute the Algorithm to be used when computing the hash chain. Finally, the HashList contains as children the ordered Hash elements which together make up the complete hash chain for verification. It also specifies an attribute representing the index in the list of the hash of the QueryKey (or GreaterValue, if the key is uncontained). This representation is compact and readable, while still providing easy access to the important elements during proof verification. A sample encoding of a response, containing both a basis and proof, is shown in Figure 3.

```
<stms:Response xmlns:stms="http://www.algomagic.com/stms/xml/2002-04-04" id="stms_response">
    <stms:Basis>
        <stms:Repository stms:Algorithm="ASL">CGC</stms:Repository>
        <stms:BasisData stms:ExpirationDate="2002-04-17T13:05:16Z"
                        stms:TimeStamp="2002-04-17T13:04:56Z">
            Izmy2t+fldfcIlimzGpxXg==</stms:BasisData>
    </stms:Basis>
    <stms:Proof stms:Algorithm="ASL">
        <stms:QueryKey stms:Contained="false">D0JA
            <stms:SmallerValue>Aa2v</stms:SmallerValue>
            <stms:GreaterValue>XUFAKrxLKna5cZ2REBfFkg==</stms:GreaterValue>
        </stms:QueryKey>
        <stms:DigestMethod stms:Algorithm="MD5"/>
        <stms:HashList stms:Index="1">
            <stms:Hash>zCBKfMLQDcesYxgxSqtjeA==</stms:Hash>
                                ...
            <stms:Hash>Ju8ouqu0yuo7uCxKABO0Fw==</stms:Hash>
        </stms:HashList>
    </stms:Proof>
</stms:Response>
```

Figure 3 XML encoding of a basis and a proof. The hash chain has been shortened for readability.

4.2 Custom XML Signature Transform

The XML Signature specification was designed with extensibility in mind. By taking advantage of this fact, it is possible to supplement the standard functionality with additional custom behavior. According to the specification, before a certain piece of data is signed, any number of specified transforms will be applied. Some of the standard transforms include XML canonicalization [6], base 64 decoding, XPath [8] and XSLT [4] transformations, and so on. The transforms are applied sequentially, with the input of the first being the original data, and the output of the last being the data that is subsequently digested and signed. As mentioned in the XML Signature specification, the input to a transform can be either a byte stream or a DOM node list.

We now define the ASL Transform (see Figure 4), whose purpose is to validate the response from an Authenticated skip list responder within the XML Signature standard.

```
if stms:Proof
        if stms:Contained = true
                computed_digest ← hash(stms:QueryKey)
        else
                computed_digest ← hash(stms:SmallerValue, stms:GreaterValue)
        for all h ← stms:Hash ∈ stms:HashList
                computed_digest ← hash(computed_digest, h)
        replace(stms:BasisData.Value, computed_digest)
return stms:Basis
```

Figure 4 Pseudocode for the ASL Transform algorithm.

Assuming prior knowledge of the input data, it is possible to construct specific transforms to operate on that data. In the case of the ASL Transform, we assume the input is a well-formed XML document conforming to the XML encoding described earlier for representing STMS data from an Authenticated Skip List source. The pseudo-code for the ASL Transform algorithm is shown in Figure 4. A detailed description follows.

The algorithm scans the input for the presence of a Proof element. If one is found, its children are validated and stored for later processing. The main processing performed is the computation of the hash chain stored in the proof; the rest of the data is supplementary to this goal. The Index attribute specifies the index in the hash list where the query key belongs (or where the larger neighbor belongs, should the key not be contained). In order to verify that this proof is indeed for this query, the hash of the query key (if it is contained) or the hashes of the smaller and greater neighbors (if it is not) replace the values stored at the specified indices. Note that these values should be exactly the same if the proof is authentic. The hash chain computation is then performed with the commutative hash function, producing a final hash value that is base 64 encoded. If the proof is valid, this value should match exactly the hash of the repository (and thus the contents of the BasisData tag); thus, the contents of the BasisData tag are replaced with the computed value.

