An abridged version of this paper appears in *Advances in Cryptology – Crypto 96 Proceedings*, Lecture Notes in Computer Science Vol. 1109, N. Koblitz ed., Springer-Verlag, 1996.

Keying Hash Functions for Message Authentication

Mihir Bellare[*] Ran Canetti[†] Hugo Krawczyk[‡]

June 1996

Abstract

The use of cryptographic hash functions like MD5 or SHA for message authentication has become a standard approach in many Internet applications and protocols. Though very easy to implement, these mechanisms are usually based on ad hoc techniques that lack a sound security analysis.

We present new constructions of message authentication schemes based on a cryptographic hash function. Our schemes, NMAC and HMAC, are proven to be secure as long as the underlying hash function has some reasonable cryptographic strengths. Moreover we show, in a quantitative way, that the schemes retain almost all the security of the underlying hash function. In addition our schemes are efficient and practical. Their performance is essentially that of the underlying hash function. Moreover they use the hash function (or its compression function) as a black box, so that widely available library code or hardware can be used to implement them in a simple way, and replaceability of the underlying hash function is easily supported.

[*]Department of Computer Science & Engineering, Mail Code 0114, University of California at San Diego, 9500 Gilman Drive, La Jolla, CA 92093. Email: `mihir@cs.ucsd.edu`. `http://www-cse.ucsd.edu/users/mihir`.

[†]IBM T.J. Watson Research Center, PO Box 704, Yorktown Heights, New York 10598. Email: `canetti@watson.ibm.com`. Work done while author was at MIT, supported by a post-doctoral grant from the Rothschild Foundation.

[‡]IBM T.J. Watson Research Center, PO Box 704, Yorktown Heights, New York 10598. `Email:hugo@watson.ibm.com`.

Contents

1 Introduction **3**
1.1 Authenticity and MACs 3
1.2 MACing with cryptographic hash functions 3
1.3 This work 3
1.4 A closer look 4
1.5 Related work 6

2 Basic Notions **6**
2.1 MACs and their security 6
2.2 Cryptographic Hash Functions 7

3 Keyed Hash Functions **9**

4 The Nested Construction NMAC **10**
4.1 The function NMAC 10
4.2 Security analysis 10

5 HMAC**: A fixed IV variant** **13**
5.1 The function HMAC 13
5.2 Security of HMAC 14
5.3 Implementation considerations for HMAC 14

6 Attacks and Comparison to other Proposals **15**

1 Introduction

1.1 Authenticity and MACs

Verifying the integrity and authenticity of information is a prime necessity in computer systems and networks. In particular, two parties communicating over an insecure channel require a method by which information sent by one party can be validated as authentic (or unmodified) by the other. Most commonly such a mechanism is based on a secret key shared between the parties and takes the form of a *Message Authentication Code* (MAC). (Other terms used include "Integrity Check Value" or "cryptographic checksum"). In this case, when party A transmits a message to party B, it appends to the message a value called the *authentication tag*, computed by the MAC algorithm as a function of the transmitted information and the shared secret key. At reception, B recomputes the authentication tag on the received message using the same mechanism (and key) and checks that the value he obtains equals the tag attached to the received message. Only if the values match is the information received considered as not altered on the way from A to B.[1] The goal is to prevent *forgery*, namely, the computation, by the adversary, of a message (not sent by the legitimate parties) and its corresponding valid authentication tag. A precise definition of MACs and their security is in Section 2.

1.2 MACing with cryptographic hash functions

MACs have most commonly been constructed out of block ciphers like DES. (The most popular in this genre is the CBC MAC, analyzed in [BKR, PV1].) More recently, however, there has been a surge of interest in the idea of constructing MACs from *cryptographic hash functions* like MD5 [Ri] and SHA-1 [SHA]. This is particularly visible in the Internet community, where the development of security protocols has led to the need for simple, efficient, and widely available MAC mechanisms.

It is easy to see why people want to MAC with cryptographic hash functions: the popular hash functions are faster than block ciphers in software implementation; these software implementations are readily and freely available; and the functions are not subject to the export restriction rules of the USA and other countries. The more difficult question is how best to do it. These hash functions were not originally designed to be used for message authentication. (One of many difficulties is that hash functions are not keyed primitives, ie. do not accommodate naturally the notion of secret key.) So special care must be taken in using them to this end. In particular, although many constructions have been proposed, they lack a sound and realistic security analysis. Thus there is a need for constructions which maintain the efficiency of the hash functions but are backed by a more rigorous analysis of their security. This is what we provide.

Section 1.5 describes some background and previous work on this subject. We now proceed to describe our work.

1.3 This work

In this paper we present two (related) new schemes, NMAC (the Nested construction) and HMAC (the Hash based mac). They can utilize any cryptographic hash function of the iterated type, and enjoy several attractive security, efficiency, and practicality features.

Security. Our constructions enjoy a formal security analysis that relates the security of the new functions to basic properties of the underlying hash schemes, like their resistance to collision finding.

[1] More generally, MAC schemes can involve the use of state information (e.g., a counter), use random nonces, or apply other mechanisms than just appending a tag. For concreteness we stick for now to simple MACs.

Our analysis considers any generic attack on MAC schemes (rather than showing security against a partial list of possible attacks) and shows that such an attack succeeds only if the underlying hash function is weak. Moreover, this relation between the assumed properties of the hash function and the security of the resultant MAC mechanism can be *tightly quantified*.

In summary, what this analysis says is that if significant weaknesses are ever found in the MAC schemes proposed here, then not only does the underlying hash function need to be dropped from these particular usages, but also it must be dropped from a wide range of other standard and popular usages to which these functions are now subject. Moreover, our constructions require from the hash function significantly weaker properties than standard collision-freeness. In particular, current successful methods for finding collisions in MD5 [Do1, Do2] seem inapplicable to breaking our schemes when the hash function in use is MD5 [Do3].

Efficiency. Our constructions use the cryptographic hash functions in a very simple way. In particular, the performance degradation relative to the underlying hash scheme is minimal. This is motivated by the use of these functions in basic applications like IP (Internet Protocol) security [At1, At2] where the performance cost of such a function influences the computational and network performance of many other applications.

