Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®
Computer Networks 43 (2003) 339–349
COMPUTER NETWORKS
www.elsevier.com/locate/comnet

How to construct optimal one-time signatures

Kemal Bicakci [a,*], Gene Tsudik [b], Brian Tung [c]

[a] Informatics Institute, Middle East Technical University, Ankara 06531, Turkey
[b] Computer Science Department, University of California, Irvine, CA 92612, USA
[c] USC Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA 90292, USA

Received 25 March 2002; received in revised form 23 April 2003; accepted 24 April 2003

Responsible Editor: S.F. Wu

Abstract

One-time signature (OTS) offer a viable alternative to public key-based digital signatures. OTS security is typically based only on the strength of the underlying one-way function and does not depend on the conjectured difficulty of some mathematical problem. Although many OTS methods have been proposed in the past, no solid foundation exists for judging their efficiency or optimality. This paper develops a methodology for evaluating OTS methods and presents optimal OTS techniques for a single OTS or a tree with many OTS's. These techniques can be used in a seesaw mode to obtain the desired tradeoff between various parameters such as the cost of signature generation and its subsequent verification.

Keywords: One-time signature; Digital signature; On-line/off-line digital signature; Hash function; Combinatorics

1. Introduction

Modern networks and Internetworks are more open than ever before, in an attempt to make information available on a ubiquitous basis. Networks are also faster than before, with available bandwidths measured in Gb/s. However, instead of alleviating congestion on the information highway, this has only encouraged the transmission of greater numbers of large data objects, especially with the recent popularity of multimedia presentations, voice- and video-conferencing, and large-scale scientific computing. The composition of network traffic has changed from yesterday's text files to today's enormous datasets produced by sophisticated remote visualization and rendering tools.

These developments make it important to maintain data integrity and privacy in a manner that is both highly secure and efficient. Traditional digital signature methods based on public key cryptography are simply untenable from a performance perspective. Furthermore, the security of public key cryptosystems (e.g., RSA or DSS [1,2]) is based on complex mathematical problems, such as factoring or discrete logarithms. The mathematical basis is both a blessing and a curse: the former because it lends itself to simple and elegant

* Corresponding author. Tel.: +90-312-2103796; fax: +90-312-2103745.
E-mail addresses: bicakci@ii.metu.edu.tr (K. Bicakci), gts@ics.uci.edu (G. Tsudik), brian@isi.edu (B. Tung).

1389-1286/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/S1389-1286(03)00285-8

design, and the latter because there is no assurance that there are no efficient algorithms for solving the underlying mathematical problems.

One-time signature (OTS) provide an attractive alternative to public key-based signatures. Unlike signatures based on public key cryptography, OTS is based on nothing more than a one-way functions (OWFs). [1] Consequently, OTSs are claimed to be more efficient since no complex arithmetic is typically involved in either OTS generation or verification. In practice, security of traditional public key-based digital signatures is based on two factors: conjectured-hard mathematical problems and the message digest function used to produce a fixed-size digest from arbitrarily long input data. (A secure message digest function suitable for this purpose must be both one-way and collision-resistant.) Using OTSs essentially allows us to eliminate the first factor altogether.

The OTS concept has been known for over two decades. It was initially developed by Lamport [6] and subsequently enhanced by Merkle [7] and Winternitz [8]. Bleichenbacher et al. [9–11] formalized the concept of OTS using directed acyclic graphs (DAGs).

In the simplest case, a message signer prepares an OTS by first generating a random number r which serves as a one-time private key. The signer then securely distributes a one-time public key $h(r)$, where $h(\cdot)$ is a suitable collision-resistant OWF. This public key, sometimes also referred to as an *anchor value*, is later used by the signature verifier(s) to verify the signature.

A signature is constructed by revealing the one-time private key r. A receiver (verifier) that obtains r' (which may or may not be the same as r) checks that it could only be have been generated by the claimed signer by computing $h(r')$. If this value matches the one-time public key $h(r)$, the OTS is considered valid. This, in effect, allows the signing of a *predictable* 1-bit value and provides one-time origin authentication. In order to sign *any* 1-bit value, two random numbers $\{r_0, r_1\}$ are needed. This way, both $h(r_0)$ and $h(r_1)$ are pre-distributed but at most one of $\{r_0, r_1\}$ is revealed as part of a signature. The pair $(r_0, h(r_0))$ represents an OTS of message "0", whereas $(r_1, h(r_1))$ is an OTS of "1".

Merkle extended this method to allow the signing of an arbitrary message. It begins by reducing the message to a fixed-length quantity using a collision-resistant message digest function, as is customary with traditional public key signatures. However, instead of transforming this quantity with a private key, each bit has an associated OTS and the signature for the entire message is represented as the concatenation of the OTS for each "1" bit in the message digest, along with some extra values to ensure that this per-bit signature is not itself modified.