Regardless of whether the proof is found, the output of the algorithm is going to be the subtree rooted at the Basis element. If no proof exists, then the basis is being signed, and the XML encoding is the output. If a proof exists, then the algorithm verifies the authenticity of that proof by replacing the BasisData value with the computed digest. The signature verification will succeed if and only if the computed digest matches the hash of the repository.

There is no loss of security in this method, provided that a trusted party provides the algorithm implementation. The same computations are being performed as in the standard STMS validation, they are just happening in a different context. This transform algorithm is the key that allows untrusted responders to return digitally signed documents containing the responses to queries. Its main function is to wrap the computation of the untrusted proof into the verification of the source's fingerprint, a task that was previously conducted separately, though achieved the same result.

4.3 Signing the Basis

The basis is signed once per time quantum at the source, and then distributed to all responders thereafter. The basis data is retrieved from the underlying data structure, and then combined with a timestamp, expiration date, repository name, and algorithm to form the unsigned basis object. This object is then serialized into XML, and placed in a new XML document under a Response tag, which is identified with an XML ID.[1] A given basis is only valid for the time period between the time stamp and expiration date; this is due to the dynamic nature of the source data structure, the shorter the time quantum, the faster updates are propagated. When a proof verifies against a basis, it indicates that the answer was accurate during the time period specified in the basis itself.

To sign the basis, an XML Signature is created with a reference to the Response tag as the data to be signed. The next step involves choosing the algorithms and transforms that are associated with the signature. In the case of STMS, we use the RSA and SHA1 algorithms, and for transforms, the standard XML C14N canonicalizer [6] and our custom ASL Transform. The actual signing occurs using the standard methods of the XML Signature. In this scenario, the functionality of the ASL Transform is to locate the Basis element, and pass that subtree as is to the subsequent transform. Finally, the Signature element is added to the XML document as a sibling of the Response element, and the resulting document is stored as the signed XML representation of the basis. This information is then propagated to all responders.

[1] The Response tag is slightly misnamed in this case; its function is more as an envelope, containing as children a single Basis tag and zero or more Proof tags.

```
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:SignedInfo>
        <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
        <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
        <ds:Reference URI="#stms_response">
            <ds:Transforms>
                <ds:Transform
Algorithm="http://www.algomagic.com/stms/xml/dsig/20020405#transform_asl"/>
                <ds:Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
            </ds:Transforms>
            <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <ds:DigestValue>0OUm6eGKLGYEd6M4Mf5JY+feqUs=</ds:DigestValue>
        </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue> ... </ds:SignatureValue>
    <ds:KeyInfo> ...</ds:KeyInfo>
</ds:Signature>
```

Figure 5 An example of an XML Signature associated with an STMS response. The SignatureValue and KeyInfo have been shortened for readability.

It is possible to include with the Signature the public key or X.509 certificate [24], which can be used during verification. This can make key distribution easier in some cases, however it has the additional overhead of encoding and transferring extra information along with the basis (and subsequently, the response).

4.4 Creating the Response

Due to the nature of the transform algorithm, creating a response to a user query is quite simple. The responder stores the XML document containing the signed basis that is sent by the source at the expiration of each time quantum. In response to a query, the data structure returns an AuthenticResponse that is serialized into XML, resulting in a Proof element that is appended to a copy of the signed basis document as a child of the Response. This new document containing the Proof can then be sent back to the user as a string. The process of creating the response is outlined in Figure 6. An example of the actual XML encoding of the response is shown in Figure 3.

