Secure Names for Bit-Strings

Stuart Haber*
stuart@surety.com

W. Scott Stornetta*
scotts@surety.com

Abstract

The increasing use of digital documents, and the need to refer to them conveniently and unambiguously, raise an important question: can one "name" a digital document in a way that conveniently enables users to find it, and at the same time enables a user in possession of a document to be sure that it is indeed the one that is referred to by the name? One crucial piece of a complete solution to this problem would be a method that provides a cryptographically verifiable label for any bit-string (for example, the content, in a particular format, of the document). This problem has become even more acute with the emergence of the World-Wide Web, where a document (whose only existence may be on-line) is now typically named by giving its URL, which is merely a pointer to its virtual location at a particular moment in time.

Using a one-way hash function to call files by their hash values is cryptographically verifiable, but the resulting names are unwieldy, because of their length and randomness, and are not permanent, since as time goes on the hash function may become vulnerable to attack. We introduce procedures to create names that are short and meaningful, while at the same time they can persist indefinitely, independent of the longevity of any given hash function. This is done by naming a bit-string according to its position in a growing, directed acyclic graph of one-way hash values. We prove the security of our naming procedures under a reasonable complexity-theoretic cryptographic assumption, and then describe practical uses for these names. An implementation of our naming scheme has been in use since January 1995.

1 Introduction

Users of documents need to refer to those documents in order to keep records and in order to communicate with other users of the documents. In practice, users name their documents in various ways. A name must be unambiguous, at least in the context of its use; this requires some connection between the name and the integrity of the document it names.

In the traditional world of paper documents, there are usually reasonable guarantees of this connection. In the case of printed books and magazines, large print runs that are the result of single typesetting efforts make it easier to be confident that all copies of a printed document are the same, with a definite name printed in a conventional place in the document. Making a change to a paper document of any sort, even a small change, typically leaves forensic evidence.

A characteristic feature of digital documents, by contrast, is that they are easy to copy and to alter. The naming problem is especially troubling if the document exists only on-line and never in conventional paper-based form. For on-line documents, a useful naming scheme would allow users to employ the name to *find* documents, as well as to check the *integrity* of the documents that they find. A number of proposals have been made for such naming systems (see e.g. [SM 94, KW 95, BD+ 95]). These proposals address in different ways the problem of how to "resolve" the name into a location where the document might be found.

It is the integrity-checking problem that we address in this work: how to make sure that the bit-string content of a given digital document is indeed the same as the bit-string that was intended. Heretofore, two different sorts of mechanisms have been proposed, digital signatures and one-way hash values.

Having the author or publisher of a document compute a digital signature for its bit-string content is a reasonable use of cryptographic tools for this purpose. (See, for example, [R 95, M 94].) However, the ability to validate many digital signatures requires the presence of a public-key infrastructure, and the trustworthiness of the validation procedure relies on the assurance that the signer's private signing key is indeed secure. For some on-line documents, the infrastructure and these assurances may not be available. For long-lived documents, the security of the binding between a public key and the person or role of the putative signer becomes even more problematic. (A general solution to the latter problem is briefly described in §5.)

Thus it would be useful to have an integrity mechanism, depending on the exact contents of the bit-string in question, that does not depend on the secrecy of a cryptographic key. A natural choice for such a mechanism is the use of a one-way hash function, naming any bit-string by its hash value. (See, for example, [BD+ 95].) However, while this method is intrinsically verifiable, there are several inconvenient features:

- A desirable feature for the names given to a collection

*Surety Technologies, 1 Main Street, Chatham, N.J. 07928, U.S.A.

CCS 97, Zurich, Switzerland

of objects is that they be long-lasting, if not permanent. (This is one of the functional requirements for URNs [SM 94].) But as technology advances, any particular choice of a presumably one-way function for a naming scheme becomes less secure, so that it must be replaced (see [Dob 96a, Dob 96b]).[1] The unpleasant result is that the name of a long-lived document will need to change over time.

- Hash values are too long for a human user to remember or even to communicate easily to another human being. (For example, it is currently recommended that one-way hash functions compute outputs that are at very least 128 bits long; this is the output length of MD5 [Riv 92]. In a 6 bit/character encoding, this is 22 alphanumeric characters long.)
- The author of a bit-string document has no control over the form of its name. A one-way hash function produces a random-appearing bit-string of the appropriate length as the hash value of a document. Thus, inconvenient as it may be for the author, there will be no connection between the names of documents that are related to each other, either in form or in substance.

