Dynamic Cryptographic Hash Functions

William R. Speirs II and Samuel S. Wagstaff, Jr.

Center for Education and Research in Information Assurance and Security (CERIAS)
Department of Computer Sciences, Purdue University

Abstract. We present the *dynamic cryptographic hash function*, a new type of hash function which takes two parameters instead of one. The additional parameter specifies the internal workings and size of the digest produced. We provide formal definitions for a dynamic cryptographic hash function and security properties required for a dynamic hash function to be considered cryptographically secure. Two additional properties, *second digest resistance* and *increasing security*, are also formally defined. We also present a construction that enables a compression function to be extended to create a dynamic cryptographic hash function. Proofs of our construction's security are provided.

Keywords: Hash function, dynamic, preimage resistance, collision resistance, construction.

1 Introduction

In this paper we introduce a new type of cryptographic hash function that is a natural extension of traditional hash functions. We call this new type of hash function a *dynamic hash function*. Dynamic hash functions take a second input parameter, besides the message, that specifies the level of security required from the function. By changing the security parameter the function dynamically changes the way a digest is computed, hence the reason for the name. The dynamic change might be as simple as changing the number of rounds used for processing a message block or the size of the output of the function.

Requiring a hash function to have a security parameter is advantageous for a number of reasons. First, it allows designers to more easily test functions by scaling down the number of rounds to a manageable amount and then launching attacks against this reduced version. If attacks can be launched against a reduced version then there is the possibility of extending the attack to the full version of the function. This technique has been used numerous times in the past: [7, 9, 10], etc.

Second, if the security parameter relates to the size of the digest, as suggested in this paper, then a single dynamic hash function can be used in a number of different applications without modification. For example, the hash-then-sign paradigm that is commonly used in a number of protocols. If the signing algorithm has the ability to vary the number of bits used for the key (like RSA,

ElGamal, or Pohlig-Hellman), then why should the hash function used be restricted to a fixed size digest? If a user requires increased security by selecting a larger key size, the size of the digest should also increase to provide potentially increased security.[1]

Finally, using a dynamic hash function in a protocol makes implementation much easier when it is mandated that the size of the digest be increased. Instead of being forced to rewrite software to include the option for a new hash function, the protocol can be designed from the start with a field to specify the size of the digest. Several protocols need to be reimplemented now that attacks on MD5 [8, 19] have became prevalent. The same type of reimplemention is currently needed as systems migrate from SHA-1 to SHA-256 or SHA-512. Using a dynamic hash function and specifying in the design that the digest can change size makes reimplemention easier when attacks are discovered; the security parameter used only needs to be changed.

1.1 The Contributions of This Paper

In this paper we formally define a dynamic cryptographic hash function and the security requirements for such a function. This definition generalizes the definition for a traditional cryptographic hash function. The security requirements of a traditional cryptographic hash function are extended to a dynamic cryptographic hash function and new security requirements are provided that do not have analogs in the traditional setting. These new requirements are needed to thwart new attacks related to the different behavior of dynamic cryptographic hash functions.

We also introduce a construction that uses a compression function to create a dynamic hash function. The construction uses the security parameter to specify the size of the dynamic hash function's digest. We provide proofs that our construction is preimage resistant, collision resistant, and increasingly secure as defined for dynamic hash functions. Also, the relative speed of our construction, using the SHA-1 compression function, is compared to SHA-1, SHA-256, and SHA-512.

1.2 Overview of this Paper

This paper begins by providing a background on traditional cryptographic hash functions and the security properties a hash function needs to be considered cryptographically secure. Next, we present the dynamic cryptographic hash function and formally define the security properties a dynamic hash function needs to be considered cryptographically secure. We also define additional security properties unique to dynamic cryptographic hash functions. In Section 4 we present our construction which creates a dynamic cryptographic hash function from a compression function. The implementation details and proofs on the security of the construction provided.

[1] Increasing the size of a key or digest of a hash function does not always relate to increased security.

2 Traditional Cryptographic Hash Functions

There are different definitions in the literature for a cryptographic hash function, including [6, 12, 15, 17] and others. While superficially these definitions are different, they all define the same concept of a cryptographic hash function. For the purposes of this paper, a hash function will be defined as follows. Let $\mathbb{N}$ define the set of natural numbers and $\Sigma = \{0, 1\}$. The length of a message M being hashed in bits is n. A hash function H is a function of the form $H : \Sigma^* \rightarrow \Sigma^{l(n)}$ where Σ^* is the infinite set of all finite binary strings and $l(n)$ is a monotone increasing function $l : \mathbb{N} \rightarrow \mathbb{N}$. The running time of $l(n)$ is bounded by $\mathcal{O}(n^k)$ where k is a constant independent of n. This defines H as a function from strings of arbitrary length to strings of a length determined by a monotone increasing function. The definition closely follows the one provided in [15]. The output of H is called the digest of the message M.

