Securely Replicated Web Documents

Bogdan C. Popescu, Maarten van Steen, Bruno Crispo, Andrew S. Tanenbaum
Division of Mathematics and Computer Science
Faculty of Sciences Vrije Universiteit, Amsterdam, The Netherlands
{bpopescu, steen, crispo, ast}@cs.vu.nl

Jan Sacha
Computer Science Department, Trinity College Dublin, Ireland
Jan.Sacha@cs.tcd.ie

Ihor Kuz
National Information and Communications Technology, Sydney, Australia
Ihor.Kuz@nicta.com.au

Abstract

In order to achieve better scalability and reduce latency in handling user requests, many Web applications make extensive use of data replication through caches and Content Delivery Networks. However, in such scenarios data is often placed on untrusted hosts. As a result, existing replication mechanisms open a wide class vulnerabilities, ranging from denial of service to content masquerading. In this paper we present an architecture that combines data content, replication strategies and security in one unified object model and offers integrity guarantees for Web documents replicated on non secure servers.

1 Introduction

Starting with the late 90's, the World Wide Web has been experiencing scalability problems due to irregular data access patterns. When a Web document suddenly becomes very popular (a phenomenon known as a *flash crowd*), clients experience long delays in retrieving it. The single hosting server simply cannot cope (CPU-wise or bandwidth-wise) with the sudden high demands. A possible solution to this problem is data replication.

A common way of replicating Web documents is through content delivery networks (CDNs). These are normally run by companies that set up a large number of servers around the world, each server dedicated to hosting Web document replicas. The owners of these documents usually have no control over the replicating servers. Thus, in the CDN scenario, clients are retrieving data from hosts that are controlled neither by them, nor by the content providers. This situation is by no means specific only to CDNs; caching presents the same problem.

Retrieving data from untrusted hosts introduces a series of security issues, which can be grouped in two categories - **document integrity** and **secure naming**. Document integrity deals with ensuring that a document retrieved by a client from an untrusted host has not been tampered with by that host. Secure naming deals with establishing secure associations between Web documents and the real-world entities in charge of them.

This paper introduces a security architecture that guarantees data integrity and secure naming for Web content even when this content is replicated on untrusted hosts. While read-only secure storage systems have already been proposed [6], the main contribution of this paper is to describe a design where data content, replication strategies and security are all integrated in one unified object model. Our design has been validated through a number of performance tests we have carried on a prototype implementation, results which are also presented in this paper.

The rest of the paper is organized as follows: in Section 2 we introduce a new Web document model with a quick discussion on how such a model can be implemented in order to use most of the current Web infrastructure. Section 3 describes our security architecture, and how this architecture tackles secure naming and document integrity issues. Section 4 describes our prototype implementation, and a number of performance tests we have performed, Section 5 discusses related work, while Section 6 points to directions for future work and concludes.

2 Globe Web Documents

For the work described in this paper we rely on the services offered by Globe [17] - a middleware architecture that allows the development of distributed applications based on replicated shared objects. One such distributed application is GlobeDoc [18]. GlobeDoc objects are the building blocks for our new Web document model.

In the GlobeDoc model of the Web, a Web site is composed of related Web documents. A Web document itself is a collection of logically related Web resources. These Web resources are referred to as a document's **page elements** and can be anything that is accessible over the Web (e.g., HTML files, text files, images, audio files, video files, applets, etc.). The relation between the resources contained in a Web document is generally stronger than that between the documents contained in a Web site. For example, an organization's Web site contains a collection of Web documents that are somehow related to that organization, while a Web document representing a news story would contain only the elements directly relating to that story (e.g., the HTML page plus any icons and images relating to the story or the page layout).

In GlobeDoc, a Web document is encapsulated in a Globe distributed shared object which contains that document's elements in its state. Such a GlobeDoc object offers methods that allow clients to access and modify its state on a per-element basis. The hyper-linked structure that is normally provided by Web pages is maintained in GlobeDoc. A relative hyper-link contained in some GlobeDoc object's element refers to another element in that same object. Likewise, an absolute hyper-link may refer to an element contained in another GlobeDoc object.

Every GlobeDoc object is identified by a unique object ID (**OID**). This is a 160-bit number which does not contain any location information and is not human readable. The GlobeDoc architecture, therefore, contains two services that facilitate locating objects. The Naming Service [3] maps human readable object names onto OIDs. The Globe Location Service [16] maps OIDs onto contact addresses, which contain information about where and how to contact a GlobeDoc object.

By virtue of it being a Globe distributed shared object, a GlobeDoc can distribute (replicate) its state over multiple physically separated address spaces (or machines). As such, requests for an object's state will be distributed over these various machines, thereby spreading the total load generated by the requests and preventing any single machine hosting the GlobeDoc from becoming overloaded. Similarly, by strategically replicating a GlobeDoc object's state so that it is close to large concentrations of clients, the traffic at each replica will have a local character, increasing responsiveness for clients and decreasing overall network traffic and saturation. This architecture is ideally suited to the creation of (peer-to-peer) content delivery networks.

