A content integrity service for digital repositories

Stuart Haber, Pandurang Kamat, Kiran Kamineni

HP Laboratories
HPL-2008-177

Keyword(s):
integrity, longevity, archive, repository

Abstract:

We present a "content integrity service" for long-lived digital documents, especially for objects stored in digital repositories. The goal of the service is to demonstrate that information in the repository is authentic and has not been unintentionally or maliciously altered, even after its bit representation in the repository has undergone one or more transformations. We describe our design for an efficient, secure service that achieves this, and our implementations of two prototypes of such a service that we developed, most recently for DSpace. Our solution relies on one-way hashing and digital time-stamping procedures. Our service applies not only to transformations to archival content such as format changes, but also to the introduction of new cryptographic primitives, such as the new one-way hash function family that will be chosen by NIST in the competition that was recently announced [10]. In the face of recent attacks on hash functions, this feature is absolutely necessary to the design of an integrity-preserving system that is meant to endure for decades.

External Posting Date: October 21, 2008 [Fulltext] Approved for External Publication
Internal Posting Date: October 21, 2008 [Fulltext]

Published and presented at OR 2008, 3rd International Conference on Open Repositories, Southampton, UK, April 2008.

A content integrity service for digital repositories

Paper supporting a presentation at the general session of Open Repositories 2008, Southampton, UK

Stuart Haber
Hewlett-Packard Labs
stuart.haber@hp.com

Pandurang Kamat
WINLAB, Rutgers University
pkamat@winlab.rutgers.edu

Kiran Kamineni
University of Illinois at Chicago
kiran.kamineni@gmail.com

April 2008

Abstract

We present a "content integrity service" for long-lived digital documents, especially for objects stored in digital repositories. The goal of the service is to demonstrate that information in the repository is authentic and has not been unintentionally or maliciously altered, even after its bit representation in the repository has undergone one or more transformations. We describe our design for an efficient, secure service that achieves this, and our implementations of two prototypes of such a service that we developed, most recently for DSpace. Our solution relies on one-way hashing and digital time-stamping procedures.

Our service applies not only to transformations to archival content such as format changes, but also to the introduction of new cryptographic primitives, such as the new one-way hash function family that will be chosen by NIST in the competition that was recently announced [10]. In the face of recent attacks on hash functions, this feature is absolutely necessary to the design of an integrity-preserving system that is meant to endure for decades.

1 Introduction

Information in a digital repository can include complex multi-part documents. In a long-term repository these documents may be expected to undergo multiple transformations during their lifetime, including, for example, format changes, and modifications to sub-parts and to accompanying metadata. Skeptical users of a digital repository may desire, or in some cases may be legally required, to verify the integrity of records that they have retrieved from the repository.

All typical algorithmic techniques for verifying the integrity of a digital object begin with a representation of the object in question as a sequence of bits. When digital objects are transformed in any nontrivial way, their bit representations are changed as well, so that these algorithmic techniques no longer apply to the transformed object. In fact, it is the usual aim of a cryptographic technique for proving integrity that it "fail"—more precisely, that it correctly succeed in proving lack of integrity—when even a single bit in the object's representation is changed.

We have designed an efficient and secure Content Integrity Service (CIS) that solves this problem [5], and we have implemented it as a service for two different platforms:

- HP's Digital Media Platform, a service-oriented architecture for content processing and storage [3];
- the DSpace open-source repository platform [11, 4].

We contributed our Dspace implementation to the DSpace SourceForge repository [12]. We invite administrators of DSpace to use it immediately, and implementors of other repository systems to adapt it to their own systems.

The essence of our solution is to use a secure digital time-stamping system, first to time-stamp every document at ingestion into the archive, storing the resulting time-stamp certificate in the archive with the document; and second to produce an auditable record of every transformation to a document in the archive, in such a way as to verifiably link the time-stamp certificate for the transformed version of the document to its original form. This is described in greater detail in §3.2.

Our solution does not require public-key techniques at all. As we explain in §3.1 below, current public-key implementations are inappropriate for use in long-lived repository systems.

2 Implementations

2.1 HP Digital Media Platform

Our first implementation of CIS was built in 2004 as a service for a prototype of HP's Digital Media Platform (DMP), a service-oriented architecture for content processing and storage [3, 5].