Black box usage of hash functions. The constructions and analysis presented here are free from any dependency on the peculiarities of the underlying hash function. We only exploit the general structure of functions like MD5 and SHA-1, as being built on top of a basic *compression function* which works on fixed length messages, and is then iterated multiple times in order to process variable length inputs (see Section 2). Therefore, the underlying hash function (or the corresponding compression function) can be seen as a module that can be easily replaced in case serious weaknesses are found in the hash function, or when new (possibly, more secure or more efficient) hash functions are designed. This replaceability property is fundamental given the limited confidence earned so far by these functions.[2]

Besides the security advantage, there is a practical advantage to MAC schemes that use the underlying hash functions as a "black-box" (ie. by applying the hash function, or compression function, "as is", without any modifications). Namely such schemes permit the immediate use of existing and widely available library code that implements these functions. They also permit use of hardware-based implementations of the underlying hash scheme. Our NMAC construction uses the compression function as a black-box; our HMAC construction, even more conveniently, uses only calls to the iterated hash function itself.

1.4 A closer look

Before getting into the more technical aspects of the paper we further discuss our approach and results.

Keying hash functions. The first obstacle that one faces when coming to design a MAC scheme based on a cryptographic hash function (we limit ourselves, from now on, to "MD5-like" iterated hash functions, as described above), is that the latter usually do not use any cryptographic key. Rather, they are public functions that anyone can compute without the involvement of keys and secrets. This is in sharp contrast to a MAC function, which uses a secret key as an inherent part of its definition. Our approach to solve this problem is to key these hash functions through their *initial*

[2] It is worth observing that in the case of message authentication, as opposed to encryption, the breaking of a MAC does not compromise traffic authenticated in the past with the broken MAC. One can avoid the vulnerabilities created by new attacks, by replacing the underlying hash scheme as soon as this is broken.

variable (IV) (for details see Section 2). That is, the usually fixed IV defined by these functions is replaced by a random (and secret) string which becomes the key to the MAC.

Secure MACs from secure hash functions. A more fundamental problem is how to build the message authentication function in a way that the hardness of forging an authenticated message can be related to the cryptographic strength of the underlying hash function.

You can't make good wine from bad grapes: obviously, some strengths must be assumed of the hash function. On the other hand the assumptions should not be too strong, especially given that not enough confidence has been gathered in current candidates like MD5 and SHA-1. (In particular, it would certainly be possible to come up with "provably secure" MACs if one assumed the hash functions behaved like completely random functions, but this is less useful.) Our goal is to design MACs that involve the use of cryptographic hash functions in a simple way, yet their security can be argued on the basis of reasonable security assumptions on the underlying hash function. Moreover, we want this analysis to provide a *quantitative* relationship between the *assumed* strength of the hash function and the *proven* strength of the MAC.

We achieve the above goals in a strong sense. We are able to present a relatively simple analysis of our scheme which shows that an attacker that is able to forge our MAC function can, with the same effort (time and amount of collected information), break the underlying hash function in one of the following ways: (1) The attacker finds collisions in the hash function even when the IV is random and secret, and the hash value is not explicitly known; or, (2) The attacker is able to forge the secretly keyed compression function viewed as a MAC function applied to fixed length and partially unknown messages.

Consequently, existence of such attacks would contradict some of the basic assumptions about the cryptographic strength of these hash functions. Success in the first of the above attacks means success in finding collisions, the prevention of which is the main design goal of cryptographic hash functions. But in fact, even more is true: success in the first attack above is even *harder* than finding collisions in the hash function, because collisions when the IV is secret and the hash value is not explicitly known (as is the case here) is far more difficult than finding collisions in the plain (fixed IV) hash function. In particular, attacks when the IV is secret require interaction with the legitimate user of the function, and disallows the parallelism of traditional birthday attacks. Thus, even if the hash function is not collision-free in the traditional sense, our schemes could be secure. The success of the second attack above would imply that the randomness properties of the hash functions are very poor, and that all the bits of the hash output are simultaneously predictable (even with a secret IV and partially unknown input).

We stress that our analyses use *exact* analysis (no asymptotics involved), consider *generic* rather than specific attacks, and establish a *tight* relationship between the securities of the MAC and the underlying hash function.

Actual versus proven strengths. It is important to realize that our results are guided by the desire to have simple to state assumptions and a simple analysis. In reality, our constructions are even stronger than the analyses indicate, in the sense that even were the hash functions found not to meet the stated assumptions, our schemes might be secure. For example, even the weak collision resistance property as we state is an overkill, because in actuality, in our constructions, the attacker must find collisions in the keyed function without seeing any outputs of this function, which is significantly harder.

The later remark is relevant to the recently discovered collision attacks on MD5 [Do2, Do3]. While these attacks could be eventually adapted to attack the weak collision-resistance property of MD5, they do not seem to lead to a breaking of NMAC or HMAC even when used with MD5.

(See Section 6 for further discussion.) A more complex set of assumptions on the hash functions can be formulated to capture these extra strengths of our constructions but it is omitted here.

1.5 Related work

The exact security treatment of MACs began in [BKR] (where CBC-MAC is analyzed), and we use their definitions. Further block cipher based constructions were provided and analyzed in [BGR].

MAC constructions based on cryptographic hash functions have been in use for a few years (see Tsudik [Ts] for an early description of such constructions and Touch [To] for a list of Internet protocols that use this approach). Preneel and van Oorschot [PV1, PV2] survey existing constructions and point out to some of their properties and weaknesses; in particular, they present a detailed description of the effect of birthday attacks on iterated constructions. (These attacks remain the best possible ones on our schemes. But in practice they are infeasible. See Section 6 for more information.) They also present a heuristic construction, the MDx-MAC, based on these findings. Kaliski and Robshaw [KR] discuss and compare various constructions. Performance issues are discussed in [To, BGV].

In this work we have initiated the first rigorous treatment of the subject and, in particular, present the first constructions whose security can be formally analyzed, without resorting to unrealistic assumptions such as the "ideality" of the underlying hash functions.

In a companion work [BCK1] we consider how to design "pseudo-random functions" based on iterated compression functions. We show that if the compression function is pseudo-random then so is its iteration. The notion of a pseudo-random function is stronger than that of a MAC, and therefore that work can be viewed as making stronger assumptions than us (namely that the compression function is pseudo-random) in order to attain a stronger end (namely that the iterated construction too is pseudo-random).

Our HMAC construction was recently chosen as the mandatory to implement authentication transform for Internet security protocols and for this purpose is described in an Internet draft [KBC] and an upcoming RFC.