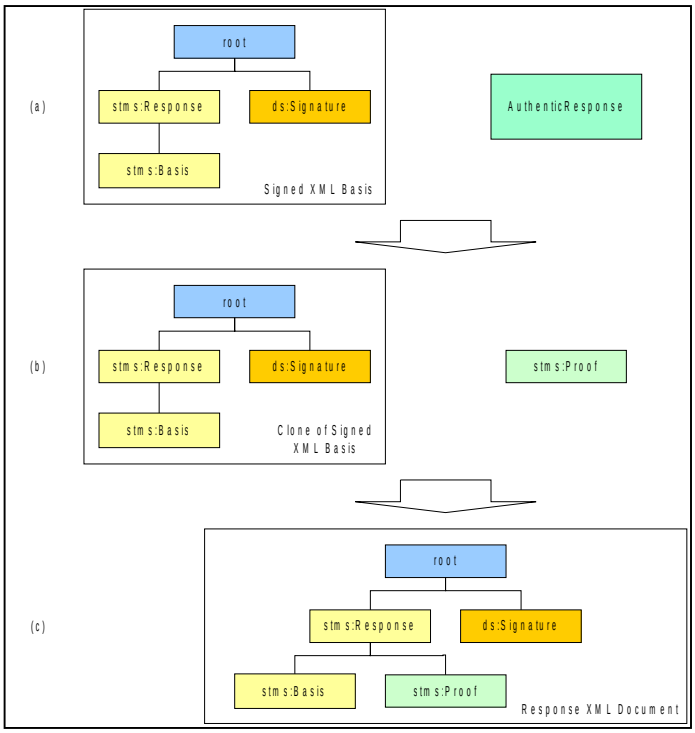

Figure 6 Creation of the response. It all begins with the signed XML basis and the AuthenticResponse (a); the document is cloned, and the AuthenticResponse is serialized into XML (b), which is finally appended to the copy and returned to the client (c).

4.5 Verifying the Proof

Once the user has received the XML document containing the Response, the verification of its integrity is reduced to a verification of an XML Signature using a custom transform. The verification process works by transforming the Proof through proven cryptographic means into information that has been previously signed by the source, namely the Basis. The ASL transform performs the computation part of the proof verification, replacing the original BasisData value with the computed value. The validation is left up to the digital signature; if the computed value is equivalent to the hash of the data structure that was signed at the source, then verification will succeed; otherwise, it will fail.

4.6 Integrating Web Services/SOAP with STMS

To write an STMS client, one previously had to use Java RMI to communicate with the responder. While this mechanism does achieve a certain degree of portability and interoperability between client operating systems and devices, it requires a large custom and complex toolkit that is tied to a single programming language. By allowing the STMS responder to be accessed as a Web Service using SOAP, much of the communications protocol can be abstracted out using platform-agnostic protocols for which a plethora of toolkits are available. Given the recent rise in popularity of Web Services, support for them has been integrated into major development platforms such as Java and .NET. Additionally, having a Web Services framework for STMS allows it to seamlessly integrate with other Web Services—as an authentication/credential provider, for example.

The service, schematically illustrated in Figure 7, accepts queries from clients and returns the response as signed XML. The goal of this Web Service is to reduce the size and complexity of the client toolkit, allowing it to better integrate with existing toolkits and standard digital signature techniques and technologies.

