Authenticating Web Content with Prooflets

Michael Shin  
Brown University

Christian D. Straub†  
Brown University

Roberto Tamassia  
Brown University

Daniel J. Polivy  
Microsoft Corporation

ABSTRACT
In this paper, we describe an efficient approach for securely authenticating dynamic content embedded in a web page. This technology, which we call prooflets, consists of an extension of the HTML tag library and of a service for publishing and distributing authenticated data. Prooflets leverage emerging authentication methods, such as authenticated dictionaries and XML digital signatures.

We show how prooflets protect Web content against the risk of tampering by providing cryptographic proofs of integrity. We also demonstrate the advantages, efficiency and scalability of prooflets and discuss their applications to a variety of data authentication problems. Finally, we present our prototype implementation of prooflets by means of a browser toolbar and of a distributed authenticated Web service.

Author Christian Straub is a full time student.

1. INTRODUCTION

Often, people and corporations alike assume that data is authentic merely by implicitly trusting the entity that delivers it. This is evident in the numerous stock tickers, sports scores, weather forecasts, and news articles widely available on the Internet. But as digital information becomes more pervasive, the threat to the integrity of the data increases. Clearly, there exists a demand for an efficient system that will guarantee the end-to-end integrity of content from a trusted source to a client, especially when it is distributed and aggregated through third parties.

In this paper, we present prooflets, a scalable architecture for authenticating Web content. Prooflets overcome the major limitations of the traditional approach to content authentication, which is based on signing a time-stamped cryptographic hash of the content. Prooflets are especially effective when the content rapidly changes over time (e.g., stock quotes). We have designed prooflets by leveraging standard web technologies. Thus, prooflets can be readily adopted in today’s Web publishing applications.

1.1 Main Contributions

This paper introduces prooflets, describes their role in web services, and analyzes the security of this new technology. The main contributions of our work are as follows:

- We define prooflets and show how they provide a mechanism to verify the integrity of (or securely retrieve) data from a trusted source of information that is delivered to a client via intermediate third parties.
- We design a prooflet tag library that extends standard HTML tags and can be readily embedded in any HTML document,
- We introduce the End-to-End Integrity Client (EEI client), a web browser enhancement serving as a trusted application for the verification of prooflets.
- We demonstrate how the EEI client can effectively inform the user about the authenticity of web content that is tagged with prooflets.
- We show how prooflets allow portal sites to reduce the liability and costs associated with aggregating and publishing securely Web content originating from third part providers.
- We describe how prooflets achieve architectural, computational and economic scalability through the incorporation of authenticated dictionaries, an emerging authentication technology for dynamic and distributed data.

1.2 Requirements

The following two primary requirements have guided the development of prooflets:

- First, we seek to ensure that information sent from a distributor on behalf of a trusted content provider arrives at the client intact, i.e., it matches the records of the provider’s database. If the data received by the end user is different from the provider’s records, the user must be able to detect the modification.
- Second, we want to provide authentication by extending rather than supplanting the current Web content delivery infrastructure. That is, the authentication components must be specified in a standard way, and consequently, must be compatible with web service platforms such as Enterprise JavaBeans (EJB), ColdFusion, and .NET.

By the above requirements, prooflets include a tag library that extends the standard HTML tag set in common use today. By representing prooflets with HTML tags, we make prooflets available to any publisher that has capability of developing static or dynamic web pages.
1.3 Previous Approaches

Several methods have been devised for authenticating web content and numerous approaches, reviewed below, are currently available. It is important to note that proofs were not designed to replace such approaches, but rather leverage standard web services technology.

1.3.1 Transport Layer Security (TLS)

TLS technology [10] is in widespread use and serves a variety of applications on the Internet. It functions by exchanging a secret key via known public key delivery systems. All data between client and host are then encrypted and decrypted using this key. While the technology is primarily concerned with protecting the communications channel, it can also be used to ensure end-to-end integrity from the host to the client.

The use of TLS for end-to-end integrity is not fully satisfactory for various reasons. The main problem is that, the integrity of the data is ensured only in the communication between the end host and the client, where the end host is typically a web server (or some other content provider). However, content is often originated from a source (e.g., news items from Reuters) and then distributed and published through third-party servers (e.g., Yahoo!). Such content could be tainted either at the web server level, or between the source and the server. TLS is vulnerable to this type of attack that does not compromise the stream of data from the web server to the web browser.