2 Basic Notions

2.1 MACs and their security

A MAC is a function which takes the secret key k (shared between the parties) and the message m to return a tag $\text{MAC}_k(m)$. The adversary sees a sequence $(m_1, a_1), (m_2, a_2), \cdots, (m_q, a_q)$ of pairs of messages and their corresponding tags (that is, $a_i = \text{MAC}_k(m_i)$) transmitted between the parties. The adversary breaks the MAC if she can find a message m, not included among $m_1, \cdots, m_q$, together with its corresponding valid authentication tag $a = \text{MAC}_k(m)$. The success probability of the adversary is the probability that she breaks the MAC. (Notice that an adversary who finds the key certainly breaks the scheme, but the scheme can also be broken by somehow combining a few messages and corresponding checksums into a new message and its valid checksum.)

The sequence $(m_1, a_1), (m_2, a_2) \cdots, (m_q, a_q)$ might have arisen in several ways. The simplest is a known message attack in which the legitimate parties choose the messages in a way un-influenced by the adversary, and the adversary, eavesdropping on the wire, picks them up. If the adversary can choose the sequence of messages $m_1, \cdots, m_q$, then it is called a *chosen message attack*. Notice that a MAC scheme secure against chosen messages is stronger than one which is secure only against "known messages". Here we will refer to chosen-message attacks, except if otherwise stated. We will consider the chosen messages as "queries" chosen by the adversary and answered (i.e., the corresponding checksums revealed) by the party (or parties) possessing the key to the MAC. Notice that a message m_i can be chosen by the adversary as a function of the previously seen messages and corresponding authentication tags.

Following [BKR], we quantify security in terms of the success probability achievable as a function of the number q of valid MAC examples seen by the adversary, and the available time t. (Note the success probability achievable for given t, q depends on the parameters of the MAC scheme, in particular its key length.) Then we say that MAC is a (ϵ, t, q, L)*-secure MAC* if any adversary that is not given the key k, is limited to spend total time (number of operations) t on the attack, and to request the value of the function MAC_k in up to q messages $m_1, m_2, \ldots, m_q$ of its choice, each of length at most L, cannot break the scheme except with probability better than ϵ.

As a convention we include in the time bound t the time it takes to compute the function MAC_k in each of the requested queries. This captures more realistically the fact that the time taken to compute the MACs which form the answers to these queries (especially when the number q of them is big) may constitute a significant portion of the time complexity of the attack. We also include in it the size of the code of the adversary's algorithm. (One can imagine an adversary who has pre-computed a lot of information and put it into its code).

Notice that the above definition is stated in terms of a generic attacker; we do not limit the attacker to any particular attacks or cryptanalytical techniques. Anything the adversary can do under the given resource bounds (time and queries) is captured by this definition. Also, notice the lack of asymptotics in this definition. Here t, q, and L can be replaced by actual numbers.

We observe that although we motivate and present our message authentication schemes in the context of message authentication for communications, the same techniques can be used to authenticate information that is stored in an insecure medium, and can be subject to malicious modification. Finally, we stress that while in this paper we concentrate solely on MAC constructions that use cryptographic hash function as their basis, there is an extensive literature covering MAC schemes based on different techniques.

2.2 Cryptographic Hash Functions

Basic properties. Cryptographic hash functions map strings of different lengths to short, fixed-size, outputs. These functions, e.g., MD5 or SHA-1, are primarily designed to be *collision resistant.* This means that if we represent such a hash function by F, then it should be infeasible for an adversary to find two strings x and x' such that $F(x) = F(x')$. Notice that this cryptographic notion does not involve any secret key. Indeed, the collision-resistance property is usually attached to key-less functions. The prime motivation for such functions is to be combined with digital signatures in a way that makes these signatures more efficient and yet unforgeable. For that application it is *required* that the function be publicly computable and, in particular, that it involve no secret key. (See [Ne] for background on collision-resistant hash functions.)

In addition to the basic collision-resistance property, cryptographic hash functions are usually designed to have some randomness-like properties, like good mixing properties, independence of input/output, unpredictability of the output when parts of the input are unknown, etc. Not only do these properties help in making it harder to find collisions, but also they help to randomize the input presented to the signature algorithm (e.g., RSA) as usually required for the security of these functions.

It is the combination of these properties attributed to cryptographic hash functions that make them so attractive for many uses beyond the original design as collision-resistant functions. These functions have been proposed as the basis for pseudorandom generation, block ciphers, random

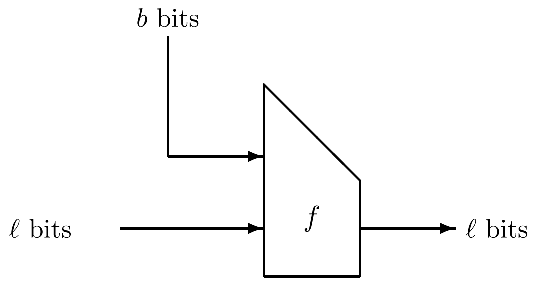

Figure 1: *A compression function. ($b = 512$, $\ell = 128$ in MD5.).*

transformation, and message authentication codes. We concentrate on a particular class of cryptographic hash functions, which we call *iterated constructions*.

ITERATED CONSTRUCTIONS. A particular methodology for constructing collision-resistant hash function has been proposed by Merkle [Me] (and later by Damgård [Da].) This methodology forms the basis for the design of the most common cryptographic hash functions like MD5 and SHA-1. It is based on a basic component called *compression function* which processes short fixed-length inputs, and is then iterated in a particular way in order to hash arbitrarily long inputs. Such a compression function, which we denote by f, accepts two inputs: a *chaining variable* of length ℓ and a block of data of length b. (For MD5 and SHA-1 we have $b = 512$, while for the first $\ell = 128$ and for the second $\ell = 160$. For the detailed description of the compression functions of these functions see [Ri, SHA].) See Figure 1.

The operation of the iterated hash function is as follows. First, an b-bit value IV is fixed. Next an input is hashed by iterating the compression function. That is, if $x = x_1, x_2, \ldots, x_n$ is the input, where the x_i's are blocks of length b each and n is an arbitrary number of blocks, then let $x_{n+1} = |x|$ be the message length. The value of the iterated function F on x is h_{n+1} where $h_0 = IV$ and $h_i = f(h_{i-1}, x_i)$ for $i = 1, 2, \ldots, n+1$. See Figure 2.

