A Composition Theorem for Universal One-Way Hash Functions

Victor Shoup

IBM Zurich Research Lab, Säumerstr. 4, 8803 Rüschlikon, Switzerland
sho@zurich.ibm.com

Abstract. In this paper we present a new scheme for constructing universal one-way hash functions that hash arbitrarily long messages out of universal one-way hash functions that hash fixed-length messages. The new construction is extremely simple and is also very efficient, yielding shorter keys than previously proposed composition constructions.

1 Introduction

In this paper we consider the problem of constructing *universal one-way hash functions* (UOWHFs).

The notion of a UOWHF was introduced by Naor and Yung [NY89]. A UOWHF is a keyed hash function with the following property: if an adversary chooses a message x, and then a key K is chosen at random and given to the adversary, it is hard for he adversary to find a different message $x' \neq x$ such that $H_K(x) = H_K(x')$.

As a cryptographic primitive, a UOWHF is an attractive alternative to the more traditional notion of a *collision-resistant hash function* (CRHF), which is characterized by the following property: given a random key K, it is hard to find two different messages x, x' such that $H_K(x) = H_K(x')$.

A reasonable approach to designing a UOWHF that hashes messages of arbitrary and variable length is to first design a *compression function*, that is, UOWHF that hashes fixed-length messages, and then design a method for composing these compression functions so as to hash arbitrary messages. In this paper, we address the second problem, that of composing compression functions. The main technical problem in designing such composition schemes is to keep the key length of the composite scheme from getting too large.

This composition problem was studied in some detail by Bellare and Rogaway [BR97]. They proposed and analyzed several composition schemes.

In this paper, we propose and analyze a new composition scheme. This scheme is extremely simple, and yields shorter keys than previously proposed schemes.

We also suggest an efficient and concrete implementation based on our composition technique, using a standard "off the shelf" compression function, like SHA-1, under the weak assumption of second preimage collision resistance.

2 UOWHFs versus CRHFs

A UOWHF is an attractive alternative to a CRHF because

(1) it seems easier to build an efficient and secure UOWHF than to build an efficient and secure CRHF, and
(2) in many applications, most importantly for building digital signature schemes, a UOWHF is sufficient.

As evidence for claim (1), we point out the recent attacks on MD5 [dBB93,Dob96]. We also point out the complexity theoretic result of Simon [Sim98] that shows that there exists an oracle relative to which UOWHFs exist but CRHFs do not. CRHFs can be constructed based on the hardness of specific number-theoretic problems, like the discrete logarithm problem [Dam87]. Simon's result is strong evidence that CRHFs cannot be constructed based on an arbitrary one-way permutation, whereas Naor and Yung [NY89] show that a UOWHF can be so constructed.

As for claim (2), one of the main applications of collision resistant hashing is digital signatures. The idea is to create a short "message digest" that can then be signed using a signature algorithm that needs to work only on short messages. As pointed out by Bellare and Rogaway [BR97], a UOWHF suffices for this. To sign a message x, the signer chooses a key K for a UOWHF H, and produces the signature $(K, \sigma(K, H_K(x)))$, where σ is the underlying signing function for short messages. For some UOWHFs, the key K can grow with the message length—indeed, the scheme we propose here has a key that grows logarithmically with the message length. This can lead to technical difficulties, since then the message we need to sign with σ can get too large. One solution to this problem is to instead make the signature $(K, \sigma(H_{K'}(K), H_K(x)))$, where K' is a UOWHF key that is part of the signer's public key. This is a somewhat simpler solution to this problem than the one presented in [BR97], and we leave it to the reader to verify the security of this composite signature scheme.

Naor and Yung [NY89] in fact show how to build a secure digital signature scheme based solely on a UOWHF; however, the resulting scheme is not particularly practical.

3 Previous Composition Constructions

We briefly summarize here previous constructions for composing UOWHFs.

We assume we have UOWHF H that maps strings of length a to strings of length b, where $a > b$, and that H is keyed by a key K. The goal is to build from this a composite UOWHF that hashes messages of arbitrary and variable length. To simplify the discussion, we restrict our attention in this section to the problem of hashing long, but fixed-length messages. There are general techniques to deal with variable length messages (see [BR97]).