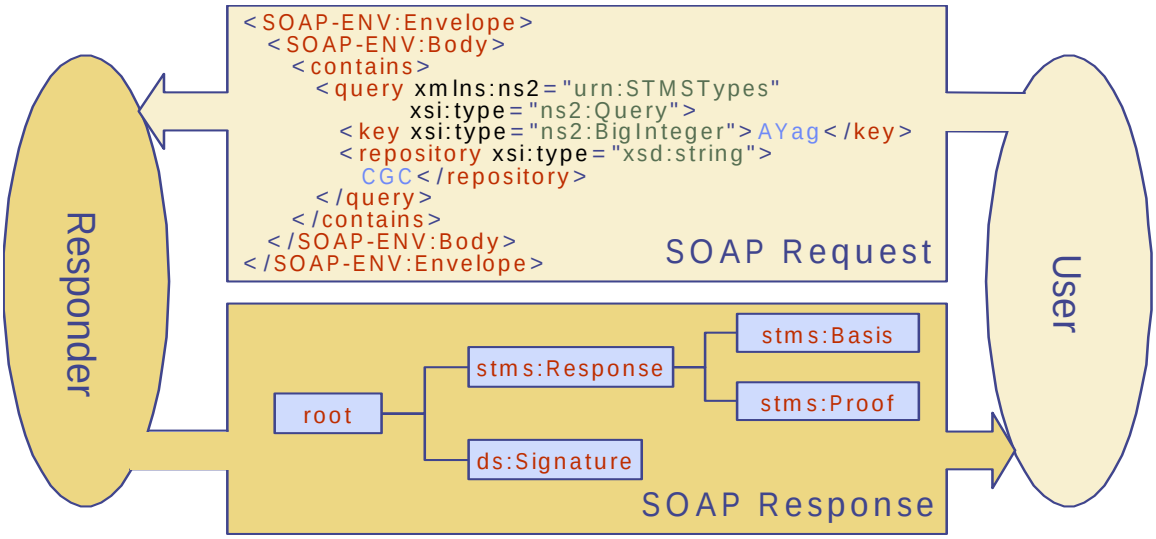

Figure 7 A SOAP-encoded query returning a document signed with the XML Signature standard. See Figure 5 for an example of the Signature element, and Figure 3 for an example of the Response.

The service implementation is quite simple. The process for generating an XML response to a query has been described above in Section 4.4. When the SOAP request comes in, it is forwarded to a method in a proxy class that performs the query on the dictionary and receives

the signed XML document as the result. It then returns this document to the client, encoded as a string.

Instead of parsing the result back into complex objects, the client simply takes the XML string and returns it to the user. This document will most likely be run back through an XML parser once the client receives it. In this scenario, verification becomes trivial: the Signature element must be located and then used in conjunction with an XML Signature toolkit to perform signature verification. If the certificate/public key information is not encoded in the signed document, it must be procured through other means (such as XKMS).

It is this verification step that yields the greatest savings in complexity over the previous methods. Being able to treat an STMS response as a true digital signature allows it to fit in existing security paradigms, and integrate into existing applications with minimal custom handling. Although it is not immediately apparent, the ASL transform still relies on the presence of a commutative hash function and an appropriate comparator for the query keys.

This type of system could be implemented as a part of any authenticated distributed database, and serves mainly to make the 3-level architecture (source, responder, client) appear to the client as only a 2-level architecture (source, client). This is achieved through the verification of the response, which is treated as a standard digitally signed piece of data from the source. The main contribution of this work is this "elimination" of the middle party from the end-user verification process, thus simplifying and standardizing the authentication of distributed data.

5 IMPLEMENTATION

One of the goals of this project is to allow STMS to become more platform and language independent on the client side, thus allowing queries to originate anywhere and on any device. To that end, a prototype implementation of the architecture discussed in Section 4 has been implemented in the two major platforms for next-generation Web Services. The first is Java, combined with various other toolkits freely available from the Apache Software Foundation [22]. The second is Microsoft's .NET platform [25].

5.1 Java Implementation

The Java implementation is written using the JDK version 1.3.1, and relies heavily on external toolkits for various standards such as SOAP, XML, and XML Signature. We chose to use those from the Apache Software Foundation [22], as they are freely distributable, open-source, and widely used. The XML Parser used is Xerces-J 2.0.1. The SOAP toolkit is Apache SOAP 2.2, and the XML Security package is Apache XML Security 1.0.3. Finally, we use Jakarta-Tomcat 3.3 as the servlet engine for interfacing with Apache SOAP and listening for SOAP requests. It is important to note that the implementation of this research is not limited to these specific toolkits; it is certainly possible to use a different servlet engine, SOAP package, or even XML Security package. With the standardization of the Java APIs for XML Security, using different packages should be as simple as switching XML parsers.

5.1.1 Classes

The addition of a few new classes to the STMS package is necessary for the system to function. The XML representations of the proof and basis are generated by the stms.io.newxml.XMLDataWriter class, which relies on the JAXP (Java API for XML Processing) API. The ASL Transform is implemented in the stms.io.newxml.ASLTransform class. It is an implementation of the org.apache.xml.security.transforms.TransformSpi abstract class, the main super-class of all transforms in the Apache XML Security package. The source has an instance of the stms.daemon.source.XMLBasisSigner class which, when given a SignedBasis object, generates the signed XML representation of it, storing the resulting Document in the SignedBasis itself.