2.1 Cryptographic Hash Function Properties

While the above definition describes the shape of a hash function, there is no mention of what is required of a hash function to be considered cryptographically secure. There are three informal properties a hash function must have to be considered cryptographically secure. They are enumerated below, taken from [12]:

1. *Preimage Resistance* - For essentially all pre-specified outputs, it is computationally infeasible to find any input which hashes to that output, i.e., to find any preimage x' such that $h(x') = y$ when given any y for which a corresponding input is not known.
2. *2nd-Preimage Resistance* - It is computationally infeasible to find any second input which has the same output as any specified input, i.e., given x, to find a 2nd-preimage $x' \neq x$ such that $h(x) = h(x')$.
3. *Collision Resistance* - It is computationally infeasible to find any two distinct inputs x, x' which hash to the same output, i.e., such that $h(x) = h(x')$.[2]

Unfortunately, there are deficiencies in the description of these requirements based on the lack of formal definitions for "essentially all" and "pre-specified outputs". In [17] these properties are defined formally and expanded to seven notions of security across three different attack models. For the purposes of this paper, only preimage resistance (one-wayness) and collision resistance are considered.[3] Formal definitions, like the ones provided in Section 3.1, can be found in [6] and [15].

[2] There is a free choice in both x and x' by the adversary.

[3] Collision resistance implies 2nd-preimage resistance so a formal definition is not provided. See [17] for a formal treatment of 2nd-preimage resistance.

3 Definition of a Dynamic Cryptographic Hash Function

A dynamic cryptographic hash function is the same as a traditional cryptographic hash function except that a security parameter is provided which can affect how the function works internally and the size of the output. The definition of a traditional hash function is modified to define a dynamic hash function as follows. Let $l(n)$, $u(n)$, and $d(s)$ all be monotone increasing functions $l : \mathbb{N} \to \mathbb{N}$, $u : \mathbb{N} \to \mathbb{N}$ and $d : \mathbb{N} \to \mathbb{N}$ such that $0 < l(n) \leq u(n)$ and $0 < d(s) < n$ for all $n > 1$. The running time of $l(n)$, $u(n)$, and $d(s)$ is bounded from above by a polynomial. Let $[a, b] = \{i \in \mathbb{N} : a \leq i \leq b\}$. A dynamic hash function is then defined as follows.

Definition 1. Dynamic Hash Function – *A dynamic hash function is a function H of the form: $H : \Sigma^* \times [l(n), u(n)] \to \Sigma^{d(s)}$, for some security parameter $s \in [l(n), u(n)]$.*

One should note that a dynamic cryptographic hash function creates a family of traditional cryptographic hash functions each meeting the properties of a traditional cryptographic hash function. Since a dynamic cryptographic hash function takes a second parameter that can potentially vary the digest's size and how the function is computed, the traditional properties must be modified appropriately.

3.1 Traditional Properties for Dynamic Hash Functions

Before defining preimage and collision resistance for a dynamic hash function, a dynamic hash function family must be defined. The reason for this is that for both preimage and collision resistance one could imagine a probabilistic polynomial time algorithm which has a table of (message, security parameter, digest) triples encoded in the algorithm for a specific dynamic hash function. When asked to compute either the preimage of a given digest, or a collision for the hash function, the algorithm would search the table, in polynomial time, for either a preimage or a collision and output accordingly [16]. However, defining a function family as an infinite family of finite sets of hash functions prevents such an algorithm from succeeding for all hash functions in the family, because there are infinitely many.

Definition 2. Dynamic Hash Function Family – *A dynamic hash function family $\mathcal{H}$ with security parameter interval $[l(n), u(n)]$ is an infinite family of finite sets $\mathcal{H} = \{H_n\}_{n=1}^{\infty}$ where the members of H_n are called "instances" of size n. An instance $I \in H_n$ is a triple,*

$$I = (h_n, D_n, R_n),$$

where $D_n = \Sigma^n$, $R_n \subseteq \Sigma^{d(s)}$ for some $s \in [l(n), u(n)]$, and $h_n : D_n \times [l(n), u(n)] \to R_n$ so that for all $s \in [l(n), u(n)]$, $h_n(\cdot, s) : D_n \to \Sigma^{d(s)}$.