The simplest construction is the *linear hash*. Let $m = a - b$. Suppose the message x consists of l blocks $x_1, \ldots, x_l$, where each block is an m-bit string.

Then using l keys $K_1, \ldots, K_l$ for H, and an arbitrary b-bit "initial vector" h_0, we compute h_i for $1 \leq i \leq l$ as $h_i = H_{K_i}(h_{i-1} \circ x_i)$, where "$\circ$" denotes concatenation. The output of the composite hash function is h_l.

The security of this scheme is analyzed in detail in [BR97]. Note that we need to use l independent keys $K_1, \ldots, K_l$. If we use instead just a single key, the resulting scheme does not necessarily preserve the UOW property of the compression function. This situation is quite different from the situation where we are constructing a composite hash function out of a CR compression function; in that situation, the composite hash function does indeed inherit the CR property from the compression function [Dam89,Mer89].

Although the linear hash is quite simple, it is not very attractive from a practical point of view, as the key length for the composite scheme grows linearly with the message length.

If the keys for the compression function are longer than the output length b of the compression function, then a variant of the linear hash, the *XOR linear hash* [BR97], yields somewhat shorter, though still linear sized keys. In this scheme, we use a single key K for the compression function H, and in addition, the key of the composite scheme has l "masks" $M_1, \ldots, M_l$, each of which is a random b-bit string. The scheme is then the same as the linear hash, except that we compute h_i for $1 \leq i \leq l$ as $h_i = H_K((h_{i-1} \oplus M_i) \circ x_i)$.

As pointed out by Naor and Yung [NY89], we can get composite schemes with logarithmic key size by using a *tree hash*, which is the same as a construction proposed by Wegman and Carter [WC81] for composing universal hash functions. For simplicity, assume that $a = bd$ for an integer d, and that we want to hash messages of length bd^t for some $t > 0$. Then we hash using a tree evaluation scheme, where at each level i of the tree, for $1 \leq i \leq t$, we hash bd^i bits to bd^{i-1} bits. At a given level i, we apply the compression function H d^{i-1} times, using the same key K_i. So in the composite scheme we need t keys $K_1, \ldots, K_t$.

If the keys of the compression function are long, a more efficient scheme is the *XOR tree hash* [BR97]. This is the same as the tree hash scheme, except as follows. We used a single compression function key K, and in addition, we use t "masks" $M_1, \ldots, M_t$, each of which is a random a-bit string. Whenever we evaluate the compression function at level i in the tree, we "mask" its input with M_i; that is, we compute its input as the bit-wise exclusive-or of M_i and the input used in the normal tree hash.

The new scheme we present in the next section most closely resembles the XOR linear hash, except that we re-use the masks as much as possible to minimize the key length. The key length of the new scheme is smaller (asymptotically) than the key length of the XOR tree hash by a factor of $d/\log_2 d$, while at the same requiring essentially the same amount of computation. This, combined with the fact that the new scheme is extremely simple, makes it an attractive alternative to the XOR tree hash.

4 The New Scheme

We now describe our new scheme, which is the same as the XOR linear hash, except that we get by with a smaller number of masks. Since it is not difficult to do, we describe how our scheme works for variable length messages.

Again, our starting point is a UOW compression function H that is keyed by a key K, and compresses a bits to b bits. Let $m = a - b$. We assume that a message x is formatted as a sequence of l blocks $x_1, \ldots, x_l$, each of which is an m-bit string, and we assume that the last block x_l encodes the bit length of x in some canonical way. The number of blocks l may vary, but we assume that $l \leq L$ for some given L.

The key for the composite scheme consists of a single key K for H, together with a number of "masks," each of which is a random b-bit string. We need $t+1$ masks $M_0, \ldots, M_t$, where $t = \lceil \log_2 L \rceil$.

To define the scheme, we use the function $\nu_2(i)$ which counts the number of times 2 divides i, i.e., for $i \geq 1$, $\nu_2(i)$ is the largest integer ν such that 2^ν divides i.

The hash function is defined as follows. Let h_0 be an arbitrary b-bit string. For $1 \leq i \leq l$, we define $h_i = H_K((M_{\nu_2(i)} \oplus h_{i-1}) \circ x_i)$. The output of the composite hash is h_l.

