On Constructing Optimal One-Time Signatures

Kemal Bicakci

METU Informatics Institute, Inonu Bulvarı, 06531, Ankara, Turkey
bicakci@ii.metu.edu.tr

Brian Tung, Gene Tsudik

USC Information Sciences Institute
4676 Admiralty Way, Marina del Rey, 90292, CA, USA
brian@isi.edu, gts@isi.edu

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

1 Introduction

Digital signatures are rapidly becoming ubiquitous in many spheres of computing. Most current techniques for generating digital signatures are based on public key cryptography, e.g., RSA or DSS [12, 3]. These, in turn, are based on complex mathematical problems such as factoring or discrete logarithms. This solid mathematical basis is both a blessing and a curse: the former because it lends itself to simple and elegant design and the latter because there is no assurance that efficient algorithms do not exist for solving the underlying mathematical problems.

One-time signatures (OTS) provide an attractive alternative to public key-based signatures since---unlike signatures based on public key cryptography---OTS is based on nothing more than a one-way function (OWF). Examples of conjectured OWFs include DES [10], MD5 [11], and SHA [9]. There is strong (albeit folkloric) evidence as to the existence of true OWFs. Furthermore, OTSs are claimed to be more efficient since no complex arithmetic is typically involved in either OTS generation or verification.

The OTS concept has been known for almost two decades. It was initially developed by Lamport [4] and enhanced by Merkle [8] and Winternitz [7]. For the sake of brevity, we do not review the history of OTS research here; we defer instead to [6, 5] for a comprehensive treatment of the subject.

One efficient OTS construction is due to Merkle [6]. (Other efficient constructions can be found in [1] [2].) To summarize the cost of Merkle's OTS construction, the signer generates n = (b+log b) random numbers and performs n OWF computations. Each verifier performs, on the average, n/2 OWF computations which yields the average total of 1.5 * (b+log b). If we were to sign a 128-bit message (e.g., an MD5 digest) an average of 202 OWF operations would be necessary.

Despite its relatively low cost and overall elegance, this 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. This open issue is precisely the topic of this 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., with multi-destination e-mail or, more generally, secure multicast. This causes us to consider a slightly different notion of optimality.

2 One-Time Signature Generalization

The question we are trying to find an answer is how many distinct messages can be signed when R contains n random numbers? For n = 1, the answer is one, and the signature is the 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 us only one 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 [13], it was shown that for any n, the domain of mapping M is greatest when we draw from all subsets containing $\lfloor n/2 \rfloor$ random numbers. 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, using Stirling's approximation and taking the base 2 log of both sides, we can see that to represent 2^b distinct values, n must satisfy

$$n - \log\sqrt{\Pi n/2} > b \tag{2}$$

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. Note that we can freely increase n and decrease p, or similarly, decrease n and increase p, as long as $C(n,p) > 2^b$. In Figure 1, we show the number of random numbers n versus the number of hashes required for verification p, for two popular message (digest) sizes.

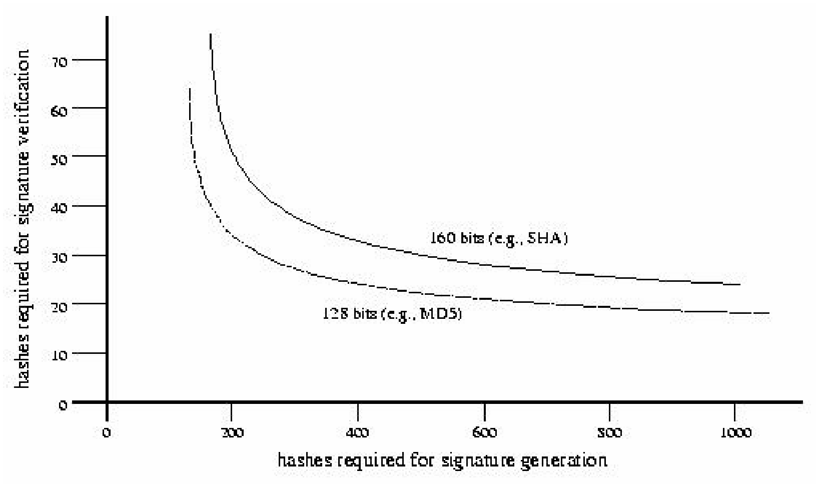

Figure 1: Signature generation/verification hash profile.

3 Cost Analysis of a Single One-Time Signature

3.1 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 one-time signature. 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 one-time signature (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 one-time signature (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 one-time signature, 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 $C(n,p) \approx C(n+\sigma,p-1)$. As a first approximation, we have, from the definition of the binomial coefficient

$$\left(\frac{n}{n-p}\right)^{\sigma} \approx \frac{n-p}{p} \qquad (3)$$

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

$$\alpha = (1-\alpha)^{\sigma+1} \qquad (4)$$

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

3.2 On-line/Off-line Case

In [2], 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 $R=\{R_1,...,R_n\}$ of n random numbers 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 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 both available bandwidth and computation power vary in a wide range. We therefore need to decide on the values of n and p with respect to bandwidth and computation power.