To request and verify a proof on the client side, two classes are used. The first class is the SOAP proxy client, stms.tools.SOAPSignatureClient, which configures the Apache SOAP toolkit

to make the remote method call to the responder. The second class, stms.tools.XMLSignatureVerifier, performs the verification of the XML Signature using the Apache XML Security toolkit. Based on the client's need, both wrapped (those that perform validation) and unwrapped (those that do not perform validation) query methods are available.

5.1.2 Deployment

Deployment of the STMS clients is meant to be as simple as possible. A number of JAR files need to be installed, including those for Apache SOAP, Apache XML Security, and Apache Xerces. It is also possible to construct a smaller JAR containing only the necessary subset of the STMS classes that are required for a client.

5.2 .NET Implementation

The ability to develop Web Services in Microsoft's .NET framework, combined with its prevalent target platform, make it a major candidate for an implementation of an STMS client. We have implemented a full client toolkit that is capable of communicating with an STMS responder using SOAP and verifying results using both the standard verification and signed XML methodology. This toolkit contains 19 classes and less than 2000 lines of code, and a large portion of it is the object structure for the response and the custom byte canonicalization of the basis, items which are used only for standard verification, not XML Signature verification.

The toolkit is implemented in C#, the native language of the .NET framework, though it is accessible by any language that is compatible with .NET.

5.2.1 Classes

The exposed API contains methods that take care of wrapping the SOAP call and validating the response; this allows the user to perform queries on an API similar to that of a standard data structure, improving code readability. Security violations are reported through the use of exceptions. Additional methods exist that give a finer grain of control over the query and verification process, should the user require that.

The .NET framework has built in support for the XML Signature standard in the System.Security.Cryptography.Xml namespace. We have developed an implementation of the custom authenticated skip list transform (Section 4.2).

5.2.2 Deployment

Client deployment in the .NET framework is slightly easier than for Java, as the SOAP and XML Signature toolkits are built in to the .NET framework. Thus, installation requires only the addition of a signed assembly to the GAC (Global Assembly Cache), the local registry of all installed .NET components. Furthermore, registration of the custom XML transform must be done by manual modification of a system configuration file.

6 EXPERIMENTATION

This section presents some experimental results of the described implementations. All client performance evaluations were performed on a dual Intel Xeon 1.7GHz machine with 1GB of RAM and 100Mbit LAN connection. It was running Windows XP Professional, Java JDK 1.3.1-01 Standard Edition, and Microsoft .NET Framework v1.0.3705, under moderate (50%) CPU load. When dealing with message sizes, numbers represent the number of bytes contained in the entire SOAP envelope, but excluding transport-protocol (e.g., HTTP) overhead.

6.1 SOAP

Because SOAP is a text-based protocol, exchanging data tends to be more verbose than using heavily optimized binary-based ones. An important factor in determining the scalability of the Web Service interface is the amount of data transferred per query. For the experiments

mentioned below, we found queries generated with Apache SOAP averaged 586 bytes, while queries generated by .NET averaged 723 bytes (see Figure 8a). The difference can be attributed to the different XML styling used by default by each implementation. The signed XML that is returned as a result of a query had an average size of 3,900 bytes, regardless of platform. Variations in message size are a result of the varying length of the query key, as well as containment status and hash chain length in the responses.

The time required to perform the SOAP RPC is very similar on both platforms, and depends heavily on network congestion and other quality of service parameters. By running both the client and responder on the same machine, we attempted to isolate network conditions and evaluate the overhead of using SOAP. In this situation, we found that the .NET client took an average of 14 milliseconds per query, while the Java client took an average of 19 milliseconds per query. This additional overhead of using SOAP appeared to be negligible when compared to the interoperability gains we acquired.