As stated, this algorithm requires the one-time public keys for the OTSs to be distributed in a secure fashion. Since this is typically done using public key methods, the benefit of using efficient OTSs is apparently lost. However in [7], Merkle also introduced a scheme where these signatures are embedded in a tree structure, allowing the cost of a single public key signature (to sign the initial anchor values) to be amortized over many OTSs. In this formulation, signatures are longer, by at most an order of magnitude. However, the extra length (which was a concern two decades ago) is negligible today owing to the high speed of modern networks.

Despite their performance advantage and increased security, OTSs have remained on the periphery of security research since their inception. In particular, no practical evaluation of OTS capabilities has been done. This open issue is precisely the topic of the present paper. In order to obtain better understanding of OTS optimality, we first address a more general issue of how to maximize the message size (of a message to be signed) while minimizing the number of random quantities to be used in OTS generation (and, hence, the number of OWF operations). Our result leads us towards an optimal OTS construction where efficiency corresponds to the smallest number of OWF operations used in both generation and verification of an OTS. We then amend this definition of efficiency to take into account situations where multiple verifications are necessary, e.g.,

[1] Examples of conjectured OWFs include DES [3], MD5 [4], and SHA [5]. There is strong (albeit, folkloric) evidence as to the existence of true OWFs.

with multi-destination e-mail or, more generally, secure multicast. This leads us to consider a slightly different notion of optimality.

The outline of the paper is as follows. After introducing Merkle's signature algorithm in Section 2, we generalize the OTS constructions and present our optimal technique in Section 3. We evaluate the performance of our OTS construction in directed acyclic computation graph notation in Section 4. Section 5 is for describing how to encode a message for the signature using our technique. We make the cost analysis of a single OTS or a tree with many OTSs in Sections 6 and 7, respectively. In Section 8, we present a method to construct the optimal tree, more precisely we show how to choose the optimal depth of this tree. Section 9 discusses practical implementation aspects and possibilities for future work and Section 10 concludes this paper.

2. Merkles one-time signature construction

One notable and efficient OTS construction is due to Merkle [12]. (Others can be found in [13,14].) Assuming input messages of size b, let $s = (\lfloor \log b \rfloor + 1)$ and let $n = b + s$. The signer generates a secret key vector of size $(b + 2s)$ of random numbers:

$$R = \{R_1, \ldots, R_b, L_{1,0}, L_{1,1}, \ldots, L_{s,0}, L_{s,1}\}.$$

The signer then applies the OWF to each element of the secret key vector and distributes the resulting public key vector to the intended verifier(s):

$$H(R) = \langle H(R_1), \ldots, H(R_b), H(L_{1,0}), H(L_{1,1}), \ldots, H(L_{s,0}), H(L_{s,1}) \rangle.$$

Subsequently, to sign a b-bit message m, the signer counts the number of "1" bits in m, encodes the count as an s-bit string and appends it to m. The result is an n-bit message m'. The actual signature $\mathrm{SIG}(m)$ is constructed as follows:

for $i = 1$ to b **do begin**
  if $(m[i] == 1)$ **then** /* ith bit of m' is "1" */
    release $H(R_i)$
end /* for */
for $i = (b + 1)$ to n **do begin**
  if $(m[i] == 1)$
    release $H(L_{i,1})$
  else
    release $H(L_{i,0})$
end /* for */

For example, if $b = 4$ (thus, $n = 7$) and $m = 0101$, then $m' = 0101010$ and $\mathrm{SIG}(m) = \{R_2, R_4, L_{1,0}, L_{2,1}, L_{3,0}\}$. The verifier checks the signature by applying H to each element of $\mathrm{SIG}(m)$ and checking it against the public key vector $H(R)$.

To summarize the cost of Merkle's OTS construction, the signer generates $(b + 2s)$ random numbers and performs as many OWF computations. Each verifier performs, on the average, $(b/2 + s)$ OWF computations. For example, for a 160-bit message (e.g., an SHA1 digest), 176 and 88 OWF operations are needed to sign and verify, respectively.

Despite its relatively low cost and simplicity, the above is basically an ad hoc construction. No argument for its optimality has been provided in Merkle's work. Moreover, it remains unclear what optimality means in the context of an OTS system.

3. One-time signature generalization

More generally, a message sender prepares a signature by generating an n-element random number vector $R = (r_1, r_2, \ldots, r_n)$. He then computes $H(R) = \langle H(r_1), H(r_2), \ldots, H(r_n) \rangle$ where $H(\cdot)$ is a suitable OWF. The sender then securely distributes the one-time public key vector $H(R)$ to all intended verifiers.