Three requirements are imposed on the dynamic hash function family $\mathcal{H}$.

1. H_n is accessible, that is, there is a probabilistic polynomial time algorithm, which on input n outputs an instance chosen uniformly from H_n.
2. D_n is samplable, that is, there is a probabilistic polynomial time algorithm, which on input I selects an element uniformly from D_n.
3. h_n is polynomial time computable, that is, on input I, and $x \in D_n$ there is a polynomial time algorithm (polynomial in n and in $|x|$) that computes $h_n(x)$.

Preimage Resistance for Dynamic Hash Functions Analogous to the definition of preimage resistance for traditional hash functions, preimage resistance for dynamic hash functions can only be defined once an appropriate definition for a preimage finder is given.

Definition 3. Dynamic Preimage Finder – *A dynamic preimage finder M for a dynamic hash function family $\mathcal{H}$ is a probabilistic polynomial time algorithm that on input n and $h_n(x,s) \in R_n$, where $(h_n, D_n, R_n) \in H_n$ and x is selected randomly from D_n, outputs either "?" or an element $M(n, h_n(x,s)) \in D_n$ such that $h_n(M(n, h_n(x,s)), s) = h_n(x,s)$.*

Definition 4. Preimage Resistant Dynamic Hash Function Family – *A dynamic hash function family $\mathcal{H}$ is preimage resistant if for all preimage finders M, for all polynomial Q, and for all sufficiently large n*

$$Pr\{h_n(M(n, h_n(x,s)), s) = h_n(x,s)\} < \frac{1}{Q(n)}$$

where the probability is taken over all $h_n \in H_n$, $x \in D_n$ and the random choices of M.

This definition states that the probability of an adversary successfully finding a message that will hash to a given digest is bounded by the reciprocal of any polynomial in n.

Collision Resistance for Dynamic Hash Functions Like preimage resistance, collision resistance can only be defined for dynamic cryptographic hash functions once an appropriate definition for a dynamic collision string finder is given.

Definition 5. Dynamic Collision String Finder – *A dynamic collision string finder F for a dynamic hash function family $\mathcal{H}$ is a probabilistic polynomial time algorithm that on input n, a function $h_n \in H_n$ and $s \in [l(n), u(n)]$ outputs either "?" or a pair $x, x' \in D_n$ with $x \neq x'$ and $h_n(x,s) = h_n(x',s)$.*

Definition 6. Collision Resistance Dynamic Hash Function Family – *A dynamic hash function family $\mathcal{H}$ is collision resistant if for all collision string finders F, for all polynomials Q, and for all sufficiently large n*

$$Pr\{F(n, h_n, s) \neq \text{"?"}\} < \frac{1}{Q(n)}$$

where the probability is taken over all $h_n \in H_n$ and the random choices of F.

This definition states that the probability of an adversary successfully finding two message that will hash to the same digest is bounded by the reciprocal of any polynomial in n.

3.2 Additional Properties for Dynamic Hash Functions

Since the way in which the digest is computed can change and the size of the digest can be arbitrary, a dynamic hash function must have additional properties to be considered cryptographically secure. Having these properties prevent simple and potentially dangerous constructions that use naive modifications of traditional hash functions to create a dynamic hash function. The additional properties of second digest resistance, and increasing security are defined next.

Second Digest Resistance Second digest resistance states that that it is computationally infeasible to compute the digest of a message and security parameter using the digest of the same message and a different security parameter. This property indicates the digest of some message and security parameter is generated independently of all other digests of the same message and different security parameters. Before the formal definition for second digest resistance can be stated, the definition of a dynamic digest generator must be given.

Definition 7. Dynamic Digest Generator – *A dynamic digest generator G for a dynamic hash function family $\mathcal{H}$ is a probabilistic polynomial time algorithm that on input n, $s \in [l(n), u(n)]$, $s' \in [l(n), u(n)]$, and $h_n(x, s) \in R_n$, where $s \neq s'$, $(h_n, D_n, R_n) \in H_n$, x is selected randomly from D_n, outputs either "?" or an element $G(n, s, s', h_n(x, s)) \in R_n$ such that $G(n, s, s', h_n(x, s)) = h_n(x, s')$.*

Definition 8. Second Digest Resistant Dynamic Hash Function Family – *A dynamic hash function family $\mathcal{H}$ is second digest resistant if for all dynamic digest generators G, for all polynomials Q, and for all sufficiently large n*

$$Pr\{G(n, s, s', h_n(x, s)) = h_n(x, s')\} < \frac{1}{Q(n)}$$

where the probability is taken over all $h_n \in H_n$, $x \in D_n$, $s \in [l(n), u(n)]$, $s' \in [l(n), u(n)]$, and the random choices of G.