Also, because Globe makes the state distribution transparent to clients and because the distribution policies can be determined independently per Globe object, GlobeDoc makes it possible to apply distribution strategies on a per-document basis. In this way GlobeDoc allows replication of Web documents without imposing any single global replication policy on all documents. [13] has shown that applying per-document distribution strategies can lead to better efficiency than the application of one-size-fits-all strategies.

2.1 GlobeDoc Services

A client accesses a GlobeDoc object using a standard Web browser. However, standard Web browsers cannot directly connect to GlobeDoc objects and invoke its methods to retrieve the page elements. For this task, browsers rely on a GlobeDoc proxy server to intercept requests to GlobeDoc objects and manage the interaction with the object (i.e., connect to the object and retrieve its elements). Because standard Web browsers do not understand GlobeDoc object names, **hybrid URLs** are used. These are just regular URLs that start with a distinguishing prefix (which in our case is *http://enter.globeworld.org*). GlobeDoc and page element names are then embedded in these hybrid URLs.

GlobeDoc proxies connect to GlobeDoc objects by binding to them. The result of binding to a GlobeDoc object is that a local representative of that object is placed in the address space of the binding process. A local representative is a local part of the GlobeDoc object. It can be a simple object proxy, forwarding method invocations to other replicas, or it can be a full replica containing a local copy of the GlobeDoc object's state. The client, however, is unaware of this and simply invokes methods from the local representative as though the object was local.

Binding itself consists of two distinct phases: (1) finding the object, and (2) installing the local representative. This is illustrated in Figure 1. Finding an object is separated into a name-lookup and a location-lookup step; installing the local representative requires that a suitable contact address and implementation be selected.

2.1.1 Naming Service

To find an object, a process must resolve the object's name to an OID. This is done by passing the object name to a naming service. The naming service returns the object's OID. Whereas an OID uniquely identifies an object (i.e., an object always has exactly one OID), an object may have multiple names that resolve to its OID. An object name is simply a human-readable string that represents an object. It is up to the naming service to interpret this name and resolve it to an OID.

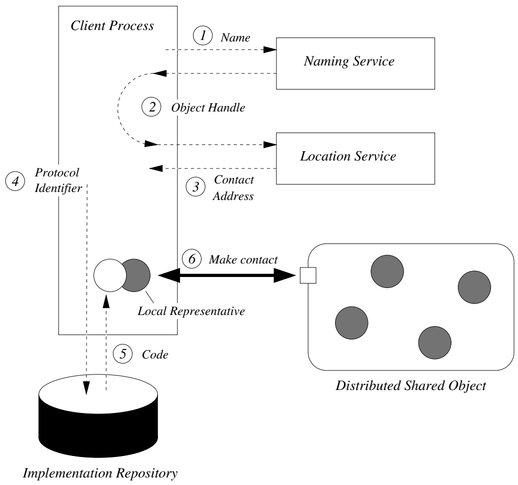

Figure 1. Binding to a GlobeDoc object

2.1.2 Globe Location Service

Resolving an object's name leads to an OID. An OID, however, does not identify the location of an object and its replicas. To find the actual object, the OID must first be passed to a location service, which returns one or several contact addresses which represent GlobeDoc object contact points. The location service is responsible for storing contact addresses and resolving OIDs to these contact addresses. Besides looking up contact addresses for objects, the location service also allows addition, deletion and modification of contact-address mappings.

The Globe Location Service is implemented as a distributed search tree. In this tree, the world is divided into a hierarchical set of domains. At the lowest level there is one domain per site; a collection of sites form a region, etc. An object is recorded at each site where it has a contact address, and recursively in each enclosing region up to the root of the tree. Initially, a record at the site level contains the actual contact addresses while records at higher levels contain pointers to the next lower level. Recording an object at multiple levels allows searches with expanding rings: a search starts at the local site, followed by the local region, then the next higher level region, etc., and eventually followed by the root.

2.1.3 Object Server

A GlobeDoc object's replicas are implemented as stateful local representatives hosted on object servers. An object server is a process that provides an **address space**, **contact points** and **runtime services** to the local representatives that it hosts. The object server also manages a local representative's access to local resources such as local disks and networking resources. Besides simply hosting LRs, a Globe object server also has a remotely accessible interface that allows other local representatives, other Globe object servers, or administrators to request services from it. These services include the creation and destruction of GlobeDoc objects and their replicas.

3 The GlobeDoc Security Architecture

In this section we will describe the mechanisms used to make GlobeDoc objects secure. We have been using well-known security and cryptographic techniques as building blocks, and combine them in an architecture that can offer secure naming and data integrity guarantees.

Following the model of the Web master for regular Web documents, we require that behind each GlobeDoc object there is a person or organization - the **object owner** - that is in charge of it. The owner is the one who creates the object and is responsible for providing permanent storage for it, updating it, and setting up the replication and security policies for the object. Furthermore, we require each GlobeDoc to have a unique public/private key pair associated with it, which is generated by the owner when the object is created. As we will describe next, the object owner uses the object's private key to sign the object's state (files) before it replicates it, while clients use the public key to perform integrity checks when retrieving parts of the object's state from untrusted hosts.