Thus, when using TLS, additional security is needed (1) between the external data sources and the web servers that deliver them and (2) at the web server. In addition, the user needs to trust not only the source of the content but also the third-party web server distributing it.

One further problem with the TLS architecture is that once the data is received over the secure connection, we can no longer prove that the data originated from a trusted source. In contrast, data sent over an authenticated dictionary system cannot be repudiated because the content is ultimately protected by a digital signature.

1.3.2 Public Key Infrastructure

Another approach, popular in email applications, is the use of messages signed by public keys. Such systems as Pretty Good Privacy and digital certificates create a digital signature (using the source’s private key) of the hash the data, thereby certifying its authenticity. There are two drawbacks to this method.

- First, it requires the source to respond to all queries (in order to sign the data), thus introducing extreme load on the server and opening it up for outside attacks. The source could be replicated to handle the first issue, but protecting a large number of source replication servers is not economically scalable.

- Second, the technology used to verify digitally signed web documents is still in its infancy. TLS, which makes use of public key technology, signs documents at the web server level, not at the source. The source could theoretically push requests to the web servers (and sign the data as well), but this brings about the same scalability issues that were addressed earlier.

1.3.3 Web Spoofing Countermeasures

Another method to combat the unauthorized modification of source data is to simply make the end user aware of material that can be potentially unsafe. Ye and Smith [18] developed an approach that splits web content into trusted and untrusted sections. Any data not originating from a specified source server would appear in the untrusted sections. This method is useful for determining whether entire pages have been modified, but cannot achieve fine granularity (i.e., it cannot isolate discrete authenticated sections in the document). In addition, it heavily relies on human intervention to detect spoofing or tampering.

1.4 Authentication Architecture

The authentication architecture underlying proofs, called an authenticated dictionary system (ADS), includes a trusted source and associated responders. The source is a server trusted by the user that maintains the database records. Content providers are trusted entities that can modify the source database. A responder is a server storing a complete copy of the source database and responsible for handling requests issued from clients on behalf of the source. Clients are referred to responders by distributors or publishers, who serve the prooflet-enabled documents. The ADS architecture is described in greater detail in Section 4.

1.5 Organization of the Paper

The rest of this paper is organized as follows. We begin with a detailed discussion of prooflets in Section 2. Section 3 addresses the security aspects of prooflets from the client’s perspective. Section 4 describes authenticated dictionaries, the efficient authentication infrastructure behind prooflet technology. Sections 2 and 6 illustrate the use of XML signatures in prooflets. A web services infrastructure supporting prooflets is discussed in Section 7. Section 8 overviews additional deployment issues. Section 9 concludes the paper.

2. PROOFLETS

Prooflets are a technology for publishing authenticated data and making it available in a highly distributed manner. A prooflet tag is a specific HTML tag delimiting a portion of an HTML or XML document that can be authenticated or securely retrieved through a prooflet. Prooflet-aware browsers recognize such tags. A sample document containing prooflets is shown in Figure 3.

2.1 End-to-End Integrity Client

Given that prooflets are a new technology, there are no current prooflet-aware browsers available. Instead, we have developed an End-to-End Integrity client (EEI client) that can be installed on top of conventional web browsers such as Internet ExplorerTM and NetscapeTM. The EEI client communicates with the browser through the COM+ API and is able to intercept and preprocess documents containing prooflets before they are displayed to the user.

The EEI client parses the prooflet tags, performs the associated authentication query, and verifies the integrity of the requested data. In our prototype implementation of the prooflet architecture, the client interface is a toolbar object that appears next to the browser’s standard buttons (see Figure 1). The inclusion of the EEI client is thus unobtrusive to the end user, yet provides the user with sufficient visual feedback.

2.2 EEI Client User Interface
In addition to resolving the prooflet references, the EEI client also allows the users to further discern additional prooflet information and modify the presentations of prooflets in the document. To provide for a uniform look and feel for prooflets, a user may specify that a particular style be used for valid and invalid prooflets. Right clicking on a prooflet will present the user with additional information such as the trusted source of the content, the key tag used to identify the prooflet, and the expiration date. The user may also quickly view all prooflets embedded on the page (and their validation status) in a single window, and be able to navigate to a particular prooflet by double clicking the prooflet in this window. Note that all prooflet ’windows’ are not browser windows, but rather formed by Windows API and thus not modifiable by client side code. An example of this prooflet window can be seen in Figure 2.