Signature generation is the process of mapping the input message into a subset $S \subset R$. S is then attached to the message as its signature. To verify S, each receiver computes a similar mapping from the input message into a subset T of $H(R)$. The signature is considered valid only if $T = H(S)$.

The mapping function must satisfy a condition which we refer to as *incomparability:* for any message D_1, an attacker must be unable to find another message D_2 such that $F(D_2) \subset F(D_1)$ where $F(D_i)$ corresponds to signature subset S_i for the message D_i. Otherwise, if the legitimate signer distributes $\langle D_1, F(D_1) \rangle$, the attacker could replace

D_1 with D_2, reduce the set $F(D_1)$ to $F(D_2)$, and obtain a valid message–signature pair $\langle D_2, F(D_2)\rangle$.

This leads us to ask the question: *When R contains n random numbers, how many distinct messages can be signed or in other words what would be the maximum size of the message space?*

For $n = 1$, the answer is one, and the signature is one random number. For $n = 2$, the answer is two: the signature can be either r_1 or r_2. If we were to map a message onto the signature subset $\{r_1, r_2\}$, that choice would eliminate any other subset, allowing a single distinct message.

In general, we observe that, for any n, we can obtain a valid message mapping by drawing from all subsets containing $p < n$ random numbers. Clearly, no one such subset can be the subset of another, allowing us $C(n,p) = n!/p!(n-p)!$ distinct messages.

In [15], it is shown that for any n, the domain of mapping M is greatest when $p = \lfloor n/2 \rfloor$. This allows us to sign any one of

$$B_n = \binom{n}{\lfloor n/2 \rfloor} \tag{1}$$

distinct messages, i.e., we are able to sign an arbitrary $(\log B_n)$-bit message. For example, if R contains four elements 1, 2, 3, and 4, then the largest valid message set of R is

$$V = \{\{1,2\},\{1,3\},\{1,4\},\{2,3\},\{2,4\},\{3,4\}\}$$

which contains $B_4 = 6$ elements.

By inverting this formula, and using Stirling's approximation, we can see that to represent 2^b distinct values, n must satisfy

$$B_n > 2^b,$$

$$\frac{\sqrt{2\pi n}(n/e)^n}{[\sqrt{\pi n}(n/2e)^{n/2}]^2} > 2^b,$$

$$2^n\sqrt{2/\pi n} > 2^b$$

or, after taking base 2 logarithm of both sides,

$$n - \lg\sqrt{\pi n/2} > b. \tag{2}$$

For $b = 128$ (e.g., MD5), n must be at least 132, and each subset can be as small as size 64, since $C(132,64) > 2^{128}$. For $b = 160$ (e.g., SHA1), n must be at least 165 ($n = 164$ is just barely insufficient), with subsets of size 75.

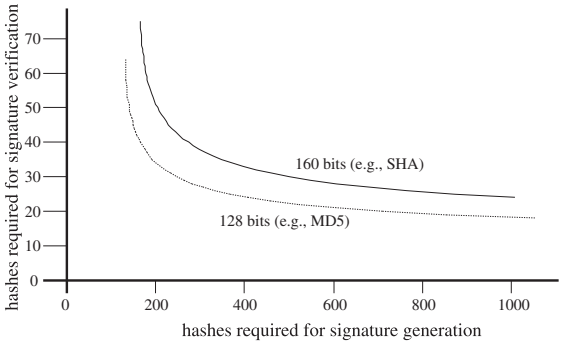

Fig. 1. Signature generation/verification hash profile.

Note that we can freely increase n and decrease p, or similarly, decrease n and increase p, provided $C(n,p) > 2^b$ and the signer and verifier(s) decide beforehand on the values of n and p. At one extreme, as we have shown, there is the lowest n such that there exists a p so that $C(n,p) > 2^b$. At the other extreme, one could choose $n = 2^b$, and allow $p = 1$; this would correspond to the case where there is a random number associated with each possible message, and the sender simply picks the appropriate one for each message to be signed! (This is clearly an intractable storage problem.)

In Fig. 1, we show the number of random numbers n versus the number of hashes required for verification p, for two popular message (digest) sizes.

We also observe that, for a valid (n,p) pair satisfying $C(n,p) > 2^b$, if the anchor values $H(R)$ are encrypted, the release of a subset in V containing p numbers can be used to exchange b bits of confidential information, since no one observing the p numbers can distinguish them from any others, or verify them, without being able to decrypt $H(R)$.