This property ensures, among other things, that a dynamic cryptographic hash function is not constructed by concatenating or truncating a standard cryptographic hash function. One motivation for this property is to protect key generation algorithms that use a dynamic hash function. It is insecure for the 128-bit digest of message to be a substring of the 256-bit digest of the same message when using this function to generate cryptographic keys.

Suppose Alice wants to securely communicate with Bob and Carol. Alice and Bob will communicate using a 128-bit key while Alice and Carol will use a 256-bit key. Suppose Alice creates both keys by hashing the current time concatenated with some random data. If Alice creates the 128-bit and 256-bit

keys immediately after each other, using the construction above, there is a very good chance that the 128-bit key will simply be the 256-bit key truncated to 128 bits if the granularity of the clock is too large. With minimal effort Carol can compute the key Alice made for Bob..

While one might argue that protocol designers and implementers should be aware that such an attack might be possible, it is all too often that such attacks are overlooked. Instead, we argue the burden of having this property should be placed on the function designer. Such a requirement is relatively easy to achieve, assuming a preimage and collision resistant hash function, by concatenating the value of the security parameter to the message.

Increasing Security Since dynamic hash functions contain a security parameter that changes how the digest of a message is computed, it is desirable to be able to say something about the security of the function with respect to the security parameter. The obvious relationship is simply that as you increase the value of the security parameter it becomes harder to break the function with respect to Definitions 4, 6, and 8. The notion of increasing security is formally defined below.

Definition 9. Increasingly Secure Dynamic Hash Function Family – *A dynamic hash function family $\mathcal{H}$ is increasingly secure if for all dynamic preimage finders M, for all dynamic collision string finders F, for all dynamic digest generators G, and for all sufficiently large n*

$$Pr\{h_n(M(n, h_n(x,s)), s) = h_n(x,s)\} \geq Pr\{h_n(M(n, h_n(x,s{+}1)), s{+}1) = h_n(x,s{+}1)\}$$

and

$$Pr\{F(n, h_n, s) \neq \text{“?”}\} \geq Pr\{F(n, h_n, s+1) \neq \text{“?”}\}$$

and

$$Pr\{G(n, s, s', h_n(x,s)) = h_n(x,s')\} \geq Pr\{G(n, s, s'{+}1, h_n(x,s)) = h_n(x,s'{+}1)\}$$

where $s \neq s'$, $s \neq s'+1$, and the probability is taken over all $h_n \in H_n$, $x \in D_n$, $\{s, s+1, s', s'+1\} \in [l(n), u(n)]$, and the random choices of M, F, and G respectively.

This definition states that as you increase the value of the security parameter the probability of a successful attack against preimage resistance, collision resistance, and second digest resistance does not become larger.

While in the concrete model this is extremely difficult, if not impossible, to prove, usually certain assumptions can be made about the underlying pieces used to build the function that allow such a property to be proved. While these assumptions cannot usually be realized in practice, such as the existence of random oracles, proving it true in a theoretical model results in a stronger function than functions where it is not true.

4 A Dynamic Hash Function Construction

In this section we present a construction that creates a dynamic cryptographic hash function from a compression function. This construction is similar to the Merkle-Damgård construction except that it incorporates a security parameter that changes the resulting digest's size.[4] Our construction has the ability to securely create a digest that is larger or smaller than the output of the compression function.

In [3] Dunkelman and Biham present HAIFA, or a Hash Iterative Framework. Part of this framework explains a method for creating a variable size hash function which can produce digests of sizes less then or equal to that of the underlying compression function's output size. Two of the ideas discussed in HAIFA are used to create the dynamic cryptographic hash function construction presented in this paper. HAIFA however has no provision for creating a digest with a size larger then the output of the underlying compression function. This construction extends HAIFA by providing a method for securely creating digests of sizes larger than the output of the underlying compression function. Also, rigorous proofs for the security properties defined earlier are provided.