The EEI client also has a notion of a time-to-live (ttl) for prooflet enhanced data. The user may specify the maximum ttl of a prooflet (based on a source, or key tag, etc). After this lifetime has expired, the prooflet will invalidate itself and update its screen presentation to reflect this change. A complete screenshot of the EEI Client can be seen in the Appendix.

2.3 Components

In a minimalist environment, a prooflet tag contains a control domain, a request type, and a key or value. The control domain uniquely identifies the content source and the associated set of responders to access. Content providers are responsible for maintaining their own prooflet source. The inclusion of the control domain allows a prooflet to specify which of these systems to communicate with. The request type is either retrieval, in which case the key to the data is supplied, or containment (authentication of data within the prooflet).

In addition to the requisite attributes necessary to generate the authentication query, numerous optional attributes may be included in the prooflet tag. For instance, the CSS display style of authenticated (and non-authenticated) data can be specified. A behavior may be set if some data is not verified (i.e., a level of warning to the user can be specified). The prooflet tag can also specify whether or not to aggregate prooflet responses.

2.4 Base Classes

Since a document may consist of many prooflets tags, all with similar sources, query methods and control domains, the concept of a prooflet base class was developed. A prooflet base class contains much of the configuration attributes presented in Section 2.3. Such base classes are declared first and then can be used by the prooflets. Prooflets can specify a base class to inherit from; the tag then needs only to supply the attributes specific to that tag, such as the key or value to request. In addition, a prooflet may override base class attributes, and other base classes may inherit (and override) the attributes of a previous base class.

2.5 Embedding Prooflets into Web Documents

Since prooflets tags are based on standard HTML tags, integrating them into HTML documents is routine. Prooflet tags can either be standalone (self contained XML snippets) or span large amounts of text (with start and end tags). Since prooflets can be embedded into text, they can be contained within spans, placed in DHTML layers, dynamically displayed, etc. In addition, prooflets can be customized and deployed at runtime by web services technology or XSLT transformations. Prooflets can also be constructed interactively by end users during runtime and deployed by re-loadable internal frames. Any technology that can produce text can formulate a prooflet, and the web developer can leave the mechanics of the authentication routines to the EEI client.

As can be seen in Figure 3, prooflet base classes are specified typically at the top of the document and follow a rigid XML format (indeed, they are parsed as individual XML documents). In order to provide the maximum flexibility in deploying prooflets, base class descriptions can be held externally and be referenced in the HTML document just as any external XML data source would typically.

Prooflet base tags are optional and are usually contained within XML external references that are recognizable by later browsers. (For complete backwards compatibility, these may be ommitted). The actual prooflet tags are an extension of the standard span HTML tags. For non-prooflet aware browsers, the type identifier will be ignored, and the content will be displayed to the user without verification. Thus, embedding prooflets in a document will not interfere with the presentation of that document to older or non-prooflet aware browsers.

2.6 Extensibility and Applications

A standard authentication interface is provided by the prooflet distribution system, which reduces the authentication process to that of verifying an XML digital signature. (See Section 7 for more details.) Since prooflets are expressed in XML, custom attributes can be easily integrated into prooflets. Additional clients based on the prooflet technology would need only an XML parser and a means to communicate with the prooflet responders. Thus, prooflets could be deployed in venues other than HTML documents. Indeed, prooflets could annotate XML documents, email, or any other distributable medium sent in plain text.

Furthermore, the data protected by prooflet technology
can span farther than simple text. For instance, data such as images, sound, and even videos can be authenticated by prooflets. Here, the prooflet would take the cryptographic hash of the multimedia object and use this to verify its integrity. Thus, prooflets are a powerful verification technique that is not only easily deployable, but can be used in a multitude of authentication scenarios.

2.7 Proxy Server Implementation

Another means to realize prooflet transactions (other than via the EEI client) is through the use of a prooflet proxy server. In situations where clients have a secure link to the proxy server (i.e., corporate LANs, VPNs, etc), a prooflet proxy server could serve the same purpose as the EEI client. An example of this architecture is illustrated in Figure 4.