3.1 Secure Naming

3.1.1 The Problem

Secure naming deals with creating secure and trustworthy associations between Web resources and qualified names, which in turn convey information about the real world entities that are behind these resources. In the current Web, name binding is accomplished through the Domain Name System (DNS) [11], a hierarchical distributed database that translates between human-readable resource names and their IP addresses. Data integrity requirements were left out from the original DNS design, which made it vulnerable to various types of masquerading attacks. To prevent these, a secure DNS (DNSsec) framework has been proposed [5] and is currently being deployed.

All DNSsec does is establish secure associations between resource names and sets of IP addresses. However, clients cannot authenticate IP addresses, so their attempts to retrieve data from servers at these addresses are vulnerable to man-in-the-middle attacks. To prevent this, secure Web servers have public/private key pairs which they use to authenticate to clients. The server's public key needs to be certified by some Certificate Authority (a trusted third party) through a digital certificate that binds the public key to the server's name (which presumably can be traced to the qualified path name solved through DNSsec). Having this public key, securely associated to the server's name (by the CA), and to the server's IP address (by DNSsec), the client can then authenticate the server. In this way, the man-in-the-middle attack becomes impossible, and the name binding and secure channel establishment can be safely combined.

The problem with this approach is that it does not work well with dynamically replicated Web documents. Although DNS supports mirroring Web sites by allowing multiple addresses associated with the same name, the basic assumption is that mirroring is more or less static. Dynamic active replication of Web documents would put serious strain on the resource-record caching that makes DNS so efficient.

In conclusion, the approach as followed in DNSsec for secure naming of Web documents will not work well for the GlobeDoc model we are proposing. For GlobeDoc objects we need a naming mechanism that both:

- Securely associates objects with the real-world entities behind these objects.

- Supports efficient lookup operations even when objects are massively replicated and replicas' network addresses change frequently.

3.1.2 Our Solution

As described in Section 2, each GlobeDoc is identified through a unique 160-bit object ID (OID). Secure name binding requires creating a secure association between an OID, the object's public key, and the real-world entity (individual, company, organization) that is in charge of the object.

First, we will examine how to securely link the object's public key to the OID. This can be accomplished by having the OID be the 160-bit secure hash *SHA-1* [1] of the object's public key, in fact creating a **self-certifying OID** for the object. The *SHA-1* secure hash function has the property that it is computationally intractable to find two different inputs with the same hash. Therefore an OID obtained in this manner is securely linked with the public key of the object. This approach of using self-certifying OIDs is similar to the one taken by the designers of the SFS system [10], who have pioneered the concept of self-certifying resource names [9].

Now, the only thing left is to securely associate self-certifying OIDs with the real-world entities in charge with their corresponding objects. As stressed in the SDSI document [15], we believe that establishing trust in a remote entity (trusting that entity to be who it claims to be) is such a security-sensitive task that it is better to give most control over it to the users themselves. To facilitate this, GlobeDoc objects have a special security interface that can be used by clients (in fact by their proxies) to retrieve any certificates the objects can provide to prove their identity. The users themselves can specify a number of CAs they trust, and store their public keys with their user proxy. When the user requests an element part of a GlobeDoc, its proxy requests identity certificates that match the user's list. For the first match found, the proxy displays the naming information in the certificate. The user examines this information and can then make a decision how much trust to put in that object.

The CA-mediated secure name-binding mechanism we just described is appropriate when Web objects are used for highly sensitive applications such as e-commerce or e-banking. For less sensitive applications, users may find the name-binding information provided by a DNSsec-like name service sufficient. The good news is that DNSsec can be extended to support self-certifying OIDs by storing them in the resource records instead of IP-addresses [3]. The great advantage of this would be that DNS would store only **location-independent** data. The location-dependent information associated with each object (the addresses of its replicas) is retrieved in an additional step from the Globe Location Service. This two step secure name resolution allows us to overcome the problems DNS has when tracking dynamically replicated documents.

One point we want to stress here is that users do not have to trust the information stored in the Location Service. In fact, using the Location Service is not even mandatory for GlobeDoc objects; a per-object dedicated directory replica keeping track of all the other replicas' contact points would accomplish the same task. A malicious Location Service server can return false contact points to its clients, making these clients bind to replicas which are not part of the objects they want to contact. However, as we will see in the next section, clients can easily detect when the data they are retrieving is not part of the GlobeDoc object they want to browse, so the most harm a malicious Location Service server can do is a temporary denial of service.

As a conclusion, secure naming for GlobeDoc objects is achieved through self-certifying OIDs, which can be stored in DNSsec resource records. When additional security guarantees are necessary, object-provided certificates, signed by trusted CAs can also be employed. Because only location-independent information is stored, this mechanism has the two properties we required in Section 3.1.1: it securely associates Web objects to the real-world entities in charge of them, and can efficiently support massively replicated objects even when replicas network addresses change frequently.

3.2 Document Integrity

3.2.1 The Problem

Most of the data transfer in the current WWW is insecure. Clients simply connect to Web servers and request the documents stored there. This approach is clearly vulnerable to man-in-the-middle attacks, not to mention malicious caches. In such an attack scenario, an active attacker intercepts the client's request, and answers with his own document. Although such attacks are infrequent, that is probably as much due to the lack of determination on the part of the attackers as to the inherent security of the underlying network architecture. Furthermore, due to the possibility of such attacks, the security of an HTTP request is downgraded to the security of the weakest network link/router on the request path.