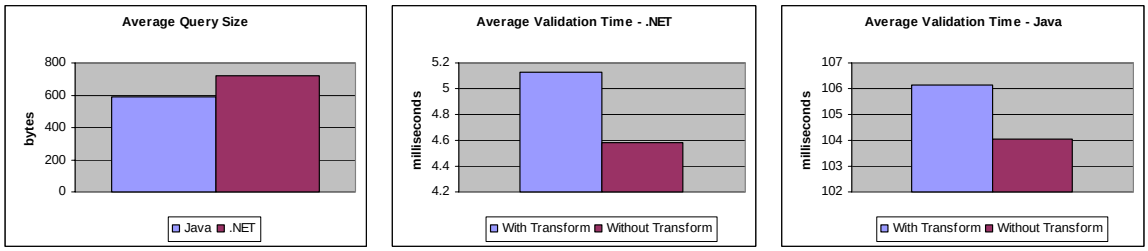

Figure 8 (a) Average query time in milliseconds for our Java and .NET implementations; (b) Average validation time for the .NET client, with and without custom transform; (c) Average validation time for the Java client, with and without custom transform.

6.2 XML Signature

In order to evaluate the efficiency of our custom transform, we performed two sets of performance tests: the first includes the custom transform, and the second has a modified version of the transform which always returns the Basis subtree, regardless of whether a proof is present (essentially, it bypasses the cryptographic computation of the proof). The motivation for this approach is to show that the additional overhead of our approach is negligible when compared to the verification times of "standard" XML Signatures. We performed 5 repetitions of a series of 10,000 queries on a dictionary with 514,000 random elements. The queries were sequential, starting at 1,000,000. We expected the number of contained queries to be very small, thus showing the algorithm's case worst (additional parsing and processing must take place when an item is not contained in the dictionary).

The results so far have been encouraging. In the .NET client, verification with the transform took an average of 5.12 milliseconds, while verification without the transform took an average of 4.59 milliseconds. On the Java side, verification with the transform took 106.14 milliseconds, while verification without took an average of 104 milliseconds. These results are shown in Figure 8b and 8c, respectively. While the toolkits exhibit dramatically different absolute times, the relative differences between the verification with and without the transform show that, on both platforms, the impact of the proof verification on overall verification time is minimal.

6.3 Interoperability

A successful outcome of our experiments is the seamless interoperability achieved between the Java and .NET implementations using SOAP and XML Signatures. In order to achieve full compatibility with SOAP, three main issues were addressed.

- The first issue is the need for custom SOAP serializers in Java. We found that the internal object structure for STMS queries and responses could be expressed in a simpler and more efficient manner by using custom written serializers that simply plug in to the Apache SOAP package. This also gave us a very fine grain of control over the format of the messages, which made it easier to build a .NET client.
- The second issue that aided interoperability is the fact that the .NET client was written from scratch; we were free to build the object hierarchy in C# around the format of the SOAP messages, thus allowing us to use the built in SOAP serialization with the aid of code attributes.
- The final issue has to do with the byte-level representation of data in Java and .NET. In STMS, query keys are stored as the most basic type of data—a byte array—and interpreted based on the application. Thus, it is imperative that the interpretation be the same on different platforms. We found that the endianness of the byte representation of an integer was reversed between Java and .NET (additionally, .NET has no equivalent of an arbitrary-length integer data type, similar to the `java.math.BigInteger` class), which caused some initial confusion. We also found that, when dealing with strings (and especially their hash values), it is important to ensure that they are encoded in the same manner on both platforms (such as ASCII, Unicode, etc.). These final caveats are simply items that application designers must pay attention to when dealing with cross-platform programming.

We encountered no major compatibility issues between XML Signature toolkits, though a few minor issues did arise. There is a well-known problem of identifying ID attributes in XML without using a DTD; however, both toolkits interpret the attribute name “Id” (case-sensitive) as being an ID attribute, thus allowing us to use document-relative URIs when referencing the data that is signed.