All clients can forward HTTP requests through the proxy server. The proxy server will then analyze the requested documents to ascertain whether or not they contain embedded prooflets. If the document contains prooflets tags, then the proxy server can initiate the prooflet query on behalf of the client. The proxy server can then embed the prooflet response in the HTML document (effectively replacing the prooflets with the associated data) and send the plain HTML document back to the client. In the case that the data is not authenticated, the Proxy server can annotate the document to indicate that an integrity violation was detected.

The addition of a prooflet proxy server eliminates any custom browser enhancements at the client level. In addition, it makes the inclusion of prooflets transparent to the end users. Having all validation efforts performed by a small set of Proxy servers also eases the deployment of prooflet technology to existing networks.

2.8 Additional Advantages

Although prooflets aim to solve the problem of data integrity, there are two corollary advantages that prooflets provide.

2.8.1 Liability to the Provider, Not the Distributor

Under previous distribution systems, portal sites and other distributors of content would first gather the content from various external sources, combine them into a web page, and finally send them to the client when the document is requested. Since the distributor serves the content on behalf of the provider, it is the distributor who has the responsibility to ensure the integrity of the data. Alternatively, the distributor needs only to refer users to the various providers through the use of prooflets. Prooflets
can then handle the authentication and retrieval of data. Since this data would typically originate from a source server under the content provider’s control, the burden of liability is placed on the provider.

2.8.2 Aggregation of Content

Since many portal sites contain content from a wide array of different external sources, if a portal site wished to guarantee the integrity of data from each source, they would have to provide secure channels between a server under their control and a source server. Proolets ease the aggregation of data by allowing the distributor of data to merely specify which source to take (or authenticate) the data from. The security and distribution of data is handled automatically by proolets.

3. CLIENT-SIDE SECURITY CONCERNS

This section is primarily concerned with maintaining security once the data arrives from the source via the proolets distribution model. We describe the user interactions and visual displays that help the user detect security infractions. We should note that employing such attacks to defeat proolet authentication is difficult and requires intimate knowledge of the structure of the source document as well as access to modify it. Even so, security by obscurity is not a proper defense against these sort of attacks. This section deals with the types of attacks against this type of system, and the measures taken to counter such attacks.

3.1 Visual Display of Authenticated Data

In order to keep proolets as general and flexible as possible, the display of data is completely customizable by the publisher. Proolet attributes determine how the data should be presented and whether it is authentic. These attributes can be specified in a proolet base class or within the tag itself. Such attributes take the standard CSS specifications to allow a familiar setup to website developers. Additional options are also available. For instance, the developer may specify that the user be immediately warned (via pop-up, sound, etc.) when tainted data appears on the web page. Actions may also be specified as to what may happen if the user clicks on a proolet in the document. In these instances, the events trigger proolet routines (i.e. popping up a window using the Windows API) and thus are independent of browser capabilities. Thus, the user will receive such indications even if, for instance, they have disabled Javascript.

In addition, there are two other methods that aid the user in determining which content is authenticated. If the user right-clicks on text that is contained within a proolet, the new window appears containing the complete specifications of that particular tag, including responder information, expiration date, and the source database. The user may also click on a button in the EEI client toolbar to display the records of all authenticated information contained within the document.

3.2 Data Modification Countermeasures

Although proolet technology can assert that data is authenticated at the time of page loading, a malicious user with access to the source document could insert code that replaces authenticated data at a specified time interval after the page has been displayed. Since this occurs after authentication, corrupted data (especially if it is on screen) may appear to be authenticated. In order to counter this threat, after the EEI client receives the proolet responses, it caches them in a local hashtable that remains current until the document changes. Every time the web page content is modified by client scripting code, an event trigger is fired and the EEI client compares the data contained within the proolets with the data in the hashtable, thereby re-authenticating the data without the need to re-query the source server via the proolet distribution sys-
3.3 Detecting Prooflet Imitation

Although the user can be immediately notified when tainted data appears on the webpage, the reverse is not true. Again, a malicious user with access to the source document could format data not contained within a prooflet to look and appear like the data that is in fact validated through the EEI client. Having the EEI client scan through the document and detect such infractions is not a viable solution as it can substantially delay page load speeds, and is further hindered by the fact that the style (and therefore the presentation) of certain text can be modified at runtime.