Theorem 1. *If H is a UOWHF, then the above composite scheme is also a UOWHF.*

The remainder of this section is devoted to a proof of this theorem. We show how an adversary A that finds collisions in the composite scheme can be turned into an adversary A' that finds collisions in the compression function H. This reduction is quite efficient: the running time of A' is essentially the same as that of A, and if A finds a collision with probability ϵ, then A' finds a collision with probability about ϵ/L.

We begin with an auxiliary definition. Let x be an input to the composite hash function; for $1 \leq i \leq l$, define $S_i(x)$ be the first b bits of the input to the ith application of the compression function H. The definition of $S_i(x)$ depends, of course, on the value of the composite hash function's key, which will be clear from context.

Consider the behavior of adversary A. Suppose its first message x—the "target" message—is formatted as $x_1, \ldots, x_l$, and its second message x' that yields the collision is formatted as $x'_1, \ldots, x'_{l'}$.

For this collision, we let δ be the smallest nonnegative integer such that $S_{l-\delta}(x) \circ x_i \neq S_{l'-\delta}(x') \circ x'_i$. Since we are encoding the bit length of a message in the last message block, if the bit lengths of x and x' differ, then clearly $\delta = 0$. Otherwise, $l = l'$ and it is easy to see that δ is well defined.

The pair $S_{l-\delta}(x) \circ x_i$, $S_{l'-\delta}(x') \circ x'_i$ will be the collision on H_K that A' finds.

The adversary A' runs as follows. We let A choose its first message x. Then A' guesses the value of δ at random. This guess will be right with probability $1/L$. A' now constructs its target message as $S \circ x_{l-\delta}$, where S is a random

b-bit string. Now a random key K for the compression function H is chosen. The task of A' is to generate masks $M_0, \ldots, M_t$ such that the composite key $(K, M_0, \ldots, M_t)$ has the correct distribution, and also that $S_{l-\delta}(x) = S$. Once this is accomplished, the adversary A attempts to find a collision with x. If A succeeds, and if the guess at δ was correct, this will yield a collision for A'.

We now present a "key construction" algorithm that on input x, δ, K, S, t as above, generates masks $M_0, \ldots, M_t$ as required. The algorithm to do this is described in Figure 1.

```
    for (j ← 0; j ≤ t; j ← j + 1) status_j ← "undefined"
    S_{l−δ} ← S
    status_{ν2(l−δ)} ← "being defined"
    for (i ← l − δ − 1; i ≥ 1; i ← i − 1) {
      j ← ν2(i)
(1)   if (status_j = "undefined") {
        choose S_i as a random b-bit string
        status_j ← "being defined"
        h_i ← H_K(S_i ∘ x_i)
        i' ← i + 1; j' ← ν2(i')
        while (status_{j'} ≠ "being defined") {
          h_{i'} ← H_K((h_{i'−1} ⊕ M_{ν2(i')}) ∘ x_{i'})
          i' ← i' + 1; j' ← ν2(i')
        }
        M_{j'} ← h_{i'−1} ⊕ S_{i'}
        status_{j'} ← "defined"
      }
    }
    i' ← 1; j' ← 0
    while (status_{j'} ≠ "being defined") {
      h_{i'} ← H_K((h_{i'−1} ⊕ M_{ν2(i')}) ∘ x_{i'})
      i' ← i' + 1; j' ← ν2(i')
    }
    M_{j'} ← h_{i'−1} ⊕ S_{i'}
    status_{j'} ← "defined"

    for (j ← 0; j ≤ t; j ← j + 1)
      if (status_j = "undefined") choose M_j as a random b-bit string
```

Fig. 1. Key Construction Algorithm

We can describe the algorithm at a high level as follows. During the course of execution, each mask M_j, for $0 \leq j \leq t$, has a status, $status_j$, where the status is one of the values "undefined," "being defined," or "defined." Initially, each status value is "undefined." As the algorithm progresses, the status of a

mask changes first to "being defined," and finally to "defined," at which point the algorithm actually assigns a value the mask.