4.1 Construction Description

Our construction modifies the Merkle-Damgård construction in three ways to create a dynamic cryptographic hash function. A block diagram of our construction is provided in Figure 1, where m_i is the i^{th} block of the message and y_i is the i^{th} output of the compression function.

The first change is that the initial value depends on the security parameter. The initial value is generated by creating a "message block" that is the concatenation of a specified initial value, the security parameter as a 32-bit binary number, and enough zeros to pad the message block to the required size. The resulting value $IV_s = g(IV, IV \parallel s \parallel 0 \cdots 0)$ is the initial value for a digest with security parameter s. The same method is used in [3] to help ensure that two digests of different sizes computed from the same message will not result in one being the truncation of the other. This method is also used by SHA-224 and SHA-384 so that the digest of a message will not be the truncation of SHA-256 and SHA-512 respectively [14].

The second change to the Merkle-Damård construction is in the way that messages are padded. In our construction a single 1 bit is concatenated to the end of the message followed by enough 0 bits to allow for the message's size as a 64-bit number and the security parameter as a 32-bit number to be concatenated. The suffix of all messages will be of the form: $10 \cdots 0 \parallel |M| \parallel s$. This change was also outlined in [3].

The final change allows the size of the digest to be larger then the size of the output of the compression function, extending HAIFA. This is accomplished

[4] The security parameter could also affect the internal workings of the underlying compression function (like the number of rounds used); however, we do not investigate this case.

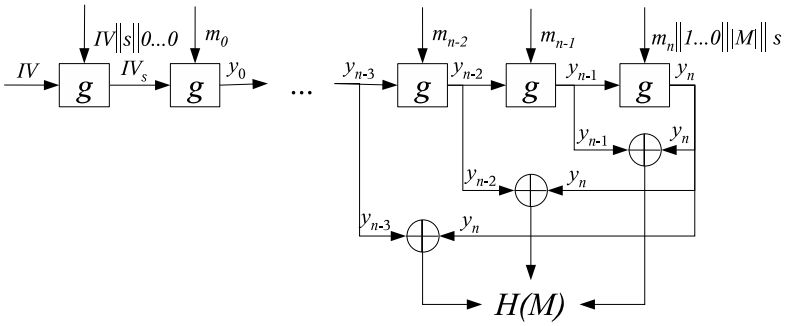

Fig. 1. The dynamic hash function construction with multiple XORs.

by XORing the output of the final chaining value with the previous chaining values. Figure 1 shows the last three chaining values used: $y_{n-3}, y_{n-2}, y_{n-1}$. Each of these chaining values is XORed with the final chaining value, y_n, and concatenated together to form the digest before truncation: $y_{n-3} \oplus y_n \parallel y_{n-2} \oplus y_n \parallel y_{n-1} \oplus y_n$. Truncation is performed, if needed, to form a digest of the desired size. Truncation is always done on the most significant bits of the result.

Security Parameter Bounds In our construction the size of the digest $d(s)$ is simply the security parameter provided: $d(s) = s$. As for all dynamic hash functions, the range of possible security parameters must be specified. Our construction specifies that bits are truncated from the result until the desired digest size is met. Therefore, there is no restriction on how small a digest could theoretically be, except that one must exist. Hence, the theoretical limit for lower bound is $l(n) = 1$. Let $c = |g(\cdot)|$ or the size of the output of the compression function. We recommend that $l(n) \geq c \geq 256$. The upper bound on the range of possible digest sizes is bounded by the ability to encode the digest size in a message block with the initial value. Let $b = |m| > 97$ bits, where m is a message block that is processed by the compression function g. Let n be the size of the message M in bits. The function $u(n)$ is then defined as follows:

$$u(n) = \begin{cases} c, & 1 \leq n \leq b - 97 \\ \lceil \frac{n}{b} \rceil c, & b - 97 < n \leq \left(\frac{2^{32}}{c} - 1 \right) b \\ 2^{32}, & \left(\frac{2^{32}}{c} - 1 \right) b < n \end{cases}$$

The upper bound on the security parameter is defined this way to ensure that the message is long enough to produce enough chaining values to create a digest of the requested size. Since the security parameter is encoded as a 32-bit number in the padding algorithm, the maximum value it can be is 2^{32}.

4.2 The Impact on Implementations

The most invasive and cumbersome implementation change is the need to save chaining values while the message is being computed. The security parameter dictates how many chaining values need to be saved to compute the digest. If the entire message size is known before hashing begins, then only those chaining values required need to be saved. However, if the size of the message is not know before hashing, as is often the case with streaming data, an appropriately sized array can store the last set of chaining values. These chaining values are replaced one by one as new chaining values are computed. While this change does require a bit of re-tooling with respect to current implementations, it is not too costly because these values must be computed already.