Notice that a way to pad messages to an exact multiple of b bits needs to be defined, in particular, MD5 and SHA pad inputs to always include an encoding of their length.

The motivation for this iterative structure arises from the observation (of Merkle [Me] and Damgård [Da]) that if the compression function is collision-resistant then so is the resultant iterated hash function. (The converse is not necessarily true). Thus, this structure provides a general design criterion for collision resistant hash functions since. Namely, it reduces the problem to the design of a collision resistant function on inputs of some fixed size.

In the rest of this paper we will concentrate on iterated hash functions, except if stated otherwise.

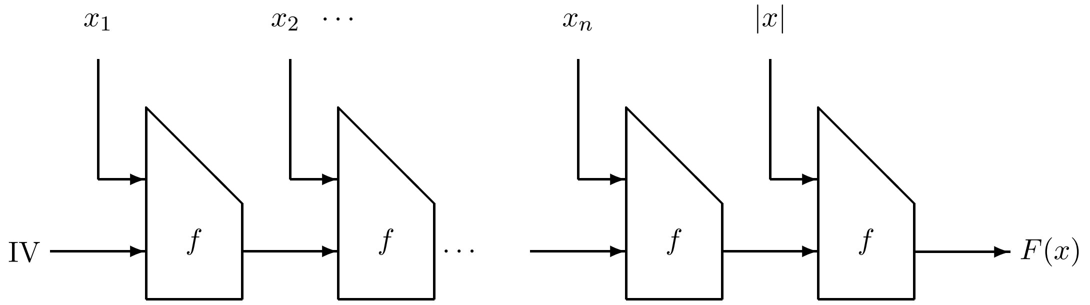

Figure 2: *The iterated construction of a hash function given a compression function f. The input $|x|$ to the last iteration illustrates the appending of the message length as in MD5 and SHA-1.*

We will use the symbol f to denote the compression function, and F to denote the associated iterated hash which we assume to include a standard way to pad inputs to to an exact multiple of b bits.

3 Keyed Hash Functions

Recall our goal is to build secure message authentication functions from cryptographic hash functions (in particular, from iterated hash functions). A first clear obstacle is that while secret keys are an essential ingredient in a message authentication function, most cryptographic hash functions, and specifically functions like MD5 or SHA, do not use keys at all. Therefore, we first need to define a way to use cryptographic hash functions in conjunction with a key.

The most common approach to key a hash function (see [Ts]) is to input the key as part of the data hashed by the function, e.g., hashing data x using key k is performed by applying the hash function F to the concatenation of k and x. Our approach is to key the function's IV. Namely, instead of using a fixed and known IV as defined by the original function, we replace it by a random and secret value known only to the parties. It turns out that the latter approach has some significant analytical advantages. It allows for a better modeling of keyed hash functions as needed for the security analysis of these functions. On the other hand, we will also see that this approach can be "simulated" through the use of keys padded and prepended to data (see Section 5).

Using the "keyed IV" approach we can define keyed hash functions as a *family of functions*. Let f_k defined by $f_k(x) = f(k, x)$ be the keyed compression function, where $|k| = \ell$ and $|x| = b$. Now we associate to any iterated hash construction (e.g., MD5, SHA-1) a family of (keyed) functions $\{F_k\}_k$. Namely for $x = x_1 \dots x_n$ define $F_k(x)$ to be k_{n+1} where $k_i = f_{k_{i-1}}(x_i)$ for $i = 1, \dots, n+1$, $k_0 = k$, and $x_{n+1} = |x|$. Notice that the space of keys is the same for the keyed compression functions and for the keyed iterated hash functions: it is the set of all the strings of length ℓ. The original iterated hash function is obtained as a particular member of the keyed family, namely, F_{IV}.

As noted before the notion of collision-resistance has been traditionally attached to public (keyless) functions. In this work we extend this notion to (secretly) keyed hash functions.

Definition 3.1 We say that a family of keyed hash functions $\{F_k\}$ is (ϵ, t, q, L)-*weakly collision-resistant* if any adversary that is not given the key k, is limited to spend total time t, and sees the values of the function F_k computed on q messages $m_1, m_2, \dots, m_q$ of its choice, each of length at most L, cannot find messages m and m' for which $F_k(m) = F_k(m')$ with probability better than ϵ.

Notice that the above requirement is weaker than the traditional requirement of collision-resistance from (key-less) hash functions. In the latter case, it is enough to find collisions for a known and fixed IV. Also in the case of secretly keyed hash functions the adversary needs to get its "examples" (the messages $m_1, m_2, \dots, m_q$ in the above definition) from the legal user itself who knows the key k. In the key-less case, the attacker can work in finding collisions independently of any user or key. Moreover, even brute force collision search attacks can become feasible for functions like MD5 due to the easiness of parallelization of these attacks (see [VW]). In contrast, attacks on secretly keyed hash functions cannot be parallelized as they require interaction with the legal user.

The best known collision attacks (see Section 6) on our schemes would require the legal user to hash about 2^{64} messages (of, say, one block each) known to the adversary using the same key. If the hashing can be done at a speed of 1Gbit/second this would require continuous hashing (using the same key) during 250,000 years.

We note that attacks that find collisions on the hash function with random and known IVs (such attacks are known for MD4 [Do1]) can be adapted (via extension attacks) to find collisions even if the IV is secret. Such an attack would compromise the weak collision resistance property of the hash function. We will see that in our constructions, extension attacks are prevented by a double application of the hash function. (This issue is further discussed in section 6.)

4 The Nested Construction NMAC

We present our basic construction NMAC (for "Nested MAC") and its analysis. In the next section we describe a variant, HMAC, that is further geared towards practical applications. Denote by f_k and F_k the keyed versions of a given compression function and its iterated function, as described in Section 2.

4.1 The function NMAC

Let $k = (k_1, k_2)$ where k_1 and k_2 are keys to the function F (i.e., random strings of length ℓ each). We define a MAC function NMAC(x) which works on inputs x of arbitrary length as

$$\text{NMAC}_k(x) = F_{k_1}(F_{k_2}(x)).$$

Notice that the outer function acts on the output of the iterated function and then it involves only one iteration of the compression function. That is, this outer function is basically the compression function f_{k_1} acting on $F_{k_2}(x)$ padded to a full block size (in some standard way as defined by the underlying hash scheme F).