Another method is to have the user validate the data manually. That is, the user can confirm the validity of data whose visual appearance suggests that the data has been authenticated. One approach that is used by the EEI client allows user verification to happen instantaneously and with little effort. Every time the user moves the mouse over a prooflet protected data, an icon indicating whether or not the prooflet is valid will appear on the prooflet band bar that is embedded in the browser and is thus off the editable DOM of the webpage. (See Figure 1). Thus, if a user doubts the authenticity of some data, all that is required is that they hover the mouse over the data. If the icon does not appear, then the data is “falsely authenticated”. Since the EEI client selects the top layered component to check, covering authenticated data by the use of DHTML layers will also not trick the user. This technique was chosen since it cannot be mimicked by client-side scripting techniques and thus is unique to the EEI client.

Finally, for all prooflet windows that contain information for the user, a personalization string is displayed on those windows. This personalization string is chosen when the EEI client is installed and differs among installations of the client.

3.4 Man-in-the-Middle Attacks

The previous attacks we have discussed assumed that the prooflet transmission and verification occurred successfully, but visual modifications to the display were made to deceive the user of the prooflet’s status. The prooflet response is as tamper proof as a digital signature, but a savvy attacker could attempt to change the subject of the prooflet before it reaches the EEI Client. For example, if a user attempts to check the weather status in Honolulu, Hawaii, a modification to the query key could be made so the prooflet would return the weather for Anchorage, Alaska.

In order to counter this type of attack, all relevant information about a prooflet must be displayed to the user, as seen in Figure 5. With the EEI client, information such as source and key tags will be displayed to the user if the user right clicks on the prooflet. It is, however, incumbent upon a trusted source and content distributor to ensure that the key tags used are meaningful to the user.

4. CONTENT DELIVERY MODEL

In order to efficiently serve digital information to a large number of clients, several requirements must be satisfied by the prooflet authentication architecture. Specifically, it must be cost-effective, easy to maintain, secure, and scalable. The underlying technology for content delivery used in this paper is based on an authenticated dictionary system (ADS). In this section, we will give a brief overview of ADSs and describe how they are used in conjunction with prooflets.

4.1 Authenticated Dictionaries

An ADS consists of a centralized source responsible for maintaining a dynamically evolving set S of key-value pairs (or, more generally, a database) and any number of responders, which can provide authentic answers to queries made by users on S. An ADS provides users with cryptographic assurances of key membership in S and of the mapping between keys and values in S. An important feature of ADSs is that the source is the only point of trust and is not exposed to user queries. Responders, on the other hand, are considered untrusted but are nevertheless enabled to provide authentication services on behalf of the source.

The source of an ADS maintains a schedule for updating responders. We call the period of time between updates a quantum. Throughout the course of a single quantum, trusted content providers can update the set S at the source with insertion and removal operations. When a quantum expires, the source distributes to its responders a list of modifications and a signed statement, which we will call the basis. Informally speaking, the basis is a time-stamped compact digest of the current contents of set S.

When a client queries a responder for a specific key, the responder returns the key-value pair, a proof of containment, and the current basis. The proof can be considered a partial digest of the source set that, when combined with the key-value pair, yields the basis. Since the basis is signed by the source, if the responder (or any attacker between the responder and client) attempts to falsify the proof, the client can easily detected the tampering. An example of the verification process for a tree-based authenticated dictionary can be seen in Figure 6.

A complete discussion of authenticated dictionary systems can be found in [1, 5, 6, 9, 12, 13, 14, 15, 16] and a high-level overview can be seen in Figure 7.
At the start of a time quantum:
(1) The source securely distributes database updates as well as the signed Basis.
When the user initiates a request:
(2) The client queries the DNS server to locate a responder.
(3) The DNS server for the control domain selects a responder based on current server loads.
(4) The client sends the SOAP request to the specified responder.
(5) The responder answers the requests and also transmits a proof and the signed Basis as part of the response.

Figure 7: Architecture of an authenticated dictionary system.

4.2 Providers and Sources
In the simplest deployment scenario for proolets, each provider of web content is responsible for maintaining its own ADS source and responders. Since the provider locally controls the source, there is a high level of assurance that the content is not compromised between creation and insertion into the ADS. Additionally, each provider can dynamically scale the availability of authenticated content by adjusting the number of deployed responders.