The repository data model of DMP is a directed graph with *nodes*, *resources*, and *literals* linked by labelled edges called *properties*, similar to the architecture of (*e.g.*) the JSR 170 Content Repository for Java [8]. All complex documents in the repository are represented using this model. Each subcomponent of a multi-part document is represented as a resource. Nodes are used to represent hierarchy within the document, and literals are strings that may represent metadata. Properties belong to nodes and link them to other nodes, resources or literals. The nodes and resources are named using Uniform Resource Identifiers (URIs).

2.2 DSpace

In 2007 we implemented CIS for the DSpace open-source repository platform [11, 4]. In DSpace, complex documents are called "items". CIS is integrated into DSpace's natural workflow so that CIS certificates are automatically created when an item is ingested into the repository, and when an item is updated in any way.

We implemented a simple versioning system for DSpace items in order to handle updates.[1] Users of the DSpace repository can make a "verify" call in order to check the validity of any version of a stored item.

In tests of our system, CIS added no noticeable slowdown to the ordinary operations of DSpace.

We give some further details of our implementations in §3.3 below, after explaining our algorithm.

[1] This is separate from the 2007 Google Summer of Code project to implement a versioning system for DSpace.

3 How it works

3.1 Background

The basic building blocks of our solution are cryptographic hash functions and time-stamping procedures, which we briefly explain here. Throughout this article we refer to the objects of concern in a digital archive or repository simply as "documents".

A *cryptographic hash function* or *one-way hash function* is a fast procedure H that compresses input bit-strings of arbitrary length to output bit-strings (called *hash values*) of a fixed length, in such a way that it is computationally infeasible to find two different inputs that produce the same hash value. (Such a pair of inputs is called a *collision* for H.) For any digital document x, its hash value $v = H(x)$ can be used as a proxy for x, as if it were a characteristic "fingerprint" for x, in procedures for guaranteeing the bit-by-bit integrity of x [9, Chap. 9].

A *digital time-stamping* scheme is a procedure that solves the following problem: given a digital document x at a specific time t, produce a *time-stamp certificate* c that anyone can later use (along with x itself) to verify that x existed at time t. Certificates that will pass the verification test should be difficult to forge. There are two different families of time-stamping algorithms, those using digital signatures and those based entirely on hash functions [6].

In what is sometimes called a *hash-and-sign* time-stamping scheme, the time-stamp certificate for a document consists of a digital signature computed by a Time-Stamping Authority (TSA) for the document and the time of signing. This has two major drawbacks as a tool for long-term archives: (1) It requires the assured existence of trustworthy key-validity history data, in order to check the validity of the TSA's public verification key. It is a problem for any TSA to manage such a key-validity database over extended periods of time, let alone integrating it with currently deployed commercial PKIs (public-key infrastructures). To be charitable, not many current PKI systems make a serious effort to build this component with an eye towards long-lasting secure use of the system (and some of them completely ignore it). (2) The trustworthiness of the certificate depends entirely on an assurance that the TSA's private signing key has never been compromised. This is an unacceptable premise for a long-term archive.

For the CIS, we chose a different time-stamping technique called *hash-linking*. In this technique, the hash value of the document to be time-stamped is combined with other hash-values received during the same time period to create a witness hash value. This value is then published by the TSA as a widely witnessed event. Hash-linking makes it computationally infeasible for an adversary to back-date a document, since that would entail computing hash collisions for the witness values or for their hash-function preimages. This technique relies only on the collision-resistance properties of hash functions, and does not have any secrets or keys that need to be securely protected over extended periods of time [7, 1, 2].

3.2 Our solution

Before describing our general solution, we first discuss the special case of a particular kind of transformation to a document, namely a method for updating its integrity certificate.

3.2.1 Renewing integrity certificates

Here we describe the process of "renewing" digital time-stamps [1]. The need for this is motivated by the fact that, with advances in computational power and resources, as well as the discovery of entirely new cryptanalytic algorithms, particular instances of cryptographic primitives that were secure when they were first deployed may become insecure several years later. In the last three years, the cryptographic community has been surprised by powerful new attacks on most of the widely used hash functions [15, 14]. In response, NIST has

recently announced an international competition for the design of a new family of one-way hash functions, similar to the process that led to the adoption of the Advanced Encryption Standard (AES) [10].