Notice the simplicity and efficiency of the construction. The cost of the internal function is exactly the same as hashing the data with the basic (keyless hash function). The only additional cost is the outer application which, as said, involves only one iteration of the compression function. Most importantly, in the next section we provide with a rigorous analysis of the strength of this construction relative to the cryptographic strength of the underlying hash function.

4.2 Security analysis

Following we state the main analytical result regarding the proposed function NMAC. For the definitions of a secure MAC and weakly collision-resistant functions see Sections 2.1 and 3, respectively.

Theorem 4.1 If the keyed compression function f is an (ϵ_f, q, t, b)-secure MAC on messages of length b bits, and the keyed iterated hash F is (ϵ_F, q, t, L)-weakly collision-resistant then the NMAC function is an $(\epsilon_f + \epsilon_F, q, t, L)$-secure MAC.

The theorem thus states that *any* adversary that mounts an attack against the function NMAC$_k$ as a MAC by querying the function in q inputs of length at most L (i.e., the adversary, that does not know k, gets the value of the function NMAC$_k$ on q messages of its choice) and spending a total processing time of t in the attack, has a probability of success which is no more than twice the probability of success of an adversary that spends the same resources in finding collisions for the *secretly keyed* iterated function, and no more than twice the probability of success of an adversary that under the same resources tries to break the keyed compression function as a MAC.

Proof of Theorem 4.1: Let's fix parameters q,t, and L, for the number of queries, processing time, and length of messages, respectively, available to an attacker that tries to break the function

```
Choose random $k_2$
For $i = 1, \ldots, q$ do
    $A_N \to x_i$
    $A_N \leftarrow f_{k_1}(\overline{F_{k_2}(x_i)})$
$A_N \to (x, y)$
output $(\overline{F_{k_2}(x)}, y)$
```

Figure 3: The algorithm A_f

NMAC. Let's denote this attacker by A_N and its probability of success by ϵ_N. Let ϵ_F be the best probability of success of any adversary that under the above same resources tries to find collisions for the function F_{k_2} without knowing k_2. Using A_N we will build an adversary A_f that forges the MAC function f_{k_1} on inputs of length b by spending q queries and time t with a success probability of $\epsilon_f \geq \epsilon_N - \epsilon_F$. Thus, we show that any adversary that tries to break the function NMAC using the above resources has a probability of success ϵ_N of at most $\epsilon_f + \epsilon_F$. The theorem then follows.

We specify A_f using A_N as a subroutine. Recall that A_N works as follows. It queries the function NMAC_k (for which it does not know k) on a message x_1 and gets the response $\text{NMAC}_k(x_1)$, it then produces a second query x_2 for which it gets $\text{NMAC}_k(x_2)$, and so on for a total of q queries. It finally outputs a pair (x, y). If $x \neq x_i$, for $i = 1, 2, \ldots, q$, and $y = \text{NMAC}_k(x)$ then the attack succeeds, otherwise it fails.

Remember that the goal of A_f is to forge f_{k_1}, by querying the latter function on messages that A_f itself chooses. In order to describe A_f we need to specify how it chooses the messages to query, and how it finally outputs a message x and an authentication tag. We use the following notation: for a string s of length ℓ, we denote by $\overline{s}$ the result of s padded to a full block of length b as specified by the underlying hashing scheme.

A_f starts by first choosing a random key k_2 for the keyed iterated hash function F_{k_2}. It then activates A_N which produces queries to the function NMAC that are answered by A_f in the following way: for each query x_i presented by A_N, the algorithm A_f computes $z_i = F_{k_2}(x_i)$, and queries f_{k_1} on $\overline{z_i}$; the answer $f_{k_1}(\overline{z_i})$ returned by f_{k_1} is then fed to A_N as the response to its query x_i. (Notice that the answers to A_N correspond to the case that A_N is querying the function NMAC_k for $k = (k_1, k_2)$.) Finally, after such q queries are answered, A_N outputs its forgery (x, y). A_f uses this pair (x, y) to compute the pair $(\overline{F_{k_2}(x)}, y)$, which it then outputs as its forgery. See Figure 3.

Next we analyze the success probability ϵ_f of the above algorithm A_f. Notice that A_f fails in two cases: whenever A_N fails (i.e., A_N fails to output a successful forgery for a new message), or when A_N outputs a correct forgery for a new message x (different from all queried x_i's) but for which $\overline{F_{k_2}(x)} = \overline{F_{k_2}(x_i)}$, for some $i = 1, \ldots, q$. Notice that in the later case,

$$F_{k_2}(x) = F_{k_2}(x_i). \tag{1}$$

We then bound the failure probability of A_f by the sum of the above two failure events, namely, that A_N fails or that equality (1) holds. Notice, first, that the behavior of A_N as used in the above algorithm A_f, is equivalent to the behavior of A_N when trying to break the function NMAC_k, where $k = (k_1, k_2)$ and each of k_1 and k_2 were chosen at random and not given to A_N. This is so since each of A_N's queries x_i is answered by A_f with the value $f_{k_1}(\overline{F_{k_2}(x_i)})$ which is exactly $\text{NMAC}_k(x_i)$. Therefore, the probability that A_N fails during the above algorithm is as its probability to fail to forge NMAC_k, which by definition is at most $1 - \epsilon_N$.

In order to bound the probability that equality (1) holds, one needs to note that the algorithm described above for A_f can be used as an algorithm for finding collisions for the function F_{k_2} with secret k_2, by choosing a random key k_1 for the compression function f_{k_1}, and quering the function F_{k_2} on the messages x_i produced by A_N. In this case equality (1) implies the success of the collision-finding algorithm. By the definition of ϵ_F, no collision search algorithm against F_{k_2} which is limited to the above resources (number of queries and time) can find collisions with probability better than ϵ_F, therefore we get that the probability that (1) holds is at most ϵ_F.

Summarizing, the probability $1 - \epsilon_f$ that the above algorithm A_f fails to forge f_{k_1} is at most $(1 - \epsilon_N) + \epsilon_F$. That is, $\epsilon_f \geq \epsilon_N - \epsilon_F$. Therefore, $\epsilon_N \leq \epsilon_f + \epsilon_F$ and the theorem follows. ▌

Next we present a few remarks on the above analysis.