Smaller content providers may choose instead to rely on a third-party to provide the ADS services. In this setting, the link between the provider and the ADS service introduces a potential vulnerability. Additionally, the provider must trust that the ADS service is not modifying the content and has a sufficient number of responders to handle the query load. Solutions to these additional security concerns are discussed in [11], which introduces the notion of cryptographic receipts.

4.3 Advanced Authenticated Data Structures
As mentioned in Section 3.4, an attacker can intercept a Web document before it reaches the EEI client and modify the keys used in retrieval queries. Thus, providers should use ADS keys that are easily understandable so that the end user can determine if the displayed authenticated content is consistent with the Web document they are viewing. Content providers that find this too restrictive, or require a richer set of queries, can use a more sophisticated authenticated dictionary that supports range queries, like the one presented in [13].

For example, a news agency may want to allow users to query for the top five most viewed news articles. Using a standard authenticated dictionary, this would be easy to spoof since it would be difficult to encode the notion of ranking in the key for the article. An attacker could replace the set of keys contained in the Web document to reflect an arbitrary set of articles (if, for example, they wished to hide negative press they were receiving). Using an advanced ADS that supports range queries [13], the provider can use proofs representing the retrieval of all and only the news items in a given range of rank (e.g., 1 through 5) or time (e.g., in the last six hours).

5. STANDARD INTERNET PROTOCOLS
In order to make our architecture as transparent as possible for use on the Internet, the proolet architecture has been built upon existing and proven standard communication protocols.

5.1 Simple Object Access Protocol (SOAP)
All communication between a proolet responder and a client are handled by SOAP calls. SOAP [3] is a platform independent XML based protocol for packaging messages between two hosts. In our implementation, all SOAP messages are sent over HTTP. We can exploit the fact that SOAP is based on XML by making use of XML signatures and by forming the data through XML transforms.

5.2 XML Signatures
The XML Signature standard [2] provides the set of rules and syntax for encoding, computing, and verifying digital signatures of arbitrary data. One of the key benefits of using XML signatures with XML data is that it can be used to sign specific portions of the XML tree rather than the complete document. This feature allows us to
send the SOAP response as a single package, with parts of the response signed by the source server. Specifically, the basis is canonicalized, digested, and then cryptographically signed. This is included in the SOAP response that is sent back to the client from a responder. Another feature of XML signatures is that they allow custom transforms to be applied before verification. As will be seen in the next section, we can take advantage of this fact to provide an automated protocol for determining the validity of a prooflet response.

5.3 XML Transforms

As will be described in the next section, the end-to-end integrity architecture utilizes XML transforms. Since the SOAP calls are based in XML, and since portions of this document are signed using standard XML signatures, we can apply transformations to the entire SOAP package of the response. Some of the transformations used in our approach include XML canonicalization [4], XPath [8], and XSLT [7].

6. XML SIGNATURE AUTHENTICATION

We define a transaction as being composed of a request for data, and the subsequent response to the request. With prooflets, a request is a query against a database controlled by the source server. Requests are handled by responder servers, which in turn formulate the response (see Section 4.1). Due to the simplistic nature of the request, we focus our discussion on the response.

6.1 Dissected View of the Prooflet Response

The response contains four main entries as depicted in Figure 6. First, it stores the basis. As mentioned in Section 4.1, the basis is a compact representation of the entire data set stored at the source server. It also contains information regarding the algorithm used to compute it. Second, it contains the query response, which consists of a Boolean describing the result of the containment query. If the key is contained, then the associated value is also included. In order to validate this query response, the response must also contain the proof, which in simplistic terms, consists of the cryptographic hash of the data structure minus the value that was requested. Finally, an expiration date and timestamp are applied to the response to indicate at what time interval the data is to be considered valid.

6.2 Prooflet Response XML Serialization

In order to send the response via SOAP, it must first be converted into XML using a custom schema as the format. A sample containment response is shown in Figure 8. In addition to containing the basis and proof, additional information, such as which digest algorithm is used, much be present to ensure proper deserialization at the client end. The query key and hash chains are represented in base 64 encoding to ensure a platform-agnostic message.