Suppose that an implementation of a particular time-stamping system is in place, and consider the pair (d, c_1), where c_1 is a valid time-stamp certificate (in this implementation) for a particular document d. Now suppose that some time later an improved time-stamping system is implemented and deployed—by replacing the hash function used in the original system with a new hash function, or even perhaps after the invention of a completely new algorithm. Is there any way to use the new time-stamping system to buttress the guarantee of integrity supplied by the certificate c_1 in the face of potential later attacks on the old system?

One could simply submit d as a request to the new time-stamping system; but this would lose the connection to the original time of certification. Another possibility is to submit c_1 as a request to the the new time-stamping system. But that would be vulnerable to the later existence of a devastating attack on the hash function used in the computation of c_1, as follows: if an adversary could find another document d' with the same hash value as d, then he could use this renewal system to back-date d' to the original time.

Suppose instead that the pair (d, c_1) is time-stamped by the new system, resulting in a new certificate c_2, and that some time after this is done (i.e. at a definite later date), the original method is compromised. The certificate c_2 provides evidence not only that the document contents d existed prior to the time of the new time-stamp, but that it existed at the time stated in the original certificate, c_1; prior to the compromise of the old implementation, the only way to create a valid time-stamp certificate was by legitimate means.

Observe that the security offered by an "updated" time-stamp certificate computed as above depends on the arguably questionable assumption that the first time-stamping system will not be compromised until a definite time after the second system was launched. But in practice, this is not an unreasonable assumption. Advances in cryptanalytic attacks on hash functions typically proceed incrementally (as in the case of the attacks cited above [15, 14]), and well before a hash function is completely broken, fielded systems can swap in a new hash function.

3.2.2 Auditable transformation history

Next we show how a similar approach can handle our general problem. For simplicity, we begin with the case where we are only interested in enabling the authentication of an entire document (as opposed to making this possible as well for parts of the document). In its original form, let d denote the bit-string representation of the document in file format f, and let us suppose that d is time-stamped at time t, with resulting time-stamp certificate c. We will write $c = TS(d; t)$ to indicate that the certificate is for input consisting of the document d, and it was computed at time t.

Now suppose that at some later time t', it is decided to make a format change to format f', using a particular conversion or migration procedure. Let d' denote the bit-string representation of the resulting document. Simply computing a new time-stamp certificate for d' would lose the connection between d', the new representation of the document, and its original version. The aim rather is to memorialize—and enable later verification of—this format conversion, while preserving the assurance of integrity all the way back to that of the original form of the document. We can do this by adapting the procedure for renewing time-stamps described above. Let i denote a standard way of describing an invocation of the migration procedure used to convert from format f to format f', perhaps including file-names and other useful meta-data for input and output files d and d', respectively. Then, immediately after performing the conversion, a new time-stamp request for $[d, d', i, c]$ is submitted, and the resulting time-stamp certificate $c' = TS(d, d', i, c; t')$ is stored with the document in the

archive. The new certificate c' can be used to verify the integrity of the latest form of the original document.

Assuming the integrity of i as a description of the transformation and the security of the time-stamping system we use, the only way to compromise the security of our solution is to compute collisions for the hash functions used.

Variations on the technique just described can be applied to complex transformations to one or more pieces of a multi-part document, including format conversions, annotations, additions of metadata, and later modifications of the document; these modifications can include the steps of a business workflow, or document redaction, for example. Naturally, transformations can follow one another, and each one can be certified by a CIS certificate.

As far as we are aware, this problem was never seriously addressed before our first publication on the subject [5], perhaps because researchers did not realize that the problem has a simple, secure solution.

3.3 Implementation details

Both of our implementations were designed so that they can use any third-party time-stamping system at all. We chose to use the hash-linking time-stamping service provided by Surety, LLC [13].

We handled the job of naming transformations in two different ways in our two implementations. In HP's Digital Media Platform (DMP), transformations on content are represented as workflow descriptions expressed as XML documents, and our implementation used this XML representation of a transformation as its invocation i. In our DSpace implementation, transformations are updates to items, which can only be made by repository administrators, and we record the administrator's free-text description of an update as its invocation i.