Remark 4.2 An important aspect of the analysis is that the proof of the theorem is *constructive*, namely, given an adversary (or algorithm) that breaks the new function NMAC with some significant probability, one can *explicitly* show an algorithm that using the same resources breaks the underlying hash function with at least half of that probability. This shows the practicality of the analysis and that the degradation of security when going from the underlying hash function to the NMAC construction is minimal. This is even stronger considering the *generic* nature of our "attacker". It represents all possible attackers, including all possible future developments in cryptanalysis.

Remark 4.3 Our definitions and analysis are stated in terms of chosen (and adaptive) message attacks since these are the strongest attacks that one wants to protect against. One can have more refined definitions which quantify separately the number of chosen messages and number of known messages required by an attack. Although this distinction is significant in practice, we omit it here for simplicity. However, it is important to realize that our analysis deals very satisfactorily with this practical aspect as it preserves the number of known/chosen messages when translating an attack on the NMAC function to an attack on the underlying hash (e.g., if there is a known-only message attack on NMAC, then such an explicit attack exist also against the iterated function F, or the compression function f).

Remark 4.4 The actual assumptions required by our analysis are even weaker than stated above since an attacker trying to break NMAC by attacking the compression function as a MAC cannot choose or control (and even know) the exact value on which this function is applied since the value of $F_{k_2}(x)$ is not exposed to the attacker. Similarly, when trying to break the function via collisions of the internal computation $F_{k_2}(x)$ the adversary does not obtain the actual result of this function but only its value after applying F_{k_1} (however, notice, that the application of the outer function does not hide the fact that collisions occurred in the internal computation).

Remark 4.5 The weak-collision-freeness assumption made in the theorem can be replaced by the significantly weaker assumption that the inner hash function is collision resistant to adversaries that see the hash value only after it was hashed again with a different secret key.

Remark 4.6 Another aspect by which the result is more general than stated in the theorem is as follows. Notice that although we use the same function for the inner and outer application in NMAC one could use totally different functions for these two applications. In that case the above theorem would still hold given the MAC assumption on the outer function, and the collision-resistance property of the internal function. This can lead to hybrid approaches like using SHA-1 (keyed through its IV) for the internal iterated function, and DES-MAC-CBC for the external, etc.

Remark 4.7 In spite of the use of two different keys k_1 and k_2, the security of the function is given by each individual key (of length ℓ) and not by the combination (2ℓ) of their lengths. For more details on this aspect see the divide and conquer attack in Section 6.

Remark 4.8 Preneel and van Oorschot [PV1] recommend outputting only half of the bits of the hash output as the authentication tag. This recommendation is motivated by the fact that forgery attacks are anyway doable with complexity $2^{\ell/2}$ through birthday attacks (see Section 6), and that by outputting only half of the bits these attacks require more chosen messages. Carrying this recommendation to our constructions would require the assumption that the compression function with truncated output (e.g., only the $\ell/2$ most significant bits of output) constitutes a secure MAC. Whether this is acceptable or not depends on the assumed properties of the particular compression function in use. (The tradeoff here is that by outputting less bits the attacker has less bits to predict in a MAC forgery but, on the other hand, the attacker also learns less about the output of the compression function from seeing the authentication tags computed by the legitimate parties.) Applications for which a saving in the MAC length is important can adopt our constructions with a reduced output of (at least) $\ell/2$ bits.

Remark 4.9 One can ask how much our assumptions on the underlying hash function can be further weakened and still have a simple construction of a secure MAC. Although we cannot answer this question in a formal way (e.g., secure MAC functions can be built from the weaker assumption that the compression function is a one-way function, but the known constructions to achieve that are totally impractical), we can point out to two facts. First, by just assuming that the compression function is a MAC one cannot guarantee that the iterated function is a MAC. That is clearly shown by the extension attacks discussed in Section 6. In particular, this shows that one cannot just omit the outer application of F_{k_1} in NMAC and still get a secure MAC. As for basing the construction in collision-resistance only, we stress that this property also is insufficient to make the function a secure MAC. Indeed, one can construct examples of strong collision-resistant functions that are easily forgeable as MAC. Moreover, one can show this to hold for specific proposals of MAC functions based on hash schemes.

5 HMAC: A fixed IV variant

Due to the wide availability of free library code for existing hash functions (especially MD5), it is a practical advantage to build MAC mechanisms that use these functions as a black-box, so that the MAC can be implemented by simply calling the existing function. The NMAC construction presented in Section 4 requires direct access to code for the compression function (rather than for the overall hash function), in order to key the IV. Such a change is trivial for functions with well-structured code like MD5 (see [Ri]). However, in some cases one would still like to avoid even those minimal changes, and use the code (or hardware implementation) as is. Here we present an adaptation of NMAC that achieves this goal. As an additional advantage, this construction involves a single ℓ-bit long key k as opposed to two different keys as in NMAC. This has some advantages at the level of key management. With an additional assumption on the underlying compression function one can show the applicability of the NMAC analysis to HMAC.

5.1 The function HMAC

Let F be the (iterated and key-less) hash function initialized with its usual fixed IV. The function HMAC works on inputs x of arbitrary length and uses a single random string k of length ℓ as its key:

$$\mathrm{HMAC}_k(x) = F(\bar{k} \oplus \mathsf{opad}, F(\bar{k} \oplus \mathsf{ipad}, x))$$

where $\bar{k}$ is the completion by adding 0's of k to a full b-bit block-size of the iterated hash function, opad and ipad are two fixed b-bits constants (the "i" and "o" are mnemonics for inner and outer), $\oplus$ is the bitwise Exclusive Or operator, and the commas represent concatenation of the information. opad is formed by repeating the byte x'36' as many times as needed to get a b-bit block, and ipad is defined similarly using the byte x'5c'. (For example, in the case of MD5 and SHA-1 these bytes are repeated 64 times).

5.2 Security of HMAC

The security of HMAC is based on the security of NMAC. The main observation for relating these two functions and their security is that by defining $k_1 = f(\bar{k} \oplus \mathsf{opad})$ and $k_2 = f(\bar{k} \oplus \mathsf{ipad})$, we get that $\mathrm{HMAC}_k(x) = \mathrm{NMAC}_{(k_1,k_2)}(x)$. In other words, the above transformation on the key makes HMAC a particular case of NMAC, where the keys k_1 and k_2 are "pseudorandomly" derived from k using the compression function f. Since the analysis of NMAC assumes that k_1 and k_2 are random and independently chosen keys, then in order to apply this analysis to HMAC one needs to assume that k_1 and k_2 derived using f cannot be distinguished by the attacker from truly random keys. This represents an additional assumption on the quality of the function f (keyed through the *input* k) as a pseudorandom function. We require a relatively weak form of pseudorandomness since the adversary trying to learn about possible dependencies of k_1 and k_2 does not get to see directly the output of the pseudorandom function on *any* input. To sum things up, attacks that work on HMAC and not on NMAC are possible, in principle. However, such an attack would reveal major weaknesses of the pseudorandom properties of the underlying hash function.