4. Efficiency assessment using directed acyclic graphs

Bleichenbacher et al. [9–11] formalized the concept of OTS using DAGs. They observed that the structure of the OTS computation leading

from the secret key components to the public key can be represented as a DAG $G = (V, E)$ with vertex set V and and edge set E, where the vertices correspond to the secret key, the intermediate results, and the public key, and where a directed edge (v_i, v_j) in E indicates that v_i is an input to the OWF and computation resulting in v_j.

In order to design a OTS based on a DAG, there are two important requirements. First, every OTS must be verifiable, i.e., the public key must be computable from it. Second, in order to prevent forgery, the set of signatures must satisfy the incomparability condition defined in previous section.

If we assume that all the public-key components are hashed in a binary tree to result in a single public-key component, then an efficient OTS algorithm can be formally defined as one in which the size of the message space is maximized while the size of the DAG is minimized. More precisely, if $\lg(\Gamma)$ is the number of message bits that can be signed, the efficiency of a one-time digital signature scheme Γ for a DAG G with $z = |V|$ vertices is given by

$$\eta(\Gamma) = \frac{\lg(\Gamma)}{z+1}.$$

In [10], the authors also presented their best graph construction, for which the efficiency is approximately equal to 0.473. We will now prove that our methodology provides a better construction in terms of their notion of efficiency.

In the previous section, we showed that the number of bits that can be signed using n random numbers is upper bounded by $n - \lg\sqrt{\pi n/2}$. In Fig. 2, we demonstrate that when we assemble the DAG of our construction with n random numbers, the number of vertices is equal to $2n + 1$. Then the efficiency is upper bounded by 0.5 as seen from Eq. (3).

$$\lim_{n\to\infty} \eta(\Gamma) = \frac{n - \lg\sqrt{\pi n/2}}{2n+2} = 0.5. \tag{3}$$

For $b = 128$ (e.g., MD5), this value is 0.4812 and for $b = 160$ (e.g., SHA1), it is 0.4819. We observe that both of them is better than the best construction in [10].

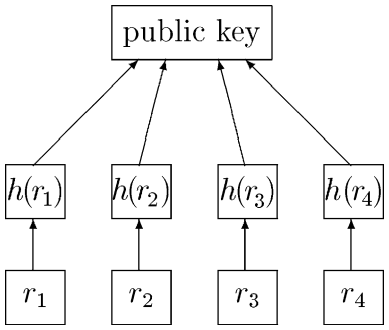

Fig. 2. The DAG representation of our construction for $n = 4$ ($z = 2n + 1 = 9$) where the public-key components are hashed in a binary tree to result in a single public-key component. To make the OTS verifiable, the signature contains the random numbers released as well as the hashes of unreleased random numbers.

Other than the efficiency concerns, we claim that our methodology is better than Bleichenbacher et. al.'s approach in two aspects:

- In practice, their proposed DAG construction is very complex and hard to implement whereas our combinatorial results are elementary and easy to grasp.
- In [10], the authors did not discuss how to encode a specific message for the DAG they used. In contrast, we provide an efficient method for encoding a message for signature in the next section.

5. Encoding a message for signature

Given a vector R of n random numbers, and a valid message set V of subsets each containing p of those numbers, we have shown that we can sign any one of $C(n, p)$ distinct messages. In this section, we describe the mapping M between messages and the elements of V, and demonstrate how to compute them efficiently.

Assume that the domain of M is composed of 2^b messages, and we have a way of representing each of the messages as a b-bit integer m. Let any subset S in V be expressed as $\{R_{a_1}, R_{a_2}, \ldots, R_{a_p}\}$. Arrange the subsets in V in lexically ascending order. For example, for $n = 4$, $p = 2$, the subsets are ordered

$$\{R_1, R_2\}, \{R_1, R_3\}, \{R_1, R_4\}, \{R_2, R_3\}, \{R_2, R_4\}, \{R_3, R_4\}.$$

Then the mapping $m = M(S, V)$ of each subset S is defined as the integer position of S in this list representation of V. For example, in the above case, $M(\{R_1, R_3\}, V) = 2$ and $M(\{R_3, R_4\}, V) = 6$. In general, for any n and p, the mapping of any subset $S = \{R_{a_1}, R_{a_2}, \ldots, R_{a_p}\}$, where $a_0 = 0$ and $a_1 < a_2 < \cdots < a_p$ is given by

$$M(S, V) = 1 + \sum_{i=1}^{p} \sum_{j=n-a_i+1}^{n-a_{i-1}-1} \binom{j}{p-i}. \tag{4}$$

Note that in order to compute the mapping for any subset, for a given n and p, we need only compute the binomial coefficients $C(j, p-i)$ for i from 1 to p, j from $p+1-i$ to $n-i$. Thus, each mapping requires $n - p - (n - a_p) = a_p - p$ additions.