In the current WWW, the most common protection against such attacks is through TLS (Transport Layer Security) [4]. TLS uses public-key cryptography to authenticate servers and establish secure channels between servers and clients.

The main problem with TLS is that it requires servers to be trusted. The secure channel between the client and server does not help at all if a malicious server sends bogus data over it. For this reason, TLS allows documents to be replicated only on trusted servers, which greatly restricts the set of acceptable hosts.

Turning back to our GlobeDoc objects, we can see that a mechanism like TLS is clearly not suited for ensuring data integrity. As mentioned in Section 2, GlobeDoc objects dynamically place their replicas on (possibly) untrusted object servers close to where their client requests are coming from. Our assumption is that most of these servers are honest, but we need to consider the possibility that some of them may try to replace the documents they host with fake data. Therefore, we need a security mechanism that enforces the following three properties on the replicated state of a GlobeDoc:

- **Authenticity** - the document the client receives from a server has indeed been created by the object's owner. No attacker or malicious server should be able to pass off one of their own documents as being part of the object.

- **Freshness** - the client is guaranteed to receive the most recent version of a document part of a object. No attacker or malicious server should be able to pass off genuine but old versions of a document and convince the client they are fresh.

- **Consistency** - the client is guaranteed to receive a document, part of the object, that is consistent to what she has requested. No attacker or malicious server should be able to replace the requested document with another fresh document part of the same object.

Our aim is to come up with a security design that enforces these three properties, and at the same time is efficient and lightweight (especially on server load), so that it can be employed with every type of Web application.

3.2.2 Our Solution

As we mentioned in Section 2, a GlobeDoc consists of a number of page elements. These can be HTML source files, images, Java applets and so on. We preserve the integrity of the object's state by having an **integrity certificate** associated with the object. As shown in Figure 2, this is a digital certificate signed with the object's private key that contains a table with entries for each page element for the object. Each page element entry contains the element's name, its secure *SHA-1* hash, and a validity interval.

Integrity Certificate		
Page Element Name	SHA-1(Element)	Expiration Time
Page Element Name	SHA-1(Element)	Expiration Time
Page Element Name	SHA-1(Element)	Expiration Time
⋮	⋮	⋮
Signature		

Figure 2. Integrity certificate for a GlobeDoc object

Every server that hosts GlobeDoc replicas is required to store all of the object's page elements **and** the object's integrity certificate. As we mentioned in Section 3.2.1, page elements retrieved from untrusted servers should be authentic, fresh, and consistent with the user request. The integrity certificate allows the user to check for these properties in the following steps.

- Using the object's public key, the client verifies that the signature on the integrity certificate has been generated using the object's private key (authenticity).

- The client checks the "element name" field in the certificate to ensure it is the same as the element name she has requested (consistency).

- The client computes the *SHA-1* hash of the page element and makes sure it is the same as the one in the certificate (authenticity).

- The client checks the time of retrieval to ensure it falls in the validity interval specified in the certificate (freshness).

3.3 Putting the Pieces Together

The security architecture for GlobeDoc objects results from combining the various security mechanisms described so far. The key advantage of this architecture is providing integrity guarantees for end-users, even when the Web content they are browsing is replicated on untrusted servers. Figure 3 shows how secure browsing through GlobeDoc objects works.