This paper presents a method for naming bit-strings that retains the verifiable security of hash-based names, while avoiding the constraints listed above, as well as avoiding the use of secret cryptographic keys. The method is a variation on the digital time-stamping schemes of [HS 91, BHS 93]. In summary, the essence of the new scheme is to keep a repository of hash values that depend on many bit-string inputs, and to name each bit-string by a concise description of a location in the repository to which it can be securely "linked" by a one-way hashing computation.

An implementation of our naming scheme has been running continuously since January 1995 [Sur 95].

The rest of this paper is organized as follows. After technical preliminaries in §2, including both a brief discussion of the wider problem of naming digital documents as well as a formal description of our sub-problem, we present our scheme and prove its security in §3. Motivated by the explosive growth of the Internet, we mention a number of possible applications of our scheme in §4. In §5, we describe a method for extending the lifetime of our digital names beyond the cryptographically secure lifetime of the hash functions used to compute them. Finally, we discuss several different sorts of practical implementation in §6.

2 Preliminaries

2.1 Naming digital documents

A naming system for digital documents should perform (at least) two functions. It should help the user (1) to find the document named; and (2) to reassure himself or herself that a given document is indeed the correct one, *i.e.* that it is indeed a perfect copy of the document that was intended.

To enable both these functions, the "name" could include both identification information as well as location information. System design may include procedures for registration of new documents, for finding a document given its name, ... for updating a document's location information, and for validating the integrity of a document. Typically, there is a server that "resolves" or translates a name into location information, for example into a URL or a list of URLs. The name may include other information about the document, including such data as title, author, format, price, and access privileges.

A large body of work has been devoted to the difficult problem of designing and building a naming system of this sort so that it is usable, useful, and reliable. In [SM 94] a set of functional requirements is described for Uniform Resource Names (URNs), the names to be assigned by a naming system for resources on the Internet. A number of researchers have built naming systems, including, among others, [KW 95, BD+ 95]. (This is by no means an exhaustive list.)

In this work we propose a new method for the integrity-checking piece of naming systems for digital documents. All previously proposed systems that included mechanisms for checking the integrity of the bit-string or bit-strings that make up a digital document have used either digital signatures or one-way hash functions for this purpose. For certain applications, these methods have the problems described in §1 above.

2.2 Hash functions

The principal technical tool we use in this paper is that of a one-way hash function. This is a function compressing digital documents of arbitrary length to bit-strings of a fixed length, for which it is computationally infeasible to find two different documents that are mapped by the function to the same *hash value*. (Such a pair is called a *collision* for the hash function.)

Practical proposals for one-way hash functions include those of MD5 [Riv 92], SHA-1 [NIST 94], and RIPEMD-160 [DBP 96]. Though the actual security of these functions (i.e., the precise difficulty of computing collisions for them) is not known, they are now in more or less widespread use.

Definition. In a more theoretical vein, Damgård defined a family of *collision-free hash functions* to be a family $\{H_k\}_k$ of sets of functions (indexed by a security parameter k) with the following properties:

1. Each H_k is a set of functions $h : \{0,1\}^* \to \{0,1\}^k$ that are computable in polynomial time.
2. Given k, it is easy to choose $h \in H_k$ at random.
3. It is computationally infeasible, given a random choice of one of these functions, to find a collision for the function. More precisely, for any polynomial algorithm A, for any positive constant c,

$$\Pr[h \leftarrow H_k; (x, x') \leftarrow A(h) : x \neq x', h(x) = h(x')] < k^{-c}$$

for sufficiently large k.

Damgård gave a constructive proof of their existence, on the assumption that there exist families of one-way "claw-free" permutations [Dam 87]. More generally, any "one-way group action" is sufficient [BY 90]. Concretely, the construction can be based on the difficulty either of factoring or of the discrete logarithm function. (As usual, the collision adversary A in condition (3) above can be uniform or non-uniform, depending on the precise form of the hypothesis made on the computational complexity of the underlying problem.) For

[1] For example, because of recent attacks on MD5, RSA Laboratories recommends that "in the future MD5 should no longer be implemented in signature schemes, where a collision-resistant hash function is required" [Dob 96c].

a variety of reasons, none of the known theoretical constructions of collision-free hash functions are practical.