Similarly, the mapping $S = M^{-1}(m, V)$ of a message represented by the integer m can be computed by subtracting binomial coefficients until zero is reached. This requires $a_p - p$ additions and comparisons. Pseudocode to do this conversion is as follows:

```
m_0 = m /* copy message to temporary value */
q = 1
for i = 1 to p do begin
   while m_0 > C(n - q, p - i) do begin
      m_0 := m_0 - C(n - q, p - i)
      q := q + 1
   end /* while */
   a_i := q
   q := q + 1
end /* for */
```

To put things in perspective, consider that a single MD5 hash computation requires approximately 500 arithmetic operations. Thus, our mapping (in both directions) costs less than one MD5 hash.

6. Cost analysis of a single one-time signature

6.1. All on-line case

In the preceding sections, we showed how to sign an arbitrary b-bit message using p of n random numbers. In this section, we will describe how to choose n and p to minimize the total cost of a OTS. Our initial assumption is that all of the signing process is performed on-line, once the message is presented.

The principal cost of generating a OTS (aside from the cost of securely distributing the anchor values $H(R)$) is the cost of computing $H(R)$; this costs n hashes. The principal cost of verifying a OTS (aside from the cost of verifying the anchor values) is the cost of computing $H(S)$; this costs p hashes. However, only a single sender generates a OTS, while potentially many receivers verify it. Thus, each hash involved in signature verification incurs a greater cost than one involved in signature generation.

In general, let each verification hash cost σ times as much as a generation hash. The total cost of a single signature is then proportional to $n + \sigma p$; this is the quantity that we shall try to minimize, subject to the condition that $C(n, p) > 2^b$.

We want to find n and p such that $C(n, p) \doteq C(n + \sigma, p - 1)$. As a first approximation, we have, from the definition of the binomial coefficient

$$\left(\frac{n}{n-p}\right)^{\sigma} \doteq \frac{n-p}{p}. \tag{5}$$

If we let $\alpha = p/n$, we have

$$(1-\alpha)^{\sigma} = \frac{\alpha}{1-\alpha}, \tag{6}$$

$$\alpha = (1-\alpha)^{\sigma+1}. \tag{7}$$

To find the optimal n and p, we find the p such that $C(\lfloor \alpha p \rfloor, p) > 2^b$.

6.2. On-line/off-line case

In [14], the authors introduce the new concept of on-line/off-line digital signature schemes. In these schemes the signing of a message is broken into two phases. The first phase is off-line. Though it requires a moderate amount of computation, it presents the advantage that it can be performed at leisure, before the message to be signed is even known. The second phase is on-line and it starts after the message becomes known. Since it utilizes

the precomputation of the first phase, it is much faster.

We observe that the signer can generate the vector $R = \{R_1, \ldots, R_n\}$, and by applying the OWF to each random number, he can generate the hashes off-line. The on-line phase is just a mapping and at the end of Section 5 we showed that it costs less than one MD5 hash operation. So, in this case we try to minimize the verification time. This is also due to the fact that many times only a single sender generates an OTS, but potentially many receivers verify it.

It is improper to take the verification time as only the time needed to make the mapping and generate the hashes of the random numbers. One of the disadvantages of OTS is its length; especially if we have a low bandwidth channel, the time needed to transmit the signature dominates the time for verification and cannot be neglected. Also, available bandwidth and computation power both vary over a wide range. We may therefore optimize n and p with respect to bandwidth and computation power.

Here, we will describe how to choose n and p to minimize the total time T needed to verify one OTS. Let's take the hash length as b and random number length as a. Assume a bandwidth of K bits/s and L seconds as the time required to perform one hash operation. Then

$$T = \frac{ap + bn}{K} + (p+1)L + m. \tag{8}$$

In the above formula, we ignore other delays such as queuing delays. The extra L is for generating the hash of the message, and m is the time to compute the mapping. The total time needed to verify one OTS is then proportional to $n + \sigma p$ where

$$\sigma = \frac{a + LK}{b}. \tag{9}$$

This optimization therefore reduces to the one analyzed in the previous section.

7. Amortized cost of many signatures

Using the OTS only once is inefficient, since the sender needs to sign the original hash image $H(R)$ using a conventional digital signature (e.g., DSS). Using Merkle's tree scheme [12] as illustrated in Fig. 3, we can sign an arbitrarily large number of messages with only one conventional signature. However, the incremental cost of generating and verifying an additional signature increases logarithmically with the number of signatures.