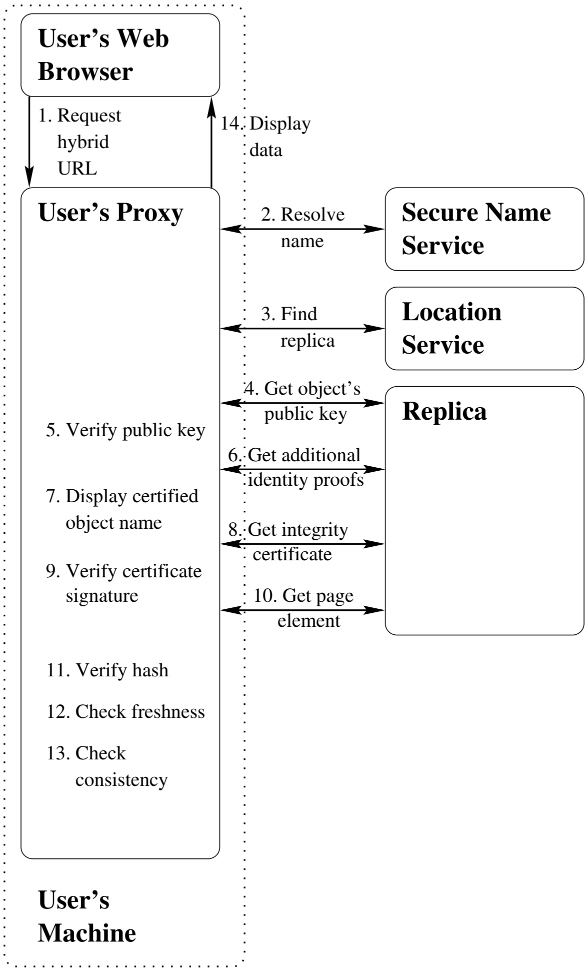

Figure 3. Secure Web browsing through GlobeDoc objects

The user starts with a hybrid URL (as described in Section 2), with an embedded object and page element name. The user can either directly type the hybrid URL in the Web browser, or get to it through an external link in some other GlobeDoc object. When the user's proxy receives this URL, it will resolve the object name by contacting the secure naming service, thus obtaining a self-certifying OID. The proxy then queries the Location Service for that OID and finds the closest object replica. The Location Service is not trusted, so there is a chance that the address it returns may point to a replica which is not part of the object. However, this can cause at most denial of service for the user, since she can always check the authenticity of the data retrieved from the replica. Our assumption is that the Location Service provides accurate information most of the time; if attackers are able to corrupt some of the LS's servers, this can be easily detected, and appropriate measures (rebooting servers, restoring the original data content from backups, etc.) can be taken.

Once the user is connected to a GlobeDoc object replica, the proxy first retrieves the object's public key, takes its *SHA-1* hash and makes sure it matches the self-certifying OID. As an extra security check, the proxy can request some additional identity proof from the object, in the form of a name certificate signed by one of the CAs trusted by the user. If such a certificate is found, the naming information in the certificate is displayed to the user in a *"Certified as:"* window. Next, the proxy requests the object's integrity certificate, and verifies its signature using the object's public key. At this point, the secure binding between the proxy and the object is complete; the proxy now requests the page element specified in the URL; once it has received it, it performs the authenticity, consistency and freshness tests discussed in the previous section. If all these tests are successful, the page element is sent to be displayed in the user's browser, otherwise a "Security Check Failed" HTML document is generated.

4 Performance Evaluation

Our implementation consists of two components: the object server and the client proxy, which are both implemented in Java, using JDK 1.3 with the standard crypto libraries.

The object server provides all the runtime services needed for creating and running GlobeDoc object replicas. These services are offered through a command interface which can be remotely accessed through a secure TLS connection, in which case TLS also takes care of the remote entity authentication. We support a simple but effective access control model: the server administrator sets up a Java keystore listing the public keys for all entities allowed to create GlobeDoc replicas on the server; such entities can be either GlobeDoc owners (individuals) or other GlobeDoc object servers (in this way we can support dynamic replication algorithms). Each entity is then allowed to manage only the replicas it creates (this includes replica destruction). We are currently working on integrating more complex policies - allowing dynamic resource negotiation, and delegation mechanisms - but this is outside the scope of this paper.

The client proxy needs to be installed by every client that wants to access secure GlobeDoc objects; the proxy identi-

Host	Architecture	RAM	OS	Java
ginger.cs.vu.nl, VU, Amsterdam "primary"	Dual Pentium III, 2×1 GHz	2 GB	Linux 2.4.9	Sun JDK 1.4
sporty.cs.vu.nl, VU, Amsterdam "secondary"	Dual Pentium III, 2×1 GHz	2 GB	Linux 2.4.9	Sun JDK 1.4
canardo.inria.fr, Inria, Paris	Pentium III 1 GHz	256 MB	Linux 2.4.7	Sun JDK 1.4.2
ensamble02. cornell.edu, Cornell, Ithaca, NY	Ultra SPARC-IIi 450 MHz	256 MB	SunOS 5.8	Sun JDK 1.3

Table 1. Experimental setting

fies GlobeDoc names from the hybrid URLs passed by the client browser, does name resolution and replica location, retrieves the desired page elements and performs the authenticity, freshness and consistency tests discussed earlier. The proxy also transparently handles any regular HTTP requests it receives from the browser.

For our experiments we have used four hosts: two in Amsterdam ("primary" and "secondary"), one in Paris, and one in Ithaca, NY. Table 4 summarizes the system configuration for each of these hosts.

In the first experiment we used four GlobeDoc objects, each consisting of one page element (image), of sizes 1KB, 10KB, 100KB, 300KB, 600KB, and 1MB respectively. For each of these objects, we placed one replica on an object server running on the Amsterdam "primary" host. Each of the objects was then accessed from the Amsterdam "secondary", Paris and Ithaca hosts, using the *wget* HTTP client. On each of these hosts, *wget* was configured to use the locally installed secure GlobeDoc proxy. The aim of the experiment was to measure the security overhead; in order to do this, we placed timers in various parts of the proxy and server code, and measured, for each object access, the amount of time dedicated to security-specific operations (these are: retrieving the object's public key, verifying its *SHA-1* hash matches the object Id, retrieving the object certificate and verifying it, computing the hash of the page element and verifying it against the hash in the certificate). Figure 4 shows the results of this experiment (these are average values computed after running the experiment for 24 consecutive hours at 6 minutes intervals).