Our construction was implemented along with the SHA-1, SHA-256, and SHA-512 functions to compare the relative speed. The implementation of these functions is not optimal with respect to speed; however, they are all optimized in the same fashion providing a good relative comparison. In the case of our construction, the underlying compression function used was the same as SHA-1. Our construction, when producing a 160-bit digest, is only slightly slower then the SHA-1 algorithm. This result is not surprising because our construction must process an extra message block and also performs additional work with respect to padding, saving chaining values and combining them to produce the digest. What is most interesting is the difference in time with respect to the 256-bit and 512-bit digests. To create a digest for 1 MB of data, SHA-256 took 14 clock ticks and SHA-512 required 21.9; whereas, our construction took only 7.7 and 8.0 for respective digest sizes. This is not too surprising because the only additional work for our construction to create a larger digest is to save more intermediate values and combine them at the end.

4.3 The Security of This Construction

Our construction, under certain assumptions, has the required properties of preimage resistance, collision resistance, second digest resistance, and increasing security making it a cryptographically secure dynamic hash function. Theorems and a conjecture are provided for each property that state the assumptions of the underlying compression function to result in the construction having the required properties.

Preimage Resistance

Theorem 1. *If the underlying compression function used in the dynamic hash function construction is preimage resistant, then the dynamic hash function construction is preimage resistant as defined in Definition 4.*

Proof. Up to the point y_{n-2} the construction is exactly the same as the Merkle-Damgård construction, which is preimage resistant if the underlying compression

function is preimage resistant [13]. Since the value of y_{n-1} cannot be forced to result in a specific value, the only way to force the digest to a specific value is to force y_n to a specific value. By the properties of XOR, there is only one value for y_n that will force the digest to a specific digest given a fixed y_{n-1}. However, finding an m_n that forces y_n to that specific value breaks the assumption that g is a preimage resistant compression function.

Since the digest is constructed by concatenating chaining values, once they are XORed with the final value y_n, it is clear that this same argument applies to each piece of the overall digest independently. Since the argument is true for each piece independently, concatenating them together does not violate the argument. Therefore, the entire construction is preimage resistant under the stated assumption. □

Collision Resistance

Theorem 2. *If the underlying compression function used in the dynamic hash function construction is both preimage resistant and collision resistant, then the dynamic hash function construction is collision resistant as defined in Definition 6.*

Proof. Since there are two different variables that can be changed to cause a collision, the first $n-1$ message blocks or block m_n, there are four possible combinations for modifying these blocks. Let m_* be all of the $n-1$ message blocks up to block m_n. Let a $'$ denote a message block(s) that is modified from the original. One of these cases, $H(m_* \parallel m_n) = H(m_* \parallel m_n)$, is not a collision because the message does not change. The remaining three cases are investigated.

CASE 1: $H(m_* \parallel m_n) = H(m_* \parallel m'_n)$
In this case none of the first $n-1$ message blocks are modified to cause a collision. Therefore, a collision can only be caused by modifying block m_n. The only way to cause a collision is for an attacker to find two message blocks m_n and m'_n such that the resulting values y_n and y'_n are the same. This can only be done if the underlying compression function is not collision resistant. Therefore, the only way to cause a collision in this case is by breaking our assumption that the underlying compression function is collision resistant.

CASE 2: $H(m_* \parallel m_n) = H(m'_* \parallel m_n)$
In this case at least one of the first $n-1$ message blocks is modified to cause a collision. This means that the value for $y_{n-1} = y'_{n-1}$. However, up to the point y_{n-1} the construction is exactly the same as the Merkle-Damgård construction which is known to be collision resistant if the underlying compression function is collision resistant [6]. Therefore, the only way to cause a collision in this case is to break our assumption that the underlying compression function is collision resistant.

CASE 3: $H(m_* \parallel m_n) = H(m'_* \parallel m'_n)$
In this case at least one of the first $n-1$ message blocks is modified along with

the last message block to cause a collision. The only way to cause a collision in this case is for $y_{n-1} \oplus y_n = y'_{n-1} \oplus y'_n$. However, in the process of calculating y_n or y'_n, y_{n-1} or y'_{n-1} respectively must be calculated first. This leads to the situation where the message block m_n or m'_n must be picked to force a particular value for y_n or y'_n respectively. The only way to achieve this is if the underlying compression function is not preimage resistant. Therefore, the only way to cause a collision is by breaking the assumption that the underlying compression function is preimage resistant.