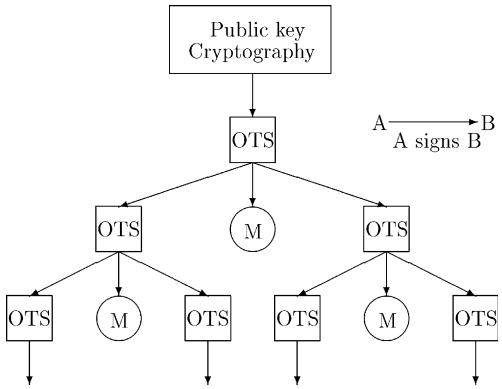

Fig. 3. Tree scheme for maintaining signature vectors.

In the tree scheme, one constructs a tree of signature nodes. Each signature node has a vector for signing each of its children as well as a single message. The root node is signed by conventional means—i.e., using a digital signature key.

One of the built-in features of Merkle's OTS tree construct is the ability to unambiguously order signatures. This is a very useful service in many application domains. For example, it is imperative in a military environment to establish strict causality of commands (and signed orders in general). Failure to do so can have disastrous consequences since re-ordered messages can lead to devastating mistakes on the battlefield. In the civilian realm, signature ordering is very important in electronic commerce, among other fields.

Consider a binary tree such that each node has three vectors and suppose that we choose n and p such that $C(n,p) > 2^b$. We would like to compute the cost, in hashes, of generating and verifying a signature at any given depth d.

To generate the signature, one needs to do a OTS of the message (requiring n hashes). Assuming the signer can cache the tree, no further computation is required.

Verification requires one to perform p hashes to verify the current node's signature of the message, and an additional p hashes to verify the parent node's signature of the current node. If the receiver has cached the tree, no further computation is required; otherwise, an additional $(d-1)p$ hashes are required.

In contrast to the single signature case, then, each of a sequence of signatures costs $n+2p$ hashes if receivers cache the signature tree, or $n+(d+1)p$ hashes if they do not. How reasonable is it to cache a signature tree? If we choose SHA as our message digest, we need to sign 160 bits of message, for which we could choose $n=165$ and $p=82$. Both signer and verifier must cache, for each node, $3n$ 160-bit numbers (the signer caches the random numbers R, the verifier caches the anchor values $H(R)$); this works out to 4950 bytes per node. This is easily supported.

In fact, receivers who cache the tree need only maintain the lowest layer of nodes, so that only about half the nodes already traversed need be kept at any time. They can prune additional information off the tree by removing the message signature vectors after they are exhausted.

8. Constructing the optimal tree

In Section 4, we showed how to choose n and p to minimize the total cost of an OTS. In this section, we will describe how to choose the parameters of a k-ary tree with a depth of d. Note in particular that the tree structure does not need to be binary. Also, we should stop somewhere in our tree; at some point the cost of using our large tree to verify an OTS is more than that of starting a new tree in which we have signed the root node by conventional means. In other words, we need to optimize the value of d.

We will try to minimize the average verification cost of one OTS. Suppose that the verifier only caches the root node of the tree. This is a reasonable assumption, especially when the signer uses the tree to sign messages for different receivers. Suppose also that the cost of verifying a traditional signature is C times more than verifying a OTS. In a k-ary tree with a depth of d, the number of messages that can be signed is

$$N=\sum_{y=1}^{d}k^{y-1}=\frac{k^d-1}{k-1}$$

and the number of OTS required

$$W=\sum_{y=1}^{d}yk^{y-1}=\frac{dk^{k+1}-(d+1)k^d+1}{(k-1)^2}.$$

In a single tree, the cost per message is

$$\frac{C+W}{N}=\frac{C+\sum_{y=1}^{d}yk^{y-1}}{\sum_{y=1}^{d}k^{y-1}}. \tag{10}$$

We try to find the smallest d such that the average cost per message is smaller than it is for $d+1$.

At optimal d,

$$\frac{C+\sum_{y=1}^{d+1}yk^{y-1}}{\sum_{y=1}^{d+1}k^{y-1}}\doteq\frac{C+\sum_{y=1}^{d}yk^{y-1}}{\sum_{y=1}^{d}k^{y-1}}$$

which is approximately equivalent to

$$C+(d+1)k^d+W\doteq k(C+W). \tag{11}$$

If we put W into its place and make necessary manipulations, our formula becomes

$$k^{d+1}\doteq(k-1)^2C+1.$$

Then, we can give the optimal tree depth d as

$$d\doteq\frac{\log[(k-1)^2C+1]}{\log k}-1. \tag{12}$$

In Fig. 4, we show the depth of a binary tree versus average normalized verification cost per message. One should choose the depth d where the average normalized cost per message is minimal.