In practice, the infeasibility of computing collisions for a particular hash function depends on the current state of the art, both the current state of algorithmic knowledge about attacking the function in question, as well as the computational speed and memory available in the best current computers. As the state of the art advances, it is likely that a function that was once securely one-way will eventually cease to be so. For example, Dobbertin's recently announced attacks on MD4 and MD5 have considerably reduced the community's confidence in the strength of these two functions [Dob 96a, Dob 96b, Dob 96c]. In §5 below we offer a solution to the problem this poses for certain practical systems whose real-world security depends on the actual infeasibility of specific computational tasks.

We refer the reader to [Pre 93] for a thorough discussion of one-way hash functions.

2.3 Theoretical model

We emphasize that this is a theoretical description of the problem of verifiably "naming" bit-strings, which is only a piece of the larger problem of naming digital documents.

The setting for our problem is a distributed network of parties. The network may include a *server* S as well as a *repository* R; parties may query the repository, asking for a copy of a particular item it contains.

Definition A *naming scheme* for this setting consists of:

- a *security parameter* k;
- a polynomial-time *naming protocol* N, possibly requiring interaction with the server S, taking as input a bit-string x, and producing as output a *name* n for x, a *certificate* c, and the addition of items to the repository R; and
- a polynomial-time *validation protocol* V, that takes as input a triple (x, n, c) and the result of a query to R, and either accepts or rejects its inputs.

If (n, c) is the output of an invocation of N on input x, then V accepts the input (x, n, c) when it is accompanied by a correct response to a query to R.

It is possible, of course, to specify a naming scheme that does not require a server or a repository. In that case, the naming protocol and the validation protocol may simply be algorithms that any party in the network may invoke without interacting with outside parties.

Definition A *counterfeiting adversary* to a naming scheme $[N, V, S]$ is a (possibly probabilistic) algorithm A that performs as follows. Given k as input, A produces (polynomially many) naming requests $x_1, x_2, \ldots$; for each x_i A is given the output of $N(x_i)$. The request x_{i+1} may be computed after A has received the response to its ith request. In addition, A may make (polynomially many) queries to R. Finally (after q naming requests, say), A's output is of the form (x, n, c). This output is a *successful counterfeit* if $x \neq x_i$ (for $i = 1 \ldots q$) and V accepts (x, n, c) (after a correct response to any queries to R).

Definition A naming scheme is *secure* if for any polynomially bounded counterfeiting adversary A and for any positive constant c, A's success probability on input k is less than k^{-c} for sufficiently large k.

To illustrate our definitions, here is a simple example of a naming scheme, where the only role of the server is to announce its random choice of a hash function $h \in H_k$. The naming procedure is just $N(x) = h(x)$ with no certificates, and V accepts (x, n) if $n = h(x)$. It is clear that this defines a secure naming scheme as long as H_k is the kth set in a family of collision-free hash functions.

We remark that the roles of S as trusted server and R as trustworthy repository in these definitions are just an artifact of how we have chosen to present and to analyze our naming schemes, allowing a clean separation between issues of the security of the scheme itself and issues of how it might be implemented in practice.

2.4 Digital time-stamping

Our solution to the naming problem builds on the work of [HS 91] and [BHS 93], whose authors describe several procedures with which users can *certify* (the bit-string contents of) their digital documents, computing for any particular document a time-stamp *certificate*. Later, any user of the system can *validate* a document-certificate pair; that is, he or she can use the certificate to verify that the document existed, in exactly its current form, at the time asserted in the certificate. It is infeasible to compute an illegitimate document-certificate pair that will pass the validation procedure.

Because we use it directly in our naming scheme, we summarize here one digital time-stamping scheme. A central "coordinating server" receives *certification* requests—essentially, hash values of files—from users. At regular intervals, the server builds a binary tree out of all the requests received during the interval, following Merkle's tree authentication technique; the leaves are the requests, and each internal node is the hash of the concatenation of its two children [Merk 80]. The root of this tree is hashed together with the previous "interval hash" to produce the current interval hash, which is placed in a widely available *repository*. The server then returns to each requester a time-stamp *certificate* consisting of the time at which the interval ended, along with the list of sibling hash values along the path leading from the requester's leaf up to the interval hash, each one accompanied by a bit indicating whether it is the right or the left sibling. The scheme also includes a *validation* procedure, allowing a user to test whether a document has been certified in exactly its current form, by querying the repository for the appropriate interval hash, and comparing it against a hash value appropriately recomputed from the document and its certificate.