Since all cases result in our assumptions being broken, the theorem is proved correct for a single XOR. Since this argument is true for each piece independently, concatenating them together does not violate the argument. Therefore, the entire construction is collision resistant under the stated assumptions. □

Second Digest Resistance Second digest resistance states that given $h(x, s)$, s, and s' one cannot create the digest $h(x, s')$ from the given information. It is conjectured that that if our construction is preimage resistant, then the construction is second digest resistant.

Conjecture 1. If the dynamic hash function construction is preimage resistant, then the dynamic hash function construction is second digest resistant as defined in Definition 8.

It is clear that preimage resistance is needed. If the construction were not preimage resistant, an attacker could compute the inverse of the construction, discover x, and compute $h(x, s')$. However, this can only be done if the construction were not preimage resistant.

The difficulty in proving that our construction is second digest resistant is the possibility of some other method for attacking the construction that does not rely on how the construction works. Such an attack would be able to compute one digest given another without calculating the preimage. It is difficult to prove if such an algorithm exists or not for our construction. Therefore, we are unable to state with absolute certainty that our construction is second digest resistant; however, we believe it to be second digest resistant.

Increasing Security The motivation behind the property of increasing security is to ensure that the function's security will not become weaker as the value of the security parameter increases. While in practice this is very hard to prove, we are able to make some statement about our construction's security with respect to the security parameter assuming the underlying compression function acts as a random oracle. While creating such a compression function in practice is impossible, proving such a relation between the security of the construction and the security parameter in theory is helpful [1].

Theorem 3. *If the underlying compression function acts as a random oracle, then the overall construction will be increasingly secure as defined in Definition 9 as the security parameter increases.*

Before proving this theorem true, the following lemma is provided and proved to aid in proving the theorem. This lemma states that if the underlying compression function acts as a random oracle, then the dynamic hash function construction acts as random oracle.

Lemma 1. *If the underlying compression function used in the dynamic hash function construction acts as a random oracle, then the dynamic hash function construction acts as a random oracle.*

Proof. To show that the construction acts as a random oracle if the underlying compression function acts as a random oracle, we will borrow the proofs for the "Chop Solution" and the "NMAC Construction" presented in [5] from Sections 3.4 and 3.5 respectively. Let $c = |g(\cdot)|$ where g is the underlying compression function. There are two cases, when $d(s) \mid c$ and when $d(s) \nmid c$.

CASE 1: $d(s) \mid c$
When the size of the digest divides the size of the compression function's output, bits are not chopped from the result of the construction. In this case we apply the "NMAC Construction" proof presented in Section 3.5 of [5]. In our construction instead of applying a second random oracle function to the output, the output is generated by XORing two outputs of g. This follows the same intuition given in the paper. Applying an XOR to the final value prevents the "extension" attack described in [11]. Therefore, our construction is indifferentiable from a random oracle in this case if the underlying compression function acts as a random oracle.

CASE 2: $d(s) \nmid c$
When the size of the digest does not divide the size of the compression function's output, bits are chopped from the resulting output. In this case we apply the "Chop Solution" proof presented in Section 3.4 of [5]. While our construction is not exactly the same as the Merkle-Damgård construction, what prevents the distinguisher from differentiating between the two is not the underlying workings of the Merkle-Damgård construction, but the fact that bits are chopped from the output. Therefore, our construction is indifferentiable from a random oracle in this case if the underlying compression function acts as a random oracle.

Since in both cases our construction is indifferentiable from a random oracle if the underlying compression function is a random oracle, our construction is indifferentiable from a random oracle given the stated assumption. □

The proof for Theorem 3 follows directly from the application of the lemma and the birthday paradox.

Proof. By definition of the construction, as the security parameter increases the size of the digest increases as well. By application of Lemma 1 the construction can be shown to act as a random oracle under the stated assumption. Because of this the security of the construction will increase as the security parameter increases. The birthday attack defines precisely how much more secure it will be preimage resistance and collision resistance.

The proof that second digest resistance also increases as the security parameter increases, under the stated assumption, is essentially the same as preimage resistance. An attacker must compute a digest of size $d(s')$; however, as the length of the digest increases the probability of success decreases since all digests are random binary strings.