The algorithm starts at block $l - \delta$, and assigns the value S to $S_{l-\delta}$, where in general, S_i represents the value of $S_i(x)$ for $1 \leq i \leq l$. The algorithm sets the status of mask $\nu_2(l - \delta)$ to "being defined." Now the algorithm considers blocks $l - \delta - 1, l - \delta - 2, \ldots, 1$ in turn. When it reaches block i in this "right to left sweep," it looks at the status of mask $j = \nu_2(i)$. As we shall prove below, the status of this mask j is never "being defined" at this moment in time. If the status is "defined," it skips to the next value of i. If the status is "undefined," then it chooses S_i at random, and changes the status of mask j to "being defined." The algorithm also runs the hash algorithm from "left to right," computing $h_i, h_{i+1}, \ldots, h_{i'-1}$, until it finds a block i' whose mask $j' = \nu_2(i')$ has the status "being defined." At this point, the mask j' is computed as $M_{j'} = h_{i'-1} \oplus S_{i'}$, and the status of mask j' is changed to "defined." Thus, at any point in time, there is exactly one mask whose status is "being defined," except briefly during the "left to right hash evaluation."

When the algorithm finishes the "right to left sweep," there will still be one mask whose status is "being defined," and the "left to right hash evaluation" as described above is used to define this mask, thereby converting its status to "defined." There may still be other masks whose status is "undefined," and these are simply assigned random values.

The key to analyzing this algorithm is to show that when we visit block i in the "right to left sweep," we do not encounter a mask $j = \nu_2(i)$ such that the status of mask j is "being defined." Let us make this more precise. As i runs from $l - \delta - 1$ down to 1 in the main loop, let V_i be the value of $status_j$ when the line marked (1) in Figure 1 is executed. We prove below in Lemma 1 that $V_i \neq$ "being defined" for all i. So long as this is the case, we avoid circular definitions, and it is easy to see that the algorithm constructs masks $M_0, \ldots, M_t$ with just the right distribution. Indeed, the key construction algorithm implicitly defines a one-to-one map between tuples $(K, M_0, \ldots, M_t)$ and $(K, S, S^{(1)}, \ldots, S^{(t)})$, where $S^{(1)}, \ldots, S^{(t)}$ are randomly chosen b-bit strings, and $S = S_{l-\delta}(x)$.

So the proof of Theorem 1 now depends on the following lemma.

Lemma 1. *For $1 \leq i \leq l - \delta - 1$, $V_i \neq$ "being defined."*

To prove this lemma, we need two simple facts, which we leave to the reader to verify.

Fact 1. *For any positive integers $A < B$ with $\nu_2(A) = \nu_2(B)$, there exists an integer C with $A < C < B$ and $\nu_2(C) > \nu_2(A)$.*

Fact 2. *For any positive integers $A < B$, and for any nonnegative integer $\nu < \min\{\nu_2(A), \nu_2(B)\}$, there exists an integer C with $A < C < B$ and $\nu_2(C) = \nu$.*

Now to the proof of the lemma. Suppose $V_i =$ "being defined" for some i, and let A be the largest such value of i. Then there must be a *unique* integer B

with $A < B \leq l - \delta$ such that $\nu_2(B) = \nu_2(A)$. This is the point where we set the status of mask $\nu_2(A)$ to "being defined." The uniqueness of B follows from the maximality of the choice of A.

By Fact 1, there must be an index C with $A < C < B$ and $\nu_2(C) > \nu_2(A)$. There may be several such C; among these, choose from among those with maximal $\nu_2(C)$, and from among these, choose the largest one.

We claim that $V_C =$ "defined." To see this, note that we cannot have $V_C =$ "being defined," since we chose A to be the maximal index with this property. Also, we could not have since $V_C =$ "undefined," since then we would have defined mask $\nu_2(A)$ at this point, and we would have $V_A =$ "defined."

Since $V_C =$ "defined," we must have set the status of mask $\nu_2(C)$ to "being defined" in a loop iteration prior to C. Thus, there must exist D with $C < D \leq l - \delta$ and $\nu_2(D) = \nu_2(C)$. By the way we have chosen C, we must have $D > B$.

Again by Fact 1, there exists integer E with $C < E < D$, and $\nu_2(E) > \nu_2(C)$. Again, by the choice of C, we must have $E > B$.