It is noteworthy that the trustworthiness of the certificates computed in this scheme depends only on the integrity of the repository, and not (for example) on trusting that a particular private key has not been compromised or that a particular party's computation has been performed correctly.

3 A naming scheme for bit-strings

Next we describe a naming scheme for a network that includes a server S and a repository R. Many executions of N and of V may be performed concurrently in the network. We assume that there exists a family $\{H_k\}_k$ of collision-free hash functions. Given an initial choice of security parameter k, S announces to all parties its random choice of a one-way hash function $h \in H_k$. Our scheme is a variation on the time-stamping scheme described in §2.4 above, with

S playing the role of the coordinating server that computes certificates in response to requests and makes additions to the repository R.

We abbreviate a bit-string's certificate by omitting the list of hash values, leaving only a pointer to the relevant interval hash (for example, the time at which it was computed), and an encoding of the position of the request in the tree for that interval (for example, the sequence of left or right bits). It is this abbreviation that we propose to use as the name of the bit-string.

More explicitly, an invocation of N on input x begins with the computation of $y = h(x)$, and the submission of y to S, which includes y as one of the leaves of the tree being built in the current time interval. At the end of the interval, having built a tree of height l (that includes the previous interval hash), S places the root of the tree in R as the current interval hash with label t, say. S responds to the request by returning the certificate $c = [t; (z_1, b_1), \ldots, (z_l, b_l)]$, where each $b_i = \mathsf{L}$ or R. Finally, the name returned by N for argument x is $n = [t; b_1, \ldots, b_l]$.

One uses the entire certificate in order to validate that a particular string correctly names a particular bit-string document, first by checking that the putative name was correctly extracted from the certificate, and then by following the usual validation procedure for the document-certificate pair (recomputing the path from the leaf to the root of the tree).

To be precise, V operates as follows, given as inputs a document x, a name $n = [t; b_1, \ldots, b_l]$, and a certificate $c = [t'; (z_1, b'_1), \ldots, (z_l, b'_l)]$: First, V checks that $t = t'$ and that each $b_i = b'_i$. Next, V computes $y_1 \leftarrow h(x)$ and then (for $i \leftarrow 1 \ldots l$) if $b_i = \mathsf{L}$ then $y_{i+1} \leftarrow h(z_i \cdot y_i)$ else if $b_i = \mathsf{R}$ then $y_{i+1} \leftarrow h(y_i \cdot z_i)$. Finally, V queries R for the hash value stored at location t, and checks that it is identical to y_{l+1}. V accepts if all these checks are satisfied and rejects otherwise.

Figure 1 below illustrates the tree built by S for a time interval during which it received eight requests, containing the eight hash values a, b, c, d, e, f, g, and h. In this diagram, ab is the hash of the concatenation of a and b, etc., and IH_t and IH_{t-1} are the respective interval hashes for the current and the previous intervals. The certificate computed by S for the third request (the one containing hash value c), for example, is the following:

$$[t;\ (d, \mathsf{R}), (ab, \mathsf{L}), (eh, \mathsf{R}), (IH_{t-1}, \mathsf{L})].$$

3.1 Security

The security of this naming scheme follows directly from the infeasibility of computing hash collisions for functions from $\{H_k\}_k$, since the only possible counterfeit names include hash collisions. In essence, if x is a bit-string on which N was never invoked during a run, any triple (x, n, c) that V will accept (after the correct response to a query to R) will include a hash collision for the function h announced by S at the beginning of the run: either x itself or one of the hash values z_i in c (when combined on the left or the right with y_i) collides with another argument to h whose hash value was computed during the run. Therefore we have the following theorem.

Theorem 1 *If $\{H_k\}_k$ is a family of collision-free hash functions, then the naming scheme $[N, V, S]$ described above is secure.*

Because the reduction in the proof is so direct, it is easy to give an "exact security" analysis (cf. [Lev 85, BKR 94]) of the strength of this scheme, whether the hash functions used are from the collision-free family provided by a theoretical cryptographic assumption or rather practical hash functions, as in the implementations described in §6 below.

3.2 Variations on the scheme

Of course, the secure verifiability of the names assigned by the scheme described above does not depend on the particular combination of binary trees and linked lists used. By systematically invoking the hash function on pairs or ordered lists of hash values, new hash values can be computed from old ones so as to form a directed acyclic graph (by directing an edge from each of the inputs to the hash value output). Design considerations (including those discussed in §6.1 below) may dictate several different combinatorial structures for this directed graph.