Therefore, as the security parameter of the construction is increased, the security of the construction also increases under the stated assumption. □

5 Conclusion

In this paper we extend the notion of a cryptographic hash function to a dynamic cryptographic hash function which takes two parameters, a message and a security parameter. The security parameter is used to specify the internal workings of the function and the size of the output. Definitions for the properties of preimage and collision resistance were provided. The additional properties of second digest resistance and increasing security were also defined to ensure a secure function with the addition of a security parameter. We also presented a new construction that allows a dynamic hash function to be created from an underlying compression function. Under certain assumptions this construction was proved to be preimage resistant, collision resistant, increasingly secure, and believed to be second digest resistant as defined for dynamic hash functions.

Acknowledgments

We would like to thank Moses Liskov and Bart Preneel for their insight and comments on earlier versions of this paper.

References

1. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *First ACM Conference on Computer and Communications Security*, November 1993.
2. Eli Biham and Rafi Chen. Near-collisions of SHA-0. In *Advances in Cryptology - CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*. Springer-Verlag, 2004.
3. Eli Biham and Orr Dunkelman. A framework for iterative hash functions – HAIFA. Technical report, National Institute for Standards and Technology, August 2006.
4. Florent Chabaud and Antoine Joux. Differential collisions in SHA-0. In *CRYPTO '98: Proceedings of the 18th Annual International Cryptology Conference on Advances in Cryptology*, Lecture Notes in Computer Science, pages 56–71, London, UK, 1998. Springer-Verlag.
5. Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. Merkle-Damgård revisited: How to construct a hash function. In Victor Shoup, editor, *CRYPTO 2005: 25th Annual International Cryptology Conference*, number 3621 in Lecture Notes in Computer Science, page 430. Springer-Verlag, 2005.

6. Ivan Damgård. A design principle for hash functions. In G. Brassard, editor, *CRYPTO '89: Proceedings*, volume 435 of *Lecture Notes In Computer Science*, page 416. Springer-Verlag, 1990.
7. Bert den Boer and Antoon Bosselaers. An attack on the last two rounds of MD4. In J. Feigenbaum, editor, *CRYPTO '91: Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 194 – 203. Springer-Verlag, 1991.
8. Bert den Boer and Antoon Bosselaers. Collisions for the compression function of MD5. In T. Helleseth, editor, *EUROCRYPT '93: Workshop on the Theory and Application of Cryptographic Techniques*, volume 765 of *Lecture Notes in Computer Science*, page 293. Springer-Verlag, 1993.
9. Hans Dobbertin. The first two rounds of MD4 are not one-way. In S. Vaudenay, editor, *Fast Software Encryption: 5th International Workshop*, volume 1372 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998. To appear.
10. Hans Dobertin. RIPEMD with two-round compress function is not collision-free. In *Journal of Cryptology*, volume 10, pages 51 – 68. Springer Verlag, 1997.
11. Stefan Lucks. Design principles for iterated hash functions. In *IACR preprint archive*, 2004.
12. Alfred Menezes, Paul Van Oorschot, and Scott Vanstone. *Handbook of applied cryptography*. The CRC Press series on discrete mathematics and its applications. CRC Press LLC, 2000 N.W. Corporate Blvd., Boca Raton, Florida 33431, 1997.
13. Ralph Merkle. A fast software one-way hash function. In *Journal of Cryptology*, pages 43 – 58, 1990.
14. NIST. *FIPS PUB 180-2: Secure Hash Standard*. National Institute of Standards and Technology, Gaithersburg, MD, USA, May 2002.
15. Bart Preneel. *Analysis and Design of Cryptographic Hash Functions*. Thesis (Ph.D.), Katholieke Universiteit Leuven, Leuven, Belgium, January 1993.
16. Bart Preneel. E-Mail correspondence with Bart Preneel, December 2005.
17. Phillip Rogaway and Thomas Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In Bimal Roy and Willi Meier, editors, *Fast Software Encryption: 11th International Workshop, FSE 2004*, volume 3017 of *Lecture Notes in Computer Science*, pages 371 – 388. Springer-Verlag, 2004.
18. Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full SHA-1. In Victor Shoup, editor, *CRYPTO 2005: 25th Annual International Cryptology Conference*, volume 3621 of *Lecture Notes in Computer Science*, page 17. Springer-Verlag, 2005.
19. Xiaoyun Wang and Hongbo Yu. How to break MD5 and other hash functions. In *EUROCRYPT 2005: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35. Springer-Verlag, 2005.