As we can see, the overhead for transferring small page elements is significant (around 25%), and this is due to the fact that our proxy first needs to retrieve the GlobeDoc object public key and the object certificate (about 2KB of extra information). However, for larger data transfers, this initial security exchange becomes less significant, and the overhead is mainly dictated by the time needed to compute the hash over the actual data (which is proportional to the data size, but independent of the network placement of the replica). It is also worth noticing that for large data transfers, the security overhead is worse when the proxy and the

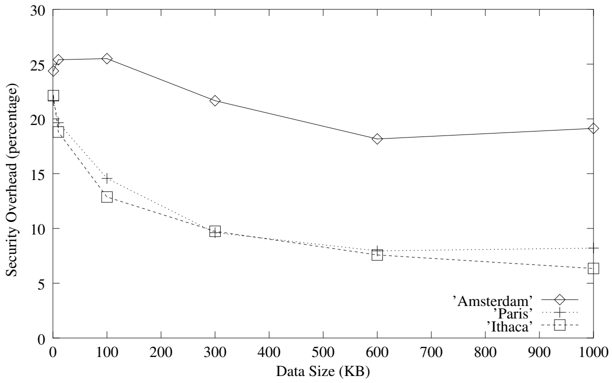

Figure 4. Security overhead

object replica are on the same LAN (the Amsterdam primary - Amsterdam secondary case), because in this case the network transfer time is so small that the time to compute the hash dominates. For the actual deployment of the GlobeDoc architecture, we expect the most common case to be the one in the Paris-Amsterdam setting (client and replica in relatively close network proximity, although not on the same LAN). In this case, we can see that the security overhead drops quite rapidly for larger data transfers. We achieve even lower overhead when replicas and clients are placed on different continents (the Amsterdam-NY setting), in which case the transfer time is so large that adding a bit of computation hardly matters. However, this latter setting is less realistic for the architecture we advocate - the whole point with GlobeDoc objects is to place replicas in close proximity of the clients.

In the second experiment we used three GlobeDoc objects, each consisting of 11 page elements. One of the page elements was always a 5KB text file. The other 10 elements are images, and are of size 1KB each for the first object, 10KB each for the second, and 100KB each for the third object. Thus, the total size for the first object is 15KB, for the second 105KB, and for the third 1005KB. For each of these objects, we place one replica on an object server running on the Amsterdam "primary" host. We also place the same page elements (as text and image files) in a directory accessible by an Apache web server, also running on the Amsterdam primary host.

The aim of the experiment was to compare the performance of the GlobeDoc server/proxy combination to the Apache server. For this, we compared the time needed to access the four GlobeDoc objects using *wget* and the GlobeDoc secure proxy with the time needed to download (again using *wget*) the same files from the Apache server through standard HTTP and secure (SSL) HTTP connections. Figures 5, 6, and 7 show the average timing computed after running this experiment for 24 consecutive hours at 15 minutes intervals.

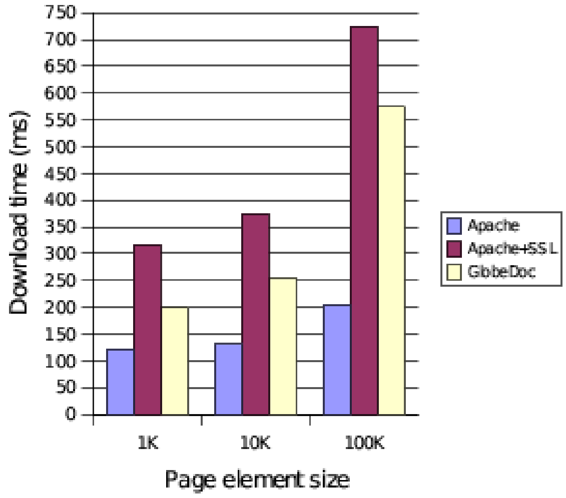

Figure 5. Performance comparison - Amsterdam client

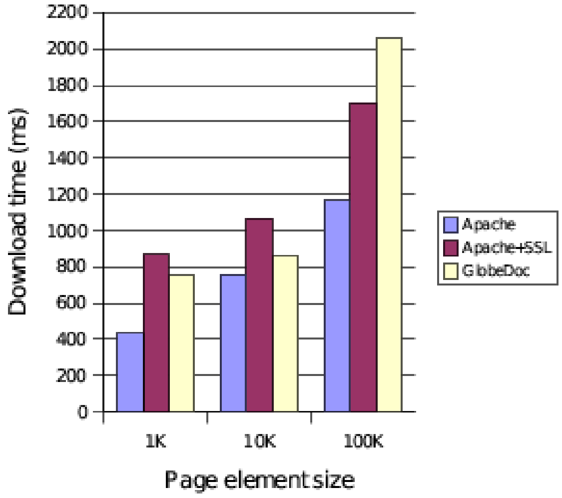

Figure 6. Performance comparison - Paris client

We can see that although running as interpreted Java code, our proxy/object server combination performs quite similar to the compiled C Apache code. On the disappointing side, it should be noticed that our implementation does not always outperform the Apache/SSL combination,