Whatever the structure of the growing graph of hash values, it is secured by making portions of the graph widely witnessed and widely available. To insure the verifiability of the names, it suffices that every document in the naming structure be linked by a directed path to a widely witnessed hash value; a standard ordering of the incoming edges at each node can be used to encode the path. Then the name of a document is given by this encoding of its location in the graph, together with a pointer to the hash value at the end of the path, and the argument of Theorem 1 applies.

For example, in one variation of the scheme described above, a list of documents may be used to build a local tree (following Merkle, again), whose root is sent off in turn as a request to the coordinating server. The location information for a document in this "tree-of-trees" scheme can be written as a position in the server's tree followed by a position in the local tree.

In another variation, the widely witnessed hash values in the repository could consist simply of a linked list (as in the simple linking scheme of [HS 91]). In this case the location information for a document is a simple pointer into the repository.

4 Applications

The problem of naming digital documents might have seemed like a curiosity only a few years ago. However, with the growth in use of the Internet, more and more people need to be able to refer confidently to meaningful bit-sequences. The problem is now a matter of immediate practical concern.

The problem has become especially acute with the emergence of the World-Wide Web. Jumping from one URL (Uniform Resource Locator) to the next in a sequence of WWW documents may seem at first to be exactly analogous to following a bibliographic reference in a traditional scholarly paper. In fact it is something quite different: a URL is only a pointer to a location, with no guarantee that what a user finds there today is the same reference that the author originally intended. If on-line citations include secure names for the bit-string contents of the documents cited, then it is possible to traverse a path of citations with confidence that one is indeed following the authors' intentions. This ability would be especially useful for the many documents on the World-Wide Web that exist only on-line.

In most electronic commerce systems, transaction records of all sorts are kept on-line, and it would be useful to have a cryptographically secure means of assigning serial numbers or tracking numbers to these records.

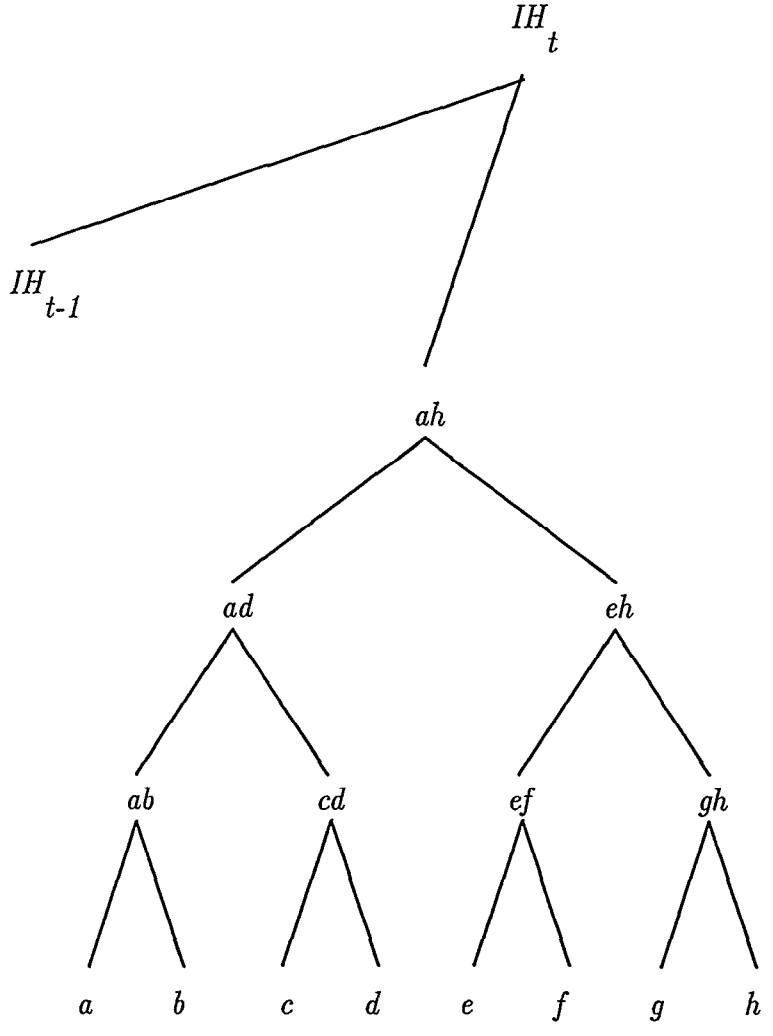

Figure 1: 8-leaf tree for the example of §3.