It is important to note that in practice most keys are chosen pseudorandomly rather than as truly random strings; in particular, it is plausible that even if one uses NMAC, implementations will choose to derive k_1 and k_2 using a pseudorandom generator. In the case of HMAC such a pseudorandom generator is "built-in" through the definition of the function using the function f and the above defined pads. This use for pseudorandom generation of functions like MD5 or SHA-1 is very common in practical implementations (in fact, the designers of SHA-1 recommended using this function for pseudorandom derivation of various quantities in the DSS standard [DSS]).

The above particular values of opad and ipad were chosen to have a very simple representation (to simplify the function's specification and minimize the potential of implementation errors), and to provide a high Hamming distance between the pads. The latter is intended to exploit the mixing properties attributed to the compression function underlying the hash schemes in use. These properties are important in order to provide computational independence between the two derived keys.

Finally, we note that the use of a single ℓ-bit long key as opposed to two (independent) keys does not represent a weakening of the function relative to exhaustive search of the key, since even when chosen independently the keys k_1 and k_2 can be individually searched through a divide and conquer attack as described in Section 6.

5.3 Implementation considerations for HMAC

Here we point out to some implementation issues. Notice that HMAC results in a slower function than NMAC since the former requires two extra computations of the compression function (on the blocks $(\bar{k} \oplus \mathsf{opad})$ and $(\bar{k} \oplus \mathsf{ipad})$). This can have a negligible effect when authenticating long streams of data but may be significant for short data. Fortunately, an implementation can avoid this extra computation by "caching" the values of k_1 and k_2 (as defined in section 5). That is, these values are computed only once when the key k is generated or shared the first time, and then stored as the actual keys to the function NMAC. To use these separate keys the implementation needs to be able to initialize the IV's of the hash function to these values before processing the data. (As said before this is usually very easy to do.) In this way, HMAC serves those implementations that require the use of the iterated hash function with no modification (i.e., with the fixed IV), and at the same time it does not penalize implementations that can key the function through the IV.

Notice that one can define the function HMAC to support variable length keys. However, less than ℓ bits for the key is not recommended since that would weaken the strength of the keyed IV (i.e, k_1 and k_2). On the other hand, longer than ℓ-bit keys will not provide, in general, with added strength since the derived k_1 and k_2 are anyway of length ℓ (still, having a longer key k may help, depending on the properties of the compression function f and the randomness of the key k, to have a stronger pseudorandom effect on the generation of k_1 and k_2).

Finally, we stress that as in any cryptographic implementation, a secure key management is essential for the security of functions like the ones proposed here. In particular, a periodic refreshment of keys is advisable. Even if under currently known attacks (see Section 6) one could use the same key for extremely long periods of time without cryptanalytic compromise, implementation should limit the time and amount of information processed with the same key.

6 Attacks and Comparison to other Proposals

In Section 4 we have established a tight and general relationship between the security of the function NMAC and the underlying hash function, in a way not known to hold for any other similar construction. We are thus assured that if a "good" cryptographic hash function is used then all attacks against our schemes will fail to be practical. Nonetheless it is instructive to actually see what known attacks achieve, and cross-check that indeed they don't work. We also compare our construction to other proposals.

Birthday attacks. As shown in [PV1] and our companion work [BCK1], birthday attacks, that are the basis to finding collisions in cryptographic hash functions, can be applied to attack also keyed MAC schemes based on iterated functions (including also CBC-MAC, and other schemes). These attacks apply to our new constructions as well. In particular, they constitute the best known forgery attacks against both the NMAC and HMAC constructions. Consideration of these attacks is important since they strongly improve on naive exhaustive search attacks. However, their practical relevance against these functions is negligible given the typical hash lengths like 128 or 160, since these attacks require knowledge of the MAC value (for a given key) on about $2^{\ell/2}$ messages (where ℓ is the length of the hash output). For values of $\ell \geq 128$ the attack becomes totally infeasible.[3] In contrast to the birthday attack on key-less hash functions, the new attacks require interaction with the key owner to produce the MAC values on a huge number of messages, and then allow for no parallelization. For example, when using MD5 such an attack would require the authentication of 2^{64} blocks (or 2^{73} bits) of data using the same key. On a 1 Gbit/sec communication link, one would

[3]We illustrate the birthday attack against NMAC (it similarly applies against HMAC). Other variations are possible. The attacker collects the value of NMAC_k on a number of equal-length messages until it finds two different messages m_1 and m_2 for which $\text{NMAC}_k(m_1) = \text{NMAC}_k(m_2)$. Subsequently, it requests the value of NMAC_k on a message $m_1' = (m_1, B)$, for some block B (i.e., m_1' equals m_1 concatenated with the block B; the issue of length padding is omitted here as it can be easily handled). The attacker then outputs a forgery for $m_2' = (m_2, B)$ using the authentication tag $\text{NMAC}_k(m_1')$. If the collision $\text{NMAC}_k(m_1) = \text{NMAC}_k(m_2)$ was due to a collision in the internal function $F_{k_2}(m_1) = F_{k_2}(m_2)$ then the forgery is successful. Only after collecting about $2^{\ell/2}$ messages the probability of such an attack to succeed is significant.

need 250,000 years to process all the data required by such an attack. This is in sharp contrast to birthday attacks on key-less hash functions which allow for far more efficient and close-to-realistic attacks [VW].

Notice that these attacks produce forgery of the MAC function but not key recovery. In [PV2], however, it is shown that in some versions of the envelope method (the case where the same key is used to prepend and append and no block alignment of the appended key is performed), the birthday attacks can be further enhanced to provide full key recovery in time much shorter than required by full exhaustive search. Since these attacks require at least the complexity mentioned above for forgery based on birthday attacks, they cannot be considered as practical ones. Yet, it is interesting to note that they do not apply to either of our constructions, since here the alignment issue exploited by these attacks is no applicable.