7 CONCLUSION

The ability of STMS to be accessed using standard technologies by lightweight clients will greatly increase its appeal. In the previous state of affairs, one must have an STMS client capable of running Java and making RMI calls through the Internet to perform a query on a responder. Clearly, the scalability of this approach is questionable: many system administrators would be reluctant to open additional ports on firewalls to accommodate STMS traffic, and clients would be limited to relatively powerful devices with high connectivity. In order for STMS, and technologies like it, to become a widely used and pervasive technology, clients must be able to access it from any location, any platform, and any device.

The main contribution of this research towards the goals of scalability and interoperability is the design and development of a SOAP-based query client for STMS. With tremendous industry-wide support, SOAP implementations are widely available, and very scalable to the size of the Internet. The Web Services interface provides a standard framework for performing queries on authenticated dictionaries over the Internet. Additionally, it allows clients to spend less code dealing with the serialization, canonicalization, and communication of data by delegating those tasks to already implemented standards. This, in turn, motivates smaller, simpler clients on many different possible platforms.

The primary contribution towards the goal of simplifying distributed authenticated dictionary proof verification is the development of a methodology for proof verification via a standard XML Signature. The main advantage of this approach over the previous verification

technique is the ease of client use. By allowing a proof to be verified using a standard cryptographic API (that provided by XML Signatures) in a single step, the integration of authenticated queries into applications becomes much simpler and easier. While this approach still requires a small client library to be installed, its use makes the fact that proofs are returned from untrusted responders transparent to the end-user.

The XML Signature verification methodology has further implications for the general problem of authenticating distributed data: since a response provided by an untrusted third party requires a proprietary verification method, applications which interface with these systems must use a non-standard API to verify data integrity. Furthermore, applications that utilize multiple sources of authenticated data might require separate APIs for each source implementation. One of the unique advantages of the new methodology is a standard interface for verifying distributed data. No matter what the implementation of the distributed data structure, the XML Signature interface provides a single, common verification API to the application that wishes to deal with authenticated data, whether it is maintained locally or in a distributed system such as STMS.

By developing prototype implementations in both Java and .NET, we have accomplished the goals of interoperability and platform-independence of proof verification. Furthermore, the prototype implementation of XML Signature-based proof verification shows the viability of such an approach to distributed data authentication. The inclusion of these technologies into the core STMS system will enable a new breed of applications. One such application is the end-to-end integrity of Web content [9], where an Internet Explorer .NET plug-in calls our .NET STMS client to verify the integrity of data within a web page integrity status.

Finally, there is much more future work to be done in this area. More extensive performance evaluations are required in order to optimize the algorithm and compare approaches. Also, there are currently only two implementations; there exist many more XML Signature and SOAP toolkits for which clients and transforms can be written. The current implementation only verifies a single proof per document; it would be significantly more efficient to be able to handle the verification of multiple proofs from the same repository with a single XML Signature validation. The algorithm to accomplish this has been designed, but not yet implemented.

8 ACKNOWLEDGMENTS

The authors would like to acknowledge the assistance and support of all the STMS team members and especially Michael Goodrich, Robert Cohen, David Emory and Michael Shin. Brian LaMacchia and Tarik Soulami provided generous insight into the workings of the .NET XML Signature package, and Christian Geuer-Pollmann provided assistance with the Apache XML Security package.

REFERENCES

[1] A. Anagnostopoulos, M. T. Goodrich, and R. Tamassia. Persistent authenticated dictionaries and their applications. In *Proc. Information Security Conference (ISC 2001)*, volume 2200 of *LNCS*, pages 379–393. Springer-Verlag, 2001.

[2] M. Bartel, J. Boyer, B. Fox, B. LaMacchia and E. Simon. XML Signature Syntax and Processing. http://www.w3.org/TR/xmldsig-core/. W3C Recommendation, February 2002.