Now, 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/sec. And L seconds as the time required to perform one hash operation. Then

$$T = \frac{ap + bn}{K} + (p+1)L + m \tag{5}$$

In the above formula, we ignore other delays such as queuing delays. One extra L is for generating the hash of the message and m is the time for the mapping. The total time needed to verify one OTS is then proportional to $n+\sigma p$ where $\sigma = (a + LK) / b$. Fortunately, this is the same quantity we have tried to minimize in the preceding subsection. So in this case we can also use the results we have obtained previously.

4 Amortized Cost of Many Signatures

Using the one-time signature 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 a tree scheme, as in Merkle [6], 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.

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. In this treatment, we consider only a binary tree, so that each node has three vectors. 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 one-time signature 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 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, then 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.

5 Constructing the Optimal Tree

In Section 4, we showed how to choose n and p to minimize the total cost of an OTS. Now in this section we will describe how to choose the parameters of a k-ary tree with a depth of d. Note that the tree structure does not need to be binary; we can increase k. On the other hand we should stop somewhere in our tree; that is 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 and the number of OTS required are

$$\sum_{y=1}^{d} k^{y-1}, \sum_{y=1}^{d} yk^{y-1}$$

respectively. In a single tree, the cost per message is

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

We try to find the smallest d such that the average cost per message
is smaller than it is for d+1. If we use the equality of

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

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

is approximately equal to

$$C + (d+1)k^d + W \approx k(C+W)$$

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

$$d \approx \frac{\log\left[(k-1)^2 C + 1\right]}{\log k} - 1 \qquad (6)$$

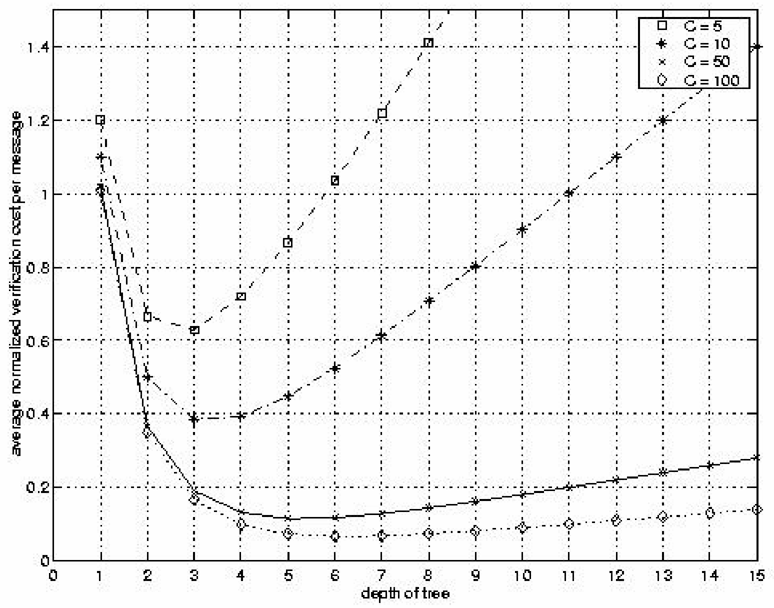

Figure 2: choosing the optimal depth for a binary tree

In Figure 2, 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.

6 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 one-time signature schemes fits into an elegant protocol for amortizing the cost of one-time signatures 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 one-time signatures.

References

[1] Jurjen N.E. Bos and David Chaum. Provable unforgeable signaturesIn Ernest F. Brickell, editor, Proc. CRYPTO 92}, pages 1-14. Springer-Verlag, 1992. Lecture Notes in Computer Science No. 740.

[2] Shimon Even, Oded Goldreich, and Silvio Micali. On-line/off-line digital signatures. In G.Brassard, editor, CRYPTO 89, pages 263--277. Springer-Verlag, 1990. Lecture Notes in Computer Science No.\ 435.

[3] National~Institute for Standards and Technology. Digital Signature Standard (DSS). Federal Register, 56(169), August 30 1991.

[4] L.Lamport. Constructing digital signatures from a one-way function. Technical Report CSL-98, SRI International, October 1979.

[5] A.Menezes, P.Van Oorschot, and S.Vanstone. Handbook of applied cryptography. CRC Press series on discrete mathematics and its applications. CRC Press, 1996. ISBN 0-8493-8523-7.

[6] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, Proc. CRYPTO 87, pages 369--378. Springer-Verlag, 1988. Lecture Notes in Computer Science No. 293.

[7] Ralph C. Merkle. A certified digital signature. In G.Brassard, editor, Proc. CRYPTO 89, pages 218--238. Springer-Verlag, 1990. Lecture Notes in Computer Science No. 435.

[8] R.C. Merkle. Secrecy, authentication, and public key systems, Stanford Univ, Jun 1979.

[9] National Institute of Standards and Technology (NIST). FIPS Publication 180: Secure Hash Standard (SHS), May 11, 1993.

[10] National Institute of Standards and Technology (NIST) FIPS Publication 46-2: Data Encryption Standard, December 30, 1993.

[11] Ronald L. Rivest. The {MD5} message-digest algorithm. April 1992. RFC 1321.

[12] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 21(2):120--126, 1978.

[13] E. Sperner. Ein satz uber untermenge einer endlichen menge. Math Z., 27, pages 544--548, 1928.