Software code is another class of digital document for which it would be useful to have an easy way for a short name to carry a guarantee of integrity. A user who downloads software (along with its naming certificate) from a site on the Net can be sure of its integrity if he or she is able to check that the code is correctly named by a short string of letters and numbers. Here, of course, bit-string equality is *exactly* the point. The great strength of using secure names in this application is that the short name of a program is considerably easier to distribute widely and robustly than the program itself. (It is also easier to distribute reliably than the sort of public-key infrastructure information that is required in order to use digital signatures in order to validate the integrity of code.)

For another example of a type of large digital document whose integrity matters a great deal, consider the case of genetic data. Scientists now routinely download others' data sets for use in their own research. The use of our naming scheme would allow the user to be sure of the data's integrity, as well as providing a convenient and verifiable way to cite the data in published descriptions of the work that was done with it.

5 Long-lived names

The technique described in [BHS 93] for renewing cryptographic certifications of authenticity applies directly to the certificates of the present naming scheme.

The renewing process works as follows. Let us suppose that an implementation of a particular time-stamping system is in place, and consider the pair (x, C), where C is a valid time-stamp certificate (in this implementation) for the bit-string x. Now suppose that an improved time-stamping system is implemented and put into practice—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. Further suppose that the pair (x, C) is time-stamped by the new system, resulting in a new certificate C', and that some time later, *i.e.* at a definite later date, the original method is compromised. C' provides evidence not only that the document contents x existed prior to the time of the new time-stamp, but that it existed at the time stated in the original certificate, C; prior to the compromise of the old implementation, the only way to create a certificate was by legitimate means. (It is similarly recommended that if a digitally signed document is likely to be important for a long time—perhaps longer than the signer's key will be valid—then the document-signature pair should be time-stamped [BHS 93, Odl 95, HKS 95].)

In our naming schemes, the verifiable name for the bit-string x is a standard abbreviation a for its original certificate C. In order that a continue to be verifiable as a name for x, the certificate C should be renewed (as above) from time to time as new time-stamping systems are put in place. As long as this is done, a is still a verifiable name for x. There is now an additional step to the procedure for validating the name: after checking that a is correctly extracted from C, one must follow the usual time-stamp validation procedure for the certificate, which now includes both the original-system validation of (x, C) and the new-system validation of $[(x, C), C']$. We note that in practice this additional validation step would be automated, and would not at all affect the convenient use of a to name x.

6 Practical implementations

A practical implementation of a naming scheme cannot use the known theoretical constructions of collision-free hash functions. If the decision is made to use practical one-way hash functions such as MD5, then users of the system do not need to trust the server's random choice of a function $h \in H_k$. (However, they do have to hope that the hash function chosen is one-way in practice; see section §5 for one way to allay users' concerns on this score.)

The naming scheme described in §3 above, based on the digital time-stamping scheme described in §2.4, was implemented by Surety Technologies, and has been in continuous commercial use since January 1995. The implementation uses practical hash functions; specifically, the current implementation uses $h(x) = (\mathrm{MD5}(x), \mathrm{SHA}(x))$ as the hash value for any argument x. A number of supplemental mechanisms are employed in order to maintain the integrity and wide distribution of the repository [Sur 95].

The names assigned by our scheme are indeed concise, growing essentially as slowly as possible while still providing unique names. If the repository contains n interval hashes, and no more than m naming requests are received during each interval, the names can be written with at most $\lg_2 nm$ bits. Just to give a numerical example, a repository representing a thousand requests per minute for the length of a century requires 36-bit names; in the MIME encoding (six bits per alphanumeric character) such a name can be jotted down with six characters, while hash-value names of this length are completely insecure.

6.1 Meaningful names

There are several variations of our naming scheme that allow an author a fair measure of control over the names of his or her documents, so that the author can choose a verifiable name that is meaningful in one or another useful way.

First, and most obviously, observe that in the scheme described in detail in §3 a convenient way to encode the location in the repository to which a document's contents are linked is by the date and time at which the interval hash at that location was computed. Instead of (*e.g.*) a MIME encoding of the number of seconds since a moment in early 1970 (Unix standard time), it would often be useful to express at least a part of this date and time in human-readable form.