Secondly, we would like to introduce a way to decide on the value of k. Suppose the signer caches the tree and S be the storage requirement for one random vector and its corresponding hash values. Then the storage cost per message is $(k+1)S$, which is independent of d. Suppose also that the storage cost per message is at most M. Then

$$k=\frac{M}{S}-1. \tag{13}$$

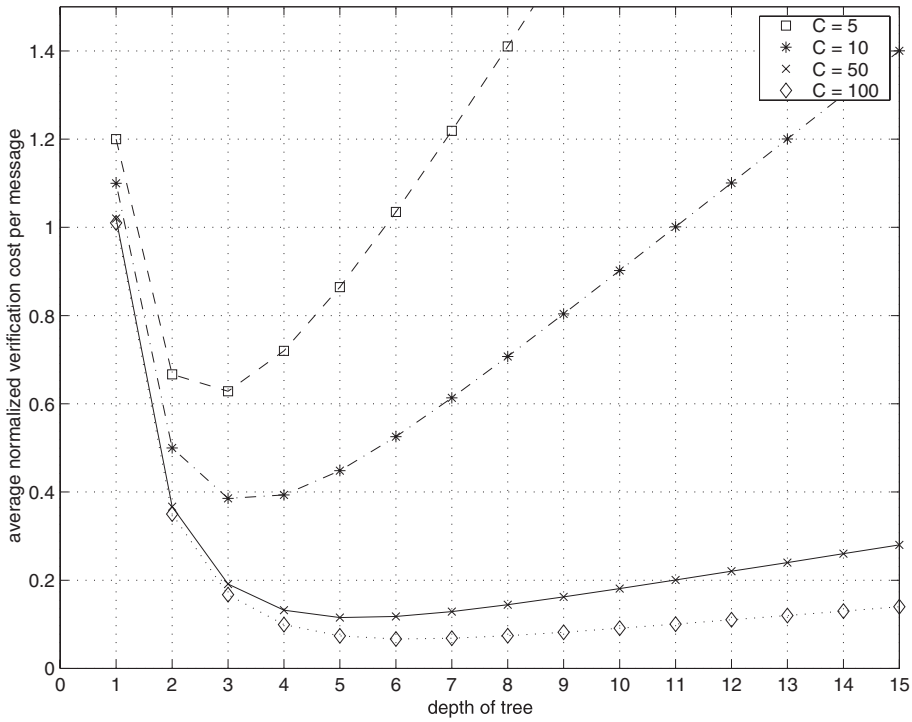

Fig. 4. Choosing the optimal depth for a binary tree.

So after estimating k with this simple formula, we can also find d by using the previous formula above.

9. Implementation and future work

In practical implementations, OTS use closely resembles that of public key signatures. For instance, in OTS, the sender distributes a certificate, which contains a public value, signed by a certification authority. The only difference is that the public value here is a sequence of hash function outputs (or a single output if these components are hashed in a single hash value), rather than a public key. Just as before, this certificate can be distributed in advance of the signed message, or it can accompany the signed message. Finally, the signature represents a transformation of a message digest of the overall message; the only differences are in the details of that transformation.

There is a clear analogy to traditional digital signature algorithms, which is summarized by the diagram below:

message digest $\Longleftrightarrow$ message digest

encryption with private key

$\Longleftrightarrow$ mapping to OTS random values

verification with public key

$\Longleftrightarrow$ verification via OTS anchors

Currently, we have implemented an OTS library which utilizes the suggested mapping and the tree structure. We will investigate the effect of changing the parameters to make it more efficient.

Subsequently, we will integrate our OTS library into specific applications that require fast digital signatures. One of the anticipated application domains is as an integrity mechanism in Active Networks. One of the basic tenets of the Active Networks concept is the use of so-called ''smart packets,'' packets that carry the means for their own handling in routers and other network entities.

This paradigm immediately raises a number of security issues, data integrity and origin authentication being chief among them. For this reason, we maintain that ultra-fast digital signatures are an absolute must for Active Networks to be practical.

The results of our research will also be of interest to an intrusion detection system. Authenticating the source and contents of a response request is fundamental to the survivability of the systems at hand. At the same time, responses must also be executed in a timely fashion and not be allowed to queue indefinitely. We expect that OTS will provide a way for components of intrusion detection systems to quickly and efficiently establish the integrity of the messages they exchange.

10. Conclusion

We have provided a theoretical foundation for evaluating the efficiency and compactness of one-time digital signatures. This work reveals the absolute minimum complexity of computing a digital signature over a space of b-bit messages, based on the weighted costs of signature generation and verification. We have shown how this theoretical minimum can be achieved by using a simple and efficient mapping between messages and subsets of random numbers. We have demonstrated how this family of OTS schemes fits into an elegant protocol for amortizing the cost of OTSs over many messages. Finally, we have provided a theoretical foundation for evaluating the efficiency of the OTS tree structure based on the weighted costs of traditional and OTSs.