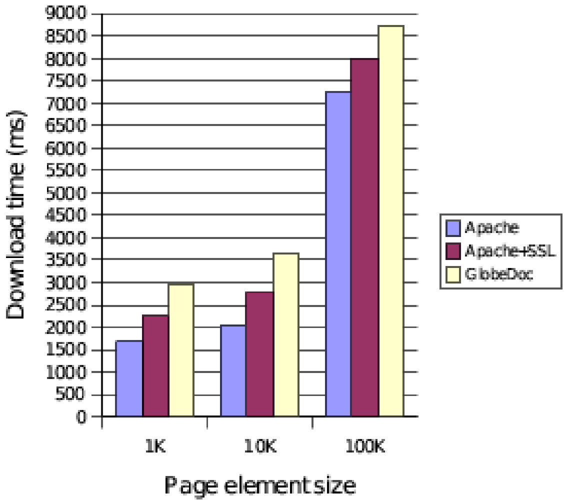

Figure 7. Performance comparison - Ithaca client

which we would have expected considering that GlobeDoc requires only public key signature verification operations which are much faster than the public key encrypt/decrypt operations required by SSL. The reason for this is the Java memory management mechanism combined with the rather limited memory resources on the Paris and Ithaca hosts; the result is extensive memory swapping which greatly reduces performance. In the case of the Amsterdam client, the large amount of physical memory reduces the need for memory swapping.

Despite these shortcoming, we believe that our proof-of-concept prototype has served its purpose, and has demonstrated that our design is viable. Furthermore, in the worst-case performance scenarios, the determining factors are the specifics of the Java environment - particularly its memory allocation mechanism - as opposed to performance limitations introduced by our design. For this work, we have used Java because it is ideal for quick prototyping, but we believe that implementing the same functionality as compiled C code - as a "GlobeDoc" Apache module for example - would greatly improve performance.

It is also important to stress that the biggest advantage of the GlobeDoc architecture is the ability to place object replicas in close proximity to clients, which greatly reduces the client-perceived latency. However, the aim of the experimental work in this paper is to prove that the security mechanisms we have devised for GlobeDoc do not introduce excessive overhead, which we believe we have demonstrated. By no means are we trying to quantify the performance improvement introduced by actually replicating Web content by means of the GlobeDoc architecture; this is work has already been done, and the results were presented in [13].

5 Related Work

The WWW has experienced explosive growth, with an increasing number of security-sensitive applications making now use of it. Not surprisingly, the issue of "making the Web more secure" has attracted considerable attention, with multiple solutions proposed for a number of specific application scenarios. What makes the GlobeDoc approach distinctive is the fact that it considers not only security but other aspects, such as data content aggregation and replication, creating a unified Web document object model.

The current "state of the art" for secure Web browsing is the combination of DNSsec [5], certified Web servers' public keys and TLS [4]. In Section 3 we have described how these mechanisms can be combined in order to achieve secure naming, server authentication and integrity guarantees for the data received by the clients. However, this approach does not support document replication on untrusted hosts. Furthermore, since DNSsec records are used to store location-dependent information (IP addresses), this approach does not scale well for dynamically-replicated documents.

Along the same lines is the SHTTP proposal [14], which extends the HTTP protocol with a number of security extensions. The aim is to allow end-to-end integrity checks, as well as data confidentiality and traceability for HTTP transactions. This proposal explicitly mentions secure data replication on untrusted hosts as a potential benefit, which is also one of the motivations for GlobeDoc. However, SHTTP comes as an Internet RFC, which means it is only a set of guidelines for system developers. GlobeDoc is an actual system architecture with a running prototype implementation. Furthermore, GlobeDoc is broader in its scope, also dealing with data replication aspects (replica hosting, replication algorithms, consistency models), which are not covered by SHTTP.

Like GlobeDoc, the read-only Secure File System (r-oSFS) [6] focuses on securely replicating data on untrusted servers. The basic architectural element for r-oSFS is the file system; because of this it is possible to use r-oSFS as a middleware and implement various distributed applications on top of it, a secure Web infrastructure being one such possible application [7]. In order to guarantee the integrity of data replicated on untrusted hosts, r-oSFS constructs a hash tree by applying a secure hash function (*SHA-1*) on the data blocks and i-nodes of the file system. This approach is very efficient, since only the root of the tree needs to be signed by the owner, but has the drawback that only one global (per-file system) consistency interval can be supported, instead of allowing per-file freshness constraints.

Although r-oSFS file systems can be replicated on untrusted hosts, there is little support for the actual replication. In contrast, each GlobeDoc comes with its own replication policy which is **part of** the object itself; this allows for very fine-grained (per page-element) replication policies to be defined, which has been proven [13] to greatly improve performance. Following the same logic, the GlobeDoc security architecture uses per page-element expiration dates, which allow owners to set per page-element freshness constraints (which is not possible with r-oSFS).

The Gemini project [12] at Carnegie Mellon University aims at creating a publisher-centric replication infrastructure for information dissemination. Gemini is built as an extension to the Squid [2] WWW cache, and - like secure GlobeDoc - aims at providing integrity guarantees for Web documents replicated on untrusted hosts. However, Gemini focuses more on dynamic Web data, and their security solution is different from the one we advocate in this paper - they require the untrusted caches to sign the data they return to clients, which ensures that malicious caches serving bogus content are eventually caught "red-handed". On the other hand, GlobeDoc makes impossible for malicious servers to pass bogus data undetected. Furthermore, GlobeDoc provides a unified Web document object model, with explicit support for replication, features which are not present in Gemini.