In a slight variation, we can allow "personalized" naming requests, as follows. Suppose that the repository items are formatted in a standard way every day, and let $F(\cdot)$ denote any standard mapping from ASCII-encoded strings to the list of daily repository locations. When the server receives a personalized naming request that includes the ASCII string s, the request is held until the appropriate moment in the day and then linked to the widely witnessed hash value stored at location $F(s)$; in this way, s is made to be part of the name of the documents included in those special naming requests. Thus, for example, the author of *The History of Computers in Zurich* can arrange for the verifiable name of its bit-string contents to have the form ["The History of Computers in Zurich" `date suffix`], where `suffix` includes a few bits of disambiguating information that distinguishes this request from all others that were linked to the same repository location.

In another example, consider the tree-of-trees variation briefly mentioned in §3.2. An author can name a multi-part document by placing the contents of each successive part at

consecutive leaf nodes of a local tree. The resulting request to the server gives the consecutive parts of the document consecutive local positions and therefore consecutive names. Furthermore, the other portions of these consecutive names are identical, explicitly encoding the fact that they are parts of the same document. And local trees can have sub-trees, so that our historian can arrange to name the ith section of the jth chapter of his masterpiece ["The History of Computers in Zurich" `infix` $i.j$], for all appropriate pairs (i, j).

More complicated ways of structuring the parts of a document can similarly be encoded in the verifiable names assigned by our naming scheme. Note that conventional naming schemes do allow for encoding document structure into names, but not in a verifiable manner.

In another variation, a table of contents for a long or complicated multi-part document can be included in a standard place in the request—for example, as its last piece. The table of contents may contain more or less detailed descriptions of the parts of the document. At a later time, together with a list of documents to be authenticated and their certificates, such an authenticated table of contents can be used to verify (1) that each document in the list is an exact copy of one that was registered with the table of contents, and (2) that none of the documents in the list are missing.

Acknowledgements

We would like to thank Ralph Merkle, R. Venkatesan, Matt Franklin, Avi Rubin, Bill Arms, and Dave Richards for helpful discussions about this work. We would also like to thank the anonymous referees for their very useful suggestions.

References

[BHS 93] D. Bayer, S. Haber, and W.S. Stornetta. Improving the efficiency and reliability of digital time-stamping. In *Sequences II: Methods in Communication, Security, and Computer Science*, ed. R.M. Capocelli, A. De Santis, U. Vaccaro, pp. 329–334, Springer-Verlag, New York (1993).

[BKR 94] M. Bellare, J. Kilian, and P. Rogaway. The security of cipher block chaining. In *Advances in Cryptology—Crypto '94*, Lecture Notes in Computer Science, Vol. 839, ed. Y. Desmedt, pp. 94-107, Springer-Verlag (1994).

[BY 90] G. Brassard and M. Yung. One-way group actions. In *Advances in Cryptology—Crypto '90*, Lecture Notes in Computer Science, Vol. 537, pp. 94-107, Springer-Verlag (1991).

[BD+ 95] S. Browne, J. Dongarra, S. Green, K. Moore, T. Pepin, T. Rowan, and R. Wade. Location-independent naming for virtual distributed software repositories. Univeristy of Tennessee Computer Science TR 95-278 (1995). (Available at `http://www.cs.utk.edu/~library/TechReports/1995/`).

[Dam 87] I. Damgård. Collision-free hash functions and public-key signature schemes. In *Advances in Cryptology—Eurocrypt '87*, Lecture Notes in Computer Science, Vol. 304, pp. 203–217, Springer-Verlag (Berlin, 1988).

[Dob 96a] H. Dobbertin. Cryptanalysis of MD4. In *Fast Software Encryption*, Lecture Notes in Computer Science, Vol. 1039, ed. D. Gollman, pp. 53–69, Springer-Verlag (Berlin, 1996).

[Dob 96b] H. Dobbertin. Cryptanalysis of MD5 compress. Private communication (May 1996). Described by B. Preneel, Rump Session, Eurocrypt '96 (May 1996).

[Dob 96c] H. Dobbertin. The status of MD5 after a recent attack. *CrytoBytes*, Vol. 2, No. 2 (Summer 1996).

[DBP 96] H. Dobbertin, A. Bosselaers, and B. Preneel. RIPEMD-160: A strengthened version of RIPEMD. In *Fast Software Encryption*, Lecture Notes in Computer Science, Vol. 1039, ed. D. Gollman, pp. 71–82, Springer-Verlag (Berlin, 1996).