References

[1] R.L. Rivest, A. Shamir, L.M. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Communications of the ACM 21 (2) (1978) 120–126.
[2] National Institute for Standards and Technology, Digital Signature Standard (DSS), Federal Register 56 (169), August 30, 1991.
[3] National Institute of Standards and Technology (NIST), FIPS Publication 46-2: Data Encryption Standard, December 30, 1993.
[4] R.L. Rivest, The MD5 message-digest algorithm, Internet Request for Comments, April 1992, RFC 1321.
[5] National Institute of Standards and Technology (NIST), FIPS Publication 180: Secure Hash Standard (SHS), May 11, 1993.
[6] L. Lamport, Constructing digital signatures from a one-way function, Technical Report CSL-98, SRI International, October 1979.
[7] R.C. Merkle, Secrecy, authentication, and public key systems, Technical report, Stanford University, June 1979.
[8] R.C. Merkle, A certified digital signature, in: G. Brassard (Ed.), Proceedings of the CRYPTO 89, Lecture Notes in Computer Science, vol. 435, Springer, Berlin, 1990, pp. 218–238.
[9] D. Bleichenbacher, U.M. Maurer, Directed acyclic graphs, one-way functions and digital signatures, in: Y.G. Desmedt (Ed.), Proceedings of the CRYPTO 94, Lecture Notes in Computer Science, vol. 839, Springer, Berlin, 1994, pp. 75–82.
[10] D. Bleichenbacher, U.M. Maurer, Optimal tree-based one-time digital signature schemes, in: Proceedings of the STACS 96, Lecture Notes in Computer Science, vol. 1046, Springer, Berlin, 1996, pp. 363–374.
[11] D. Bleichenbacher, U.M. Maurer, On the efficiency of one-time digital signatures, in: Proceedings of the ASIACRYPT 96, Lecture Notes in Computer Science, vol. 1163, Springer, Berlin, 1996, pp. 145–158.
[12] R.C. Merkle, A digital signature based on a conventional encryption function, in: C. Pomerance (Ed.), Proceedings of the CRYPTO 87, Lecture Notes in Computer Science, vol. 293, Springer, Berlin, 1988, pp. 369–378.
[13] J.N.E. Bos, D. Chaum, Provable unforgeable signatures, in: E.F. Brickell (Ed.), Proceedings of the CRYPTO 92, Lecture Notes in Computer Science, vol. 740, Springer, Berlin, 1992, pp. 1–14.
[14] S. Even, O. Goldreich, S. Micali, On-line/off-line digital signatures, in: G. Brassard (Ed.), Proceedings of the CRYPTO 89, Lecture Notes in Computer Science, vol. 435, Springer, Berlin, 1990, pp. 263–277.
[15] E. Sperner, Ein satz uber untermenge einer endlichen menge, Math. Z. 27 (1928) 544–548.

Kemal Bicakci received the B.S. degree in electronics engineering from Hacettepe University, Turkey in 1996 and the M.S. degree in electrical engineering (computer networks) from University of Southern California, USA in 1999. Between January 1998 and March 2000, he worked on various security projects as a graduate research assistant in USC Information Sciences Institute.

He is now a Ph.D. student in Informatics Institute, Middle East Technical University, Turkey. His research interests include network security and applied cryptography.

Gene Tsudik is an Associate Professor in the School of Information and Computer Science at UC Irvine. He has been active in the area of Internetworking, network security and applied cryptography since 1987. He obtained a Ph.D. in Computer Science from USC in 1991 for his research on access control in Internetworks. He then moved on to IBM Research (1991–1996) and USC Information Science Institute (1996–2000). Since 2000, he has been at UC Irvine.

Over the years, his research included: Internetwork routing, firewalls, authentication, mobile network security, secure e-commerce, anonymity, secure group communication, digital signatures, key management, ad hoc network routing, and, more recently, database privacy and secure storage. He has over 70 refereed publications and 7 patents.

Brian Tung is project leader and computer scientist at the USC Information Sciences Institute. He has been working in the security area for 8 years. During that time, he has led projects on intrusion detection and response, fast digital signatures using one-way hashes, and extensible watermarking, and he has also participated in a project to add public key cryptography to the Kerberos authentication system.

He is co-chair of the Common Intrusion Detection Framework (CIDF). In that position, he has co-edited and made significant technical contributions to the specification for the Common Intrusion Specification Language (CISL).

He received his B.S. in electrical engineering and computer science from UC Berkeley in 1989, and his M.S. and Ph.D. in computer science from UCLA in 1993 and 1994, respectively. His dissertation described the mathematics behind the use of simple finite state automata for optimization and autonomous control.