6.3 Signing and Distributing the Basis

At the beginning of a time quantum, the basis is computed at the source server and is put into its XML representation. We then make use of XML signatures to sign the part of the XML tree referring to the basis. This XML signature is then appended to the tree as a sibling to the basis (thus providing for a discreet distributable XML package containing the basis and its signature). These last two steps are performed using a custom XML transform [17]. This transform locates the basis in the XML document and feeds this value to the standard methods of the XML signature. With prooflets, the basis is signed using any standard digital signature algorithm, such as RSA with a SHA1 message digest.

We refer to this XML package as the signed basis for simplicity. Once formed, the signed basis is then distributed to the prooflet responders (along with any database updates) via multicasting.

6.4 Generating the Prooflet Response

Assuming the steps in Section 6.3 have already taken place, a prooflet responder has the latest version of the signed basis in XML format. In order to generate the response (Section 6.2), we need only to generate the query response and the proof. The proof, which in the case of a hash tree or skip list implementation, consists of a hash chain is serialized into XML using base 64 encoding. The same is performed for the value that is requested by the client. Together, these two form the response to the client’s request.

At this point, the signed basis XML document is cloned, and the previously generated client response is appended as a sibling to the basis. What remains is a single XML document containing the query response, the proof, and the basis, as well as the XML signature of the basis. This entire package can then be sent back to the client as a SOAP response for the prooflet.

6.5 Verification of the Response

The final step of the transaction is the verification of the prooflet response. As described in Section 4.1, since the proof consists of the hash of the data structure (the basis) minus the value with which we are requesting, then conceptually, all that remains is to combine the value with the proof and determine if this is equal to the basis. Performing this role on the client end however consists of having intimate knowledge of how the basis, proof, and other components of the response are serialized, and be able to undertake the operations necessary to combine the proof with the value. This last operation in particular is not trivial. To simplify the verification process, we define a custom XML transform, which is responsible for calculating the basis from the provided proof, query key, and response. This transformation is applied transparently to the user immediately before the XML signature is verified.

Revisiting one of the benefits of XML signatures, we note that a custom transformation can be specified in the

![Figure 8: Packaged ADS SOAP response.](image-url)
signature (see Figure 9) and this transformation will be applied before the signature is processed. It is therefore incumbent upon the system to perform the prooflet authentication routines through this transformation, then present the results to the user in a systematic and standard way.

The transformation first locates the proof and value sub-trees in the XML document. It then computes the combined hash of the proof with the query key and associated value. When the data is authentic, the resulting hash should be equal to the root hash stored in the basis. Finally, the transformation replaces the root hash sub-key stored in the basis sub-tree with the newly computed value.

After the transformation has been applied, the basis will be updated with the new hash value. If the data is authentic, then the new hash value will be equivalent to the old basis hash, and the basis sub-tree will not have changed. If the data is not authentic, however, then the basis sub-tree would be modified. Since the XML digital signature signs the basis, then all that is needed is to verify this signature. That is, the hash of the basis sub-tree must equal the hash that is signed by the digital signature. As shown previously, this will only be true if the data from the response is authentic. Otherwise, the verification of the digital signature would fail.

Thus, through a custom XML transform, we reduce the problem of calculating the equivalency of the proof with the data and the basis to simply verifying a standard XML signature. This transformation is algorithm-dependent, but can be specified in the XML signature during the signing of the basis (where the algorithm is known).

7. WEB SERVICES IMPLEMENTATION

Having provided the framework for the operation of prooflets through the use of XML signatures, we now describe how we implement this framework with web services technology.

7.1 Client Interaction and Responsibilities

Due to the standardized nature of the prooflet end-to-end integrity architecture, the client has minimal responsibilities in order to communicate with the ADS responders.

Specifically, all that is required is for the client to generate a SOAP request directed to a responder. This SOAP request contains the responder to contact, the key whose value must be obtained,, and the source for the data. The SOAP response, as explained in sections 6.4 and 6.5, will contain the answer, the basis, and an XML signature over the basis. All that is left for the client is to verify the basis with respect to the signature, and then obtain the value from the prooflet response. All computation required for the verification of the proof is handled by the XML transform, and is applied to the response before the signature is verified.

