Fast Accumulated Hashing

Kaisa Nyberg

Finnish Defence Forces
Communications and Information Systems Division
E-mail: viesti@pp.kolumbus.fi

Abstract. A new non-trapdoor accumulator for cumulative hashing is introduced. It can be efficiently realized in practise using existing cryptographic hash algorithms and pseudorandom sequence generators. The memory requirement is less than in comparable signature-based solutions.

1 Introduction

The management of a cryptographic security system involves creating, time-stamping, maintaining and updating of a diversity of security related lists. Typical examples are public key directories and lists for certificate revocation, system configuration, access control and software integrity. The existing solutions are based on the use of digital signatures of a trusted third party.

In 1993 J. Benaloh and M. de Mare presented an alternative to digital signatures by introducing the principle of accumulated hashing [1]. In accumulated hashing the items are hashed together to a hash code in such a way that afterwards it is possible, for each item separately, to prove its membership in the accumulation. Benaloh and de Mare also proposed a concrete design for an accumulator based on a commutative trapdoor one-way function. But the trusted third party was still needed, although off-line, to provide the trapdoor function.

A further analysis of the requirements and the definition of accumulator was performed by K. Nyberg in [2]. Instead of commutative functions an equivalent algebraic setting for accumulators in terms of commutative semigroups was introduced. Also a construction for a new accumulator was presented. This accumulator is " absolute" in the sense that it is provably secure and not based on a trapdoor known to some third party. Consequently, it offers a fully decentralized alternative to digital signatures.

The purpose of this contribution is to improve the accumulator given in [2] and make it more efficient while preserving the good properties of the previous construction. The new accumulator has fast implementations using certain modes of operation of existing algorithms for hashing and pseudorandom sequence generation. The requirement for secure memory is less than what is needed to store an itemized list of digital signatures.

2 Accumulated Hashing

In accumulated hashing the list items are cumulatively hashed together in such a way that the accumulated hash value does not depend on the order in which the items appear on the list.

This idea was formalized in [1] by the means of commutative functions. Let A and B be sets. A function $F : A \times B \to A$ is said to be *commutative* (or *quasi-commutative* [1]) if

$$F(F(a,b),c) = F(F(a,c),b), \text{ for all } a \in A \text{ and } b,c \in B.$$

Given a seed a_0 and a list of items $b_1, \ldots, b_m$ the *accumulated hash value* a_m is computed iteratively as

$$a_k = F(a_{k-1}, b_k), \ k = 1, \ldots, m.$$

The seed a_0 may be chosen to depend, in a one-way manner, on the characteristics of the list, e.g. the system description, timestamp etc.

The accumulator proposed in [1] makes use of the trapdoor commutative function

$$F(a,b) = a^b \bmod n, \ a \in \mathbf{Z}_n, \ b \in \mathbf{Z}, \tag{1}$$

where n is a RSA modulus of a trusted third party. The trusted third party is needed only off-line to provide the RSA modulus n. The users of this accumulation system for a list $b_1, \ldots, b_m$ can compute the accumulated hash value a_m without the third party. With each item b_k the partial accumulated hash value $\overline{a}_k$, which is computed for the list containing all other items except b_k, needs to be stored. To verify the membership of b_k one simply checks that

$$F(\overline{a}_k, b_k) = a_m.$$

If the accumulator of Benaloh and de Mare is used for an application such as authentication of a list of public keys, it does not offer much improvement over the solution offered by the trusted party certificates, since with each public key the partial accumulated hash value needs to be given.

An essential security requirement for an accumulator is that it is infeasible to forge the list, i.e., to find another list with the same accumulated hash value. Although the function $F(a,b)$ defined in (1) is one-way, it is easy to construct a forged list $(c_1, \ldots, c_r)$, such that

$$a_m = a_0^{b_1 \cdots b_m} \bmod n = a_0^{c_1 \cdots c_r} \bmod n$$

for example, by choosing c_i to be products of the original items b_i. On the other hand, given $b_1, \ldots, b_m$, it is very unlikely that a randomly chosen integer c divides the product $b_1 \cdots b_m$ [1]. Therefore it is necessary as also recommended in [1] to hash or encrypt each item before taking it to the accumulator.

An open problem presented in [1] was whether it is possible to design an accumulator without a trapdoor and without a trusted party. Such an accumulator would be truly decentralized and does not rely on trusted on-line or

off-line services. A first positive answer to this problem was given in [2] where a non-trapdoor accumulator was presented. The idea was to produce long hash codes b_i, for each item $i = 1, \ldots, m$, and represent them as a p-ary n-tuples $y_i = (y_{i1}, \ldots, y_{in})$, where p is a prime. The accumulated hash value is computed as the coordinatewise product *modulo* p of y_i, $i = 1, \ldots, m$. With a fixed security level, say, the probability of forgery is e^{-t}, the required length of the accumulated hash value grows as $\mathcal{O}(N \log N)$ with the number N of items on the list. Next we show how to modify this construction to improve its efficiency.

3 The New Accumulator

Let $N = 2^d$ be an upperbound to the number of items to be accumulated and let r be an integer. We assume that there is a one-way hash function h which maps bit strings of arbitrary length to bit strings of fixed length $\ell = rd$.