[HKS 95] S. Haber, B. Kaliski, and W.S. Stornetta. How do digital time-stamps support digital signatures? *CryptoBytes*, Vol. 1, No. 3 (Autumn 1995). (Available at `http://www.rsa.com/rsalabs/pubs/cryptobytes.html`.)

[HS 91] S. Haber and W.S. Stornetta. How to time-stamp a digital document. *Journal of Cryptology*, Vol. 3, No. 2, pp. 99–111 (1991).

[KW 95] R. Kahn and R. Wilensky. A framework for distributed digital object services. Corporation for National Research Initiatives technical report cnri.dlib/tn95-01 (May 1995). (Available at `http://www.cnri.reston.va.us/`.)

[Lev 85] L.A. Levin. One-way functions and pseudorandom generators. In *Proceedings of the 17th Annual Symposium on Theory of Computing*, pp. 363-365, ACM (1987).

[Merk 80] R.C. Merkle. Protocols for public key cryptosystems. In *Proc. 1980 Symposium on Security and Privacy*, IEEE Computer Society, pp. 122–133 (April 1980).

[M 94] J.W. Moore. The use of encryption to ensure the integrity of reusable software components. In *Proc. 3rd International Conf. on Software Reusability*, IEEE Computer Society Press (November 1994).

[NIST 94] National Institute of Standards and Technology. Secure Hash Standard. NIST Federal Information Processing Standard Publication 180-1 (May 1994).

[Odl 95] A. Odlyzko. The future of integer factorization. *CrytoBytes*, Vol. 1, No. 2 (1995).

[Pre 93] B. Preneel. *Analysis and Design of Cryptographic Hash Functions.* Ph.D. dissertation, Katholieke Universiteit Leuven (January 1993).

[Riv 92] R. Rivest. The MD5 Message-Digest Algorithm. Internet Network Working Group Request for Comments 1321 (April 1992).

[R 95] A. Rubin. Trusted distribution of software over the Internet. In *Internet Society 1995 Symposium on Network and Distributed System Security* (1995).

[SM 94] K. Sollins and L. Masinter. Functional requirements for Uniform Resource Names. Internet Network Working Group Request for Comments 1737 (December 1994).

[Sur 95] Surety Technologies, Inc. Answers to Frequently Asked Questions about the Digital Notary™ System. http://www.surety.com (since January 1995).

expending resources to add gold to circulation. In our case, it is CPU time and .electricity that is expended
The incentive can also be funded with transaction fees. If the output value of a transaction is less than its input value, the difference is a transaction fee that is added to the incentive value of the block containing the transaction. Once a predetermined number of coins have entered circulation, the incentive can transition entirely to transaction fees and be completely inflation .free
The incentive may help encourage nodes to stay honest. If a greedy attacker is able to assemble more CPU power than all the honest nodes, he would have to choose between using it to defraud people by stealing back his payments, or using it to generate new coins. He ought to find it more profitable to play by the rules, such rules that favour him with more new coins than everyone else combined, than to undermine the system and the validity of his own .wealth
Reclaiming Disk Space .7
Once the latest transaction in a coin is buried under enough blocks, the spent transactions before it can be discarded to save disk space. To facilitate this without breaking the block's hash, transactions are hashed in a Merkle Tree [7][2][5], with only the root included in the block's hash. Old blocks can then be compacted by stubbing off branches of the tree. The interior hashes do not need .to be stored
Block
Hash0 Hash1 Hash2 Hash3
Tx0 Tx1 Tx2 Tx3
Block Header (Block Hash)
Prev Hash
Nonce
Root Hash
Hash01
Hash23
Block
Block Header (Block Hash)
Prev Hash
Nonce
Root Hash
Hash01
Hash23
Hash2 Hash3 Tx3
Transactions Hashed in a Merkle Tree After Pruning Tx0-2 from the Block
A block header with no transactions would be about 80 bytes. If we suppose blocks are generated every 10 minutes, 80 bytes * 6 * 24 * 365 = 4.2MB per year. With computer systems typically selling with 2GB of RAM as of 2008, and Moore's Law predicting current growth of 1.2GB per year, storage should not be a problem even if the block headers must be .kept in memory
4

Simplified Payment .8
Verification
It is possible to verify payments without running a full network node. A user only needs to keep a copy of the block headers of the longest proof-of-work chain, which he can get by querying network nodes until he's convinced he has the longest