Finally, OceanStore [8] is a project that aims at using untrusted hosts to provide a "*utility infrastructure designed to span the globe and provide continuous access to persistent information.*" To accomplish this ambitious goal, the designers of the system make use of peer-to-peer technologies, such as associative storage, distributed routing, and probabilistic query algorithms. Although both make use of untrusted storage, OceanStore and GlobeDoc have slightly different goals: OceanStore assumes that all the storage is untrusted, and focuses on high data redundancy to prevent loss due to malicious hosts or catastrophic events. GlobeDoc on the other hand assumes that each document has access to some secure storage provided by its owner (the traditional Web document model), and relies on untrusted hosts for replication in order to improve performance. Although we recognize the many revolutionary ideas OceanStore introduces, we believe that the GlobeDoc Web document model is more appropriate for the next generation of secure WWW services.

6 Conclusion

In this paper we have presented a security architecture that integrates data content, replication mechanisms and security policies in one unified object model, and guarantees data integrity and secure naming even when object replicas are placed on untrusted servers. Our experimental results show that this new object model can be efficiently integrated into the current Web infrastructure, and the security mechanisms do not incur excessive performance penalty.

One direction for future work is support for more complex data hosting negotiation mechanisms. We are working on the design of a policy language that would allow object owners to express quality of service requirements before instantiating new object replicas. At the same time server administrators will be able to specify resource limitations (in terms of disk space, memory, network bandwidth among other things) for the replicas they are willing to host, with the object server being responsible with enforcing these limitations.

Another direction for future work is GlobeDoc support for dynamic Web content. We believe this will be particularly challenging because the technique we currently apply to secure static documents - signing them with the object key - does not work in the case of dynamic data - it would require the object owner to sign the results for every possible client query, which is clearly not feasible. In such a setting, a solution based on auditing the untrusted servers where the data is replicated, as suggested in [12], combined with a probabilistic double-checking of the dynamic Web content these untrusted servers generate is likely to be more effective.

References

[1] Secure Hash Standard. FIPS 180-1, Secure Hash Standard, NIST, US Dept. of Commerce, Washington D. C. April 1995.

[2] Squid web proxy cache. http://www.squid-cache.org.

[3] G. Ballintijn, M. van Steen, and A. Tanenbaum. Scalable User-Friendly Resource Names. *IEEE Internet Computing*, 5(5):20–27, 2001.

[4] T. Dierks and C. Allen. The TLS Protocol Version 1.0,. IETF RFC 2246, January 1999.

[5] D. Eastlake. Domain name system security extensions. RFC 2535, March 1999.

[6] K. Fu, M. Kaashoek, and D. Mazieres. Fast and Secure Distributed Read-only File System. In *Proc. 4th USENIX Symp. on Operating Systems Design and Implementation*, pages 181–196, San Diego, CA., Oct. 2000.

[7] M. Kaminsky and E. Banks. SFS-HTTP: Securing the web with self-certifying URLs. citeseer.nj.nec.com/470041.html.

[8] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. OceanStore: An Architecture for Global-scale Persistent Storage. In *Proc. 9th ACM ASPLOS*, pages 190–201, Cambridge, MA, November 2000. ACM.

[9] D. Mazieres and M. F. Kaashoek. Escaping the Evils of Centralized Control with Self-Certifying Pathnames. In *Proc. 8th ACM SIGOPS European Workshop*, Sintra, Portugal, Sept. 1998.

[10] D. Mazieres, M. Kaminsky, M. F. Kaashoek, and E. Witchel. Separating Key Management from File System Security. In *Proc. 17th Symp. on Operating Systems Principles*, pages 124–139, Kiawah Island, SC, 1999.

[11] P. V. Mockapetris and K. J. Dunlap. Development of the domain name system. In *SIGCOMM*, pages 123–133, 1988.

[12] A. Myers, J. Chuang, U. Hengartner, Y. Xie, W. Zhuang, and H. Zhang. A Secure, Publisher-Centric Web Caching Infrastructure. In *Proc. 20th IEEE Infocom*, Anchorage, AK., April 2001.

[13] G. Pierre, M. van Steen, and A. Tanenbaum. Dynamically Selecting Optimal Distribution Strategies for Web Documents. *IEEE Transactions on Computers*, 51(6), 2002.

[14] E. Rescorla and A. Schiffman. The Secure HyperText Transfer Protocol. IETF RFC 2260, August 1999.

[15] R. L. Rivest and B. Lampson. SDSI – A Simple Distributed Security Infrastructure. Presented at CRYPTO'96 Rumpsession, 1996.

[16] M. van Steen, F. Hauck, P. Homburg, and A. Tanenbaum. Locating Objects in Wide-Area Systems. *IEEE Commun. Mag.*, pages 104–109, January 1998.

[17] M. van Steen, P. Homburg, and A. Tanenbaum. Globe: A Wide-Area Distributed System. *IEEE Concurrency*, pages 70–78, January-March 1999.

[18] M. van Steen, A. Tanenbaum, I. Kuz, and H. Sips. A Scalable Middleware Solution for Advanced Wide-Area Web Services. *Distributed Systems Engineering*, 6(1):34–42, March 1999.