Let $x_1, \ldots, x_m$, $m \leq N$, be the items to be accumulated and let

$$h(x_i) = y_i, \ i = 1, \ldots, m$$

be their corresponding hash codes, which are bit strings of length $\ell = rd$. These strings are divided into r blocks of length d and we denote

$$y_i = (y_{i1}, \ldots, y_{ir}),$$

where y_{ij} is a string of bits of length d. Further, we map each item y_i to a binary string b_i of length r by replacing y_{ij} by 1, if $y_{ij} \neq 0$, and by replacing y_{ij} by 0, if y_{ij} is a string of zero bits.

In this way we have mapped each item x_i to a bit string $b_i = (b_{i1}, \ldots, b_{ir})$ of length r, which in the case of an ideal hash function h can be considered as values of r independent binary random variables, for which the probability of taking the value 0 is equal to 2^{-d}.

The accumulated hash code $(a_1, \ldots, a_r)$ is computed as a coordinatewise product *modulo* 2 of the binary r-tuples b_i, $i = 1, \ldots, m$.

To verify the membership of an item x_i on a list described by the accumulated hash value $a = (a_1, \ldots, a_r)$ one computes $y_i = h(x_i)$, forms the corresponding $b_i = (b_{i1}, \ldots, b_{ir})$ and cheks that, for all $j = 1, \ldots, r$, whenever $b_{ij} = 0$ then $a_j = 0$.

The verification procedure is essentially simpler than the verification in the Benaloh-de Mare accumulator where the prover also needs to provide the partial accumulator of $m - 1$ items.

Next we show that the security of the new accumulator depends in a proven way only on the randomness properties of the hash function h. We also derive estimates to what is the required size of the length of the accumulator to achieve a certain security level.

Theorem 1. *Let b_{ij} and c_j be independent binary random variables such that $Pr(b_{ij} = 0) = Pr(c_j) = 2^{-d}$, for $i = 1, \ldots, m$ and $j = 1, \ldots, r$. Let $a = (a_1, \ldots, a_r)$ be the coordinatewise product of the r-tuples $b_i = (b_{i1}, \ldots, b_{ir})$,*

$i = 1, \ldots, m$. Then the probability that, for all $j = 1, \ldots, r$, we have $c_j = 0$ only if $a_j = 0$, is equal to

$$(1 - 2^{-d}(1 - 2^{-d})^m)^r.$$

Proof. For each $j = 1, \ldots, r$ the probability that $c_j = 0$ and $a_j = 1$ equals

$$2^{-d}(1 - 2^{-d})^m.$$

□

Assuming that h produces truly random hash codes, and recalling that $N = 2^d$ is the upperbound to the number of items to be accumulated, we get by the theorem that the probability of finding a forged item to a list described by an accumulated hash value $a = (a_1, \ldots, a_r)$ can be estimated as follows

$$(1 - 2^{-d}(1 - 2^{-d})^m)^r \leq (1 - \frac{1}{N}(1 - \frac{1}{N})^N)^r \approx (1 - \frac{1}{Ne})^r \approx e^{-\frac{r}{Ne}}.$$

4 Requirements

For some applications, it is sufficient that the hash function is one-way. In other applications non-repudiation may be essential and then the hash function needs to be collision resistant. For example, if the items to be accumulated are documents belonging to different users and the accumulator is used for time-stamping, then it is desirable that the users cannot find two documents with the same hash code.

Formally, the requirements for the hash function used in combination with the new accumulator are the same as for any cryptographic hash function. However, the existing hash functions are designed for the purposes of digital signatures and produce short hash codes of 128 - 160 bits. By the theorem the length ℓ of the hash codes needed by the new accumulator is determined by the formula

$$\ell = Net \log N$$

where N is the maximum number of items, e is Neper´s number, and e^{-t} is the probability of forgery. For example with $N = 1000$ and $t = 100$ we need to compute 2.8 megabits of hash code for each item.

The length r of the accumulated hash value is shorter by the factor of $\log N$. It can be estimated by $r = Net$. This is how much memory is required to store the accumulated hash value. It is less than in a traditional directory solutions. An itemized list, with a digital signature as appendix to each item, takes at least $sN \log N$ bits of memory, where s is the length of the signature and $\log N$ is a lower bound to the length of the description of an item.

A straightforward implementation of a required "long" hash function using existing cryptographic algorithms could be as follows. The item is first hashed to a short hash code which is then fed as a seed to a binary random sequence generator. From this seed as many pseudorandom bits as needed can be generated for the long hash code.

References

1. J. Benaloh and M. de Mare, One-way accumulators: a decentralized alternative to digital signatures. In: Advances in Cryptology - Eurocrypt'93 (ed. by T. Helleseth), Lecture Notes in Computer Science 765, Springer-Verlag, Heidelberg 1994, 274–285.
2. K. Nyberg, Commutativity in cryptography. In: Proceedings of the First International Workshop on Functional Analysis at Trier University, Walter de Gruyter & Co, Berlin (to appear).