Finally, by Fact 2, there exists an integer F with $E < F < D$ and $\nu_2(F) = \nu_2(A)$. So we have $B < F < l - \delta$ with $\nu_2(F) = \nu_2(A)$, which is a contradiction. That completes the proof of the lemma. See Figure 2 for a visual aid.

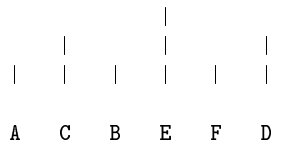


Fig. 2. From the proof of Lemma 1. The vertical lines represent the relative magnitudes of the corresponding values of ν_2.

5 A Concrete Implementation

In this section, we suggest a concrete implementation for a practical UOWHF.

Given a method for building a composite UOW hash function out of a UOW compression function, one still has to construct a UOW compression function. A pragmatic approach is to use an "off the shelf" compression function such as the SHA-1 compression function $C : \{0,1\}^{160} \times \{0,1\}^{512} \to \{0,1\}^{160}$. The assumption we make about C is that it is *second preimage collision resistant*, i.e., if a random input (S, B) is chosen, then it is hard to find different input $(S', B') \neq (S, B)$ such that $C(S, B) = C(S', B')$. This assumption seems to be much weaker than assumption that no collisions in C can be found at all (which as an intractability assumption is not even well defined). Indeed, the techniques used to find collisions in MD5 [dBB93,Dob96] do not appear to help in finding second preimages.

Note that from a complexity theoretic point of view, second preimage collision resistance is no stronger than the UOW property. Indeed, if $H_K(x)$ is a UOWHF, then the function sending (K, x) to $(K, H_K(x))$ is second preimage collision resistant.

The second preimage resistance assumption on C allows us to build a UOW compression function as follows. The key is a random element $(\hat{S}, \hat{B})$ in the domain of C, and the value of the compression function on (S, B) is $C(\hat{S} \oplus S, \hat{B} \oplus B)$.

We could apply our composition construction directly to this. However, there is one small optimization possible; namely, we can eliminate $\hat{S}$ from the key.

We can now put this all together. Assume that a message x is formatted as a sequence $x_1, \ldots, x_l$ of 512-bit blocks, where the last block encodes the bit length of x. Let L be an upper bound on l, and let $t = \lceil \log_2 L \rceil$. The key for our hash function consists of a random 512-bit string $\hat{B}$, along with $t+1$ 160-bit strings $M_0, \ldots, M_t$. Then the hash of x is defined to be h_l, where h_0 is an arbitrary 160-bit string, and $h_i = C(h_{i-1} \oplus M_{\nu_2(i)}, x_i \oplus \hat{B})$ for $1 \leq i \leq l$.

Our analysis shows that this hash function is UOW, assuming C is second preimage collision resistant.

References

[BR97] M. Bellare and P. Rogaway. Collision-resistant hashing: towards making UOWHFs practical. In *Advances in Cryptology–Crypto '97*, 1997.

[Dam87] I. Damgard. Collision free hash functions and public key signature schemes. In *Advances in Cryptology–Eurocrypt '87*, 1987.

[Dam89] I. Damgard. A design principle for hash functions. In *Advances in Cryptology–Crypto '89*, 1989.

[dBB93] D. den Boer and A. Bosselaers. Collisions for the compression function of MD5. In *Advances in Cryptology–Eurocrypt '93*, pages 293–304, 1993.

[Dob96] H. Dobbertin. The status of MD5 after a recent attack. *RSA Laboratories' CryptoBytes*, 2(2), 1996. The main result of this paper was announced at the Eurocrypt '96 rump session.

[Mer89] R. Merkle. One way hash functions and DES. In *Advances in Cryptology–Crypto '89*, 1989.

[NY89] M. Naor and M. Yung. Universal one-way hash functions and their cryptographic applications. In *21st Annual ACM Symposium on Theory of Computing*, 1989.

[Sim98] D. Simon. Finding collisions on a one-way street: can secure hash functions be based on general assumptions? In *Advances in Cryptology–Eurocrypt '98*, pages 334–345, 1998.

[WC81] M. Wegman and L. Carter. New hash functions and their use in authentication and set equality. *J. of Computer and System Sciences*, 22:265–279, 1981.