As mentioned in §2.1 above, documents in DMP are represented as directed graphs consisting of vertices called *nodes*, *resources*, and *literals*, linked by labelled edges called *properties*. An entire document is identified by the URI of its root node. To certify a multi-part document, the service computes a hash value for the entire document graph. Specifically, our algorithm hashes the graph in a lexicographically ordered depth-first traversal, beginning at the root, to recursively hash all the nodes, resources and literals that form the graph. At each level, intermediate hash values capture the bit-string representation of the resources and literals as well as the structure of the graph itself. The algorithm avoids any cycles during its traversal of the graph. The document's hash value is then sent as a time-stamp request to a TSA. The hash value and the resulting time-stamp certificate are stored in the repository, linked to the document graph.

For both of our implementations, when we modified content or certificates in the repository (by circumventing the system's usual permissions), verifications of CIS certificates failed, as they should do.

4 Future work

We look forward to working with the community in order to make it possible for all content in institutional repositories to be accompanied by cryptographically strong proofs of integrity.

In particular, we plan to extend the functionality of our current DSpace implementation:

- to enable scripted invocation of certificate creation and verification;
- to merge it with any independently contributed versioning system;
- to certify parts of a complex document, using the certificate of the main document;
- to allow stand-alone verification of documents exported from DSpace;
- to renew integrity certificates for documents.

References

[1] Dave Bayer, Stuart Haber, and W. Scott Stornetta. Improving the efficiency and reliability of digital time-stamping. In R.M. Capocelli, A. De Santis, and U. Vaccaro, editors, *Sequences II: Methods in Communication, Security, and Computer Science*, pages 329–334. Springer-Verlag, 1993. (Proceedings of the *Sequences* Workshop, Positano, Italy, 1991).

[2] Josh Benaloh and Michael de Mare. Efficient broadcast time-stamping. Technical Report TR-MCS-91-1, Clarkson University Department of Mathematics and Computer Science, 1991.

[3] HP Digital Media Platform: A technology platform to enable the transition to digital processes and workflows (White paper). Available at `www.hp.com/go/dmp`.

[4] `www.dspace.org`.

[5] Stuart Haber and Pandurang Kamat. A content integrity service for long-term digital archives. In *Proceedings of Archiving 2006*, pages 159–164. Society for Imaging Science & Technology, 2006. Available as HP Labs Technical Report HPL-2006-54, at `www.hpl.hp.com/techreports/2006/HPL-2006-54.html`.

[6] Stuart Haber and Henri Massias. Time-stamping. In H. C.A. van Tilborg, editor, *Encyclopedia of Cryptography and Security*. Springer, 2005.

[7] Stuart Haber and W. Scott Stornetta. How to time-stamp a digital document. *Journal of Cryptology*, 3(2):99–111, 1991.

[8] Content repository API for Java technology specification: Java specification request 170, version 1.0, 11 May 2005. `http://jcp.org/en/jsr/detail?id=170`.

[9] Alfred Menezes, Paul van Oorschot, and Scott Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[10] NIST. Announcing request for candidate algorithm nominations for a new cryptographic hash algorithm (SHA-3) family. U.S. **Federal Register** Notice, Vol. 72, No. 212, November 2, 2007. Available at `http://www.csrc.nist.gov/groups/ST/hash/sha-3/`.

[11] MacKenzie Smith, Mary Barton, Margret Branschofsky, Greg McClellan, Julie Harford Walker, Mick Bass, Dave Stuve, and Robert Tansley. DSpace: An open source dynamic digital repository. *D-Lib Magazine*, 9(1), January 2003. `doi:10.1045/january2003-smith`.

[12] SourceForge.net: DSpace. `http://sourceforge.net/projects/dspace/`.

[13] `www.surety.com`.

[14] X. Wang, Y.L. Yin, and H. Yu. Finding collisions in the full SHA-1. In Victor Shoup, editor, *Advances in Cryptology — CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*. Springer-Verlag, 200.

[15] X. Wang and H. Yu. How to break MD5 and other hash functions. In R. Cramer, editor, *Advances in Cryptology — EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*. Springer-Verlag, 200.