[3] T. Bray, J. Paoli, C. M. Sperberg-McQueen and E. Maler. Extensible Markup Language (XML) 1.0 (Second Edition). http://www.w3c.org/TR/2000/REC-xml-20001006/. W3C Recommendation, October 2000.

[4] J. Clark (editor). XSL Transformations (XSLT) Version 1.0. http://www.w3.org/TR/1999/REC-xslt-19991116. W3C Recommendation, November 1999.

[5] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Frystyk Nielsen, S. Thatte, and D. Winer. Simple Object Access Protocol (SOAP) 1.1. http://www.w3.org/TR/2000/NOTE-SOAP-20000508/. W3C Note, May 2000.

[6] J. Boyer. Canonical XML Version 1.0. http://www.w3.org/TR/2001/REC-xml-c14n-20010315. W3C Recommendation, March 2001.

[7] E. Christensen, F. Curbera, G. Meredith and S. Weerawarana. Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/2001/NOTE-wsdl-20010315. W3C Note, March 2001.

[8] J. Clark and S. DeRose (eds.). XML Path Language (XPATH) Version 1.0. http://www.w3.org/TR/1999/REC-xpath-19991116. W3C Recommendation, November 1999.

[9] D. Emory. End-to-end integrity of Web content with STMS. Manuscript. Brown University, 2002.

[10] W. Ford, P. Hallam-Baker, B. Fox, B. Dillaway, B. LaMacchia, J. Epstein and J. Lapp. XML Key Management Specification (XKMS). http://www.w3.org/TR/2001/NOTE-xkms-20010330/. W3C Note, March 2001.

[11] J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach and T. Berners-Lee. Hypertext Transfer Protocol (HTTP/1.1). http://www.w3.org/Protocols/rfc2616/rfc2616.html. RFC2616.

[12] M. T. Goodrich and R. Tamassia. Efficient authenticated dictionaries with skip lists and commutative hashing. Technical Report, Johns Hopkins Information Security Institute, 2000. http://www.cs.brown.edu/cgc/stms/papers/hashskip.pdf.

[13] M. T. Goodrich, R. Tamassia, and A. Schwerin. Implementation of an authenticated dictionary with skip lists and commutative hashing. In *Proc. 2001 DARPA Information Survivability Conference and Exposition*, volume 2, pages 68–82, 2001.

[14] M. T. Goodrich, and R. Tamassia and J. Hasic. An Efficient Dynamic and Distributed Cryptographic Accumulator. In *Proc. Information Security Conference (ISC 2002)*, Lecture Notes in Computer Science, Springer-Verlag, 2002 (to appear).

[15] P. C. Kocher. On certificate revocation and validation. In *Proc. International Conference on Financial Cryptography*, volume 1465 of *Lecture Notes in Computer Science*, Springer-Verlag, 1998.

[16] R. C. Merkle. Protocols for public key cryptosystems. In *Proc. Symp. on Security and Privacy*. IEEE Computer Society Press, 1980.

[17] R. C. Merkle. A certified digital signature. In G. Brassard, editor, *Advances in Cryptology—CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 218–238. Springer-Verlag, 1990.

[18] M. Naor and K. Nissim. Certificate revocation and certificate update. In *Proceedings of the 7th USENIX Security Symposium (SECURITY-98)*, pages 217–228, Berkeley, 1998.

[19] Premkumar T. Devanbu, Michael Gertz, April Kwong, Chip Martel, G. Nuckolls, Stuart G. Stubblebine. Flexible authentication of XML documents. *ACM Conference on Computer and Communications Security 2001*: 136-145.

[20] W. Pugh. Skip lists: a probabilistic alternative to balanced trees. *Commun. ACM*, 33(6):668–676, 1990.

[21] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[22] Apache Software Foundation. http://www.apache.org/.

[23] STMS System Architecture. Presentation. http://www.algomagic.com/presentations/software-architecture.ppt.

[24] ITU-T Recommendation X.509 (1997 E): Information Technology - Open Systems Interconnection - The Directory: Authentication Framework, June 1997.

[25] Microsoft .NET. http://www.microsoft.com/net/.