Due to the platform agnostic nature of SOAP, the client can be written in any number of different programming languages for any computing environment. Modern implementations of SOAP are readily available for Java and the languages of the .NET Framework (C#, Visual Basic .NET, etc.). In addition, since prooflets use SOAP through the HTTP protocol, it can be implemented through any language capable of sending requests over the Internet (i.e. standard web server requests).

7.2 Types of Prooflet Requests

There are two types of requests that can be handled by our prooflet architecture. The first is a containment request. This type of request is generally used to verify the authenticity of data. For example, a large body of text could be included in a document. A prooflet request could then be performed to verify that the text has not been modified since leaving its initial source. The values stored in the database are message digests of the actual text. The client can then take a digest of the text to verify its consistency with the value returned in the prooflet response.

The second type of request is a retrieval request. This is used when a distributor wishes to refer the client to the provider of the authentic data. Here, a key is provided in the request and the value is returned in the response and inserted into the web document. Provided that the basis is verified via the signature, the client can be assured that the value it obtains is what is stored in the source database.

7.3 Client-Server Virtual Communication

From the client’s point of view, the system appears to deploy the standard 2-level architecture between a server and a client. Since the verification of the response is handled by an XML transformation, the presence of the responder server is transparent. The effect is the same as if the client is directly communicating with the source server (since the basis is initially signed by the source server).

7.4 Integration into Distribution Systems

Since a prooflet response is handled as a standard digital signature, it leverages existing security paradigms, and integrates easily into existing applications with minimal custom handling. All that is required is to reference the correct XML transform (based on the authenticated dictionary used deployed by the prooflet) and the process reduces to a normal SOAP request.

8. ADDITIONAL CONSIDERATION

Two additional considerations of the prooflet system are discussed below.

8.1 Domain Intellectual Property

Due to the open nature of the prooflet architecture and the fact that information transmitted between client and responder are not proprietary, it is conceivable that unauthorized users may gain access to the source data and present this data as their own. More specifically, using prooflet technology, a portal site may retrieve unauthorized information from a prooflet responder by including the requisite prooflet tags in its HTML documents. A simple way to prevent this is as follows. The web server and the prooflet distribution system can negotiate a shared secret at the time that their relationship is formed. When a client accesses a web page, the web server can include a referral token which is a cryptographic hash of the shared secret and the client’s IP address. The client can then forward this token to the Responder as proof of authorization. Since the hash is one-way, the token cannot easily be forged. This technique can be extended to avoid replay attacks.

8.2 Load Balancing of the Responder Servers
One of the benefits of the prooflets architecture is that it readily allows for responders to be deployed in unsecured locations with minimal setup. However, in order to exploit the multitude of the distributed and redundant nature of these responders, the architecture must allow for load balancing between them.

In order to accomplish this goal, the control domain (see Section 2.3) is given a second purpose. With prooflets, the control domain is associated with a subset of prooflet sources. A random alphanumeric entry is added as a sub-domain to the control domain, and this URL is used to locate a responder. Since the sub-domain will (with high probability) be unique for each SOAP request, the client will be forced to resolve the responder IP with each call. The authoritative DNS server for the control domain can then select a responder in a round robin fashion when responding to prooflet responder lookups.

9. CONCLUSION

We have presented a secure and extensible system for verifying the integrity of web content. Our prooflet framework allows for content providers to easily deploy their content in a distributed and trusted way. Furthermore, the prooflet tags can be easily incorporated into HTML documents and comply with existing web standards. Our End-to-End Integrity client provides users with an intuitive and unobtrusive interface for verifying the authentication status of sections of content enclosed by prooflet tags.

Prooflets allow for a secure, high-volume content authentication system that has low overhead and small operating costs. In addition, prooflets are both architecturally and economically scalable. The use of prooflets leverages web services to achieve greater levels of portability and extensibility in authenticating web content.

10. ACKNOWLEDGEMENTS

We would like to acknowledge the work of David Emory in constructing the initial prototype of the End-to-End Integrity client. We would also like to thank Michael Goodrich, Robert Cohen, and Nikos Triandopoulos for useful discussions.

This work was supported in part by the Dynamic Collaborations Program of the Defense Advanced Research Projects Agency under grant F30602-00-2-0509.

11. REFERENCES


This is a sample prooflet-enabled website taken from a recent demo of the EEI Client. Shown are some of the user configurable options of the client and the expanded view of a prooflet’s information.