Long-lived digital integrity using short-lived hash functions

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

May 12, 2006

Abstract

New collision-finding attacks on widely used cryptographic hash functions raise questions about systems that depend on certain properties of these functions for their security. Even after new and presumably better hash functions are deployed, users may have digital signatures and digital time-stamp certificates that were computed with recently deprecated hash functions. Is there any way to use a new and currently unassailable hash function to buttress the security of an old signature or time-stamp certificate?

The main purpose of this note is to remind the technical community of a simple solution to this problem that was published more than a decade ago.

1 Introduction

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 couple of years, the cryptographic community has been surprised by powerful new attacks on the hash functions MD5 and SHA-1, among others [6, 5]. This raises the question of how best to introduce a new and presumably more secure hash function into a system that now uses an older hash-function design that may soon be subject to devastating compromise. In particular, what can be done with digital signatures and time-stamp certificates that were computed using the original system's hash function? This is no longer the merely academic question it was when it was first raised by the authors of [3], who proposed an incorrect solution, and then correctly solved by [1].

2 Renewing integrity certificates

2.1 Time-stamp certificates

Here we describe the process of "renewing" digital time-stamp certificates, as presented by [1].

Suppose that an implementation of a particular time-stamping system is in place, and consider the pair (x, c_1), where c_1 is a valid time-stamp certificate (in this implementation) for the bit-string x. 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 x 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 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 x' with the same hash value as x, then he could use this renewal system to back-date x' to the original time. (In fact, resubmission of c_1 was erroneously suggested by the authors of [3] as a solution to this problem.)

Suppose instead that the pair (x, 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 content x 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.

2.2 Digital signatures

Similar logic applies in the case of digital signatures. Let s be a digital signature for the document x, to be verified with respect to a particular public key, perhaps as part of a particular PKI.

The PKI adds an extra complication. Specifically, let V denote the extra data—public-key certificates, CRLs, signed statements by trusted parties such as Online Certificate Status Protocol (OCSP) servers, etc.—needed in this PKI in order to validate the public key for the signature s. Here are two different ways to integrate time-stamping securely:

- The receiver of (d, s) assembles the key-validating data V, requests a time-stamp certificate c for (d, s, V), and saves (d, s, V, c). A later verifier needs to revalidate each of s, V, and c.

- The signer of d computes a time-stamp certificate c for (d, s) and saves (d, s, c). Later verifiers of this triple must retrieve (from an appropriate service) a trustworthy archived version of V, and revalidate all the data.

Naturally, other choices are possible for dividing up the responsibilities.

3 Remarks

3.1 A challenge for theorists?

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, and well before a hash function is completely broken, fielded systems can swap in a new hash function.

But this does raise the question of whether it is possible to capture in a mathematically satisfying way the actual state of affairs in cryptographic security, which is that the computational difficulty of the cryptanalyst's algorithmic task is a moving target.

3.2 Practical implementation

A version of the time-stamping service described in [1] has been offered commercially by Surety since 1995 [4]. Originally, the service used MD5 and SHA-1, evaluated in parallel, as its hash function. Last year, in light of recent attacks on both of these functions, Surety deployed a new version of its service, using SHA-256 and RIPEMD-160 (also evaluated in parallel), and offered the renewal capability described above for records that were originally time-stamped with the older version of the service.

3.3 A generalization

Updating the time-stamp certificate accompanying a digital document is just one example of the sort of transformation that objects in a long-lived digital archive will undergo from time to time. In [2], the authors generalize this procedure to a broad class of transformations, describing a service that can be used to prove the integrity of the contents of a well-managed digital archive over the course of its lifetime.

References

[1] D. Bayer, S. Haber, and W.S. 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] S. Haber and P. Kamat. A content integrity service for long-term digital archives. In *Proceedings of Archiving 2006*. Society for Imaging Science and Technology, 2006. To appear. Available at `http://www.hpl.hp.com/techreports/2006/HPL-2006-54.html`.

[3] S. Haber and W.S. Stornetta. How to time-stamp a digital document. *Journal of Cryptology*, 3(2), 1991.

[4] Surety. `http://www.surety.com`.

[5] 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.

[6] 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.