The forms of birthday attacks that apply to our constructions can become feasible only if very significant weaknesses in the collision probability of the underlying hash function are discovered. However, in such a case the basic use of such a function as collision-resistant (as originally intended) would be strongly compromised, and the function should be dropped for cryptographic use. Finally, we mention that these birthday attacks (at least in their straightforward form) can be avoided by randomizing the MAC construction in a per-message basis. We refer to [BCK1] for further details.

Collision attacks on the key-less hash function. Consider the "append-only" construction: $\mathrm{MAC}_k(x) = F(x,k)$. Assume that two strings x and x' are known for which $F(x) = F(x')$ (this collision corresponds to the key-less hash function). Then, regardless of the key k in use, one knows that $\mathrm{MAC}_k(x) = \mathrm{MAC}_k(x')$ (actually, that is true for extensions of x and x' as well). Finding a collision pair x, x' for the function F is far easier than attacking NMAC through collisions in F_k, where k is unknown. As showed in the above discussion on birthday attacks, while the latter is totally infeasible even for hash lengths of $\ell = 128$, finding collisions to the plain hash function through birthday attacks approaches feasibility (see [VW]). The reason is that such a collision attack on the plain hash function can be performed off-line and independently of any secret key (and thus requires no interaction with the legitimate owner of k), and it is strongly parallelizable. None of these advantages for the attacker exist when attacking NMAC. In addition, as the recent experience teaches us, it is much easier to find collisions via analytical methods (e.g., [Do1, Do2, Do3]) against the key-less function than breaking our schemes. We finally note that a variant of NMAC where the outer function is keyed but not the internal (i.e., $F_k(F(x))$) is susceptible to the same attack through plain collisions as the append-only construction, and is significantly weaker than NMAC.

The extension attack. Consider the "prepend-only" construction: $\mathrm{MAC}_k(x) = F(k,x)$ (i.e., the key k is prepended to the data x and the hash function – with the fixed IV – computed on the concatenated information). Because of the iterative structure of F it is easy to see that if one knows the value of $\mathrm{MAC}_k(x)$ where x contains an integral number of blocks, then one can compute the value of MAC_k on any extension y of x (i.e., any string y that contains x as a prefix) by just using the result of $\mathrm{MAC}_k(x)$ as an intermediate value of the chaining variable in the computation of $\mathrm{MAC}_k(y)$. This attacks needs no knowledge of or direct attack on the key k. In NMAC this attack is prevented through the outer application of F_{k_1}, which avoids the exposure of the result of the iterated function F_{k_2}.

As noted in section 3 extension attacks can allow the transformation of attacks against a hash function that uses a random but known IV into an attack against secret IVs. Such attacks are known against MD4 [Do1] and are plaussible to exist against MD5 [Do2, Do3]. However, these attacks are inapplicable against our MAC constructions where, as said above, the outer application of the hash function prevents the extension attacks.

Divide and conquer attacks. Consider the method, known as the "envelope" method, that combines the above prepend and append constructions, namely, $\text{MAC}_{k_1,k_2}(x) = F(k_1, x, k_2)$. Preneel and van Oorschot [PV1] observe that in an attack directed to recover the whole key one does not need to work exponential time on the added length of keys k_1 and k_2 but one can recover both keys in a total time which is exponential on the length of one key. This is done by first finding collisions in the MAC function, and then searching exhaustively for a key (k_1) that produces these collisions. Once we have the right k_1 it is straightforward to find k_2 by exhaustion. Although this attack is impractical, it serves to illustrate the basic fact that the strength of the function comes from its individual keys and not from their combined length.

A similar attack holds against NMAC. This is in no contradiction with the analysis of Section 4 that shows the security of NMAC based on the strength of the individual underlying functions, i.e., the keyed compression function as a MAC and the keyed iterated function as weakly collision-resistant. The divide and conquer attack shows that one cannot replace in Theorem 4.1 the expression $\epsilon_f + \epsilon_F$ by the much stronger $\epsilon_f \cdot \epsilon_F$. It also serves to show that the use of a single ℓ-bit long key in HMAC does not weaken the function against exhaustive search.

Comparison with the construction of [PV1]. In [PV1] a construction is proposed which is also a variant of the envelope method. It uses a keyed IV and an appended key, but in addition it uses a third key that is applied to influence the internal rounds of the compression function in use. (All these keys are derived from a single underlying key.) This is a heuristic measure intended to counter possible weaknesses of the compression function in use, and no formal analysis of the construction is provided. We note that this construction is more "intrusive" in the sense that it requires some more changes to the existing hash functions, and it impacts performance in a moderate but noticeable way.

Comparison with rfc1828. The MAC scheme described in RFC1828 [MS] has been proposed as a standard mechanism for message authentication in the context of IP (Internet Protocol) security. This function, which uses MD5 as the underlying hash function, is based on the envelope method, but pads the prepended key to a full block boundary. In addition, it uses the *same* key for prepending and appending. The best analysis known for this type of functions is given in [BCK1] which show that when using *different and independent* keys (for prepend and append) the security of the function can be based on the pseudorandom properties of the underlying compression function. The NMAC function presented in this paper enjoys a superior security analysis in two important aspects: it requires weaker assumptions on the underlying hash function (since the pseudorandomness of the compression function implies our assumptions), and the security of the underlying hash function is preserved in a significantly stronger way (as quantified in Theorem 4.1) than in the analysis in [BCK1]. Another important difference is that the HMAC variant deals better with the use of a single key than does the construction of RFC1828; in the latter, the use of the same key for prepend and append makes the analysis in [BCK1] less applicable and, in particular, makes the scheme susceptible to the above mentioned key recovery attack of [PV2].

HMAC has now replaced the RFC1828 construction as the mandatory to implement authentication transform for Internet security protocols [KBC].

Acknowledgments

We thank Burt Kaliski, Bart Preneel, Matt Robshaw, Adi Shamir and Paul van Oorschot for helpful discussions on the subject of this paper. In particular we thank Adi Shamir for suggesting the use of XOR pads instead of concatenated pads in our original design of HMAC.

References

[At1] R. Atkinson, "Security Architecture for the Internet Protocol", IETF Network Working Group, RFC 1825, August 1995.

[At2] R. Atkinson, "IP Authentication Header", IETF Network Working Group, RFC 1826, August 1995.

[BCK1] M. Bellare, R. Canetti and H. Krawczyk, "Pseudorandom functions revisted: the cascade construction and its concrete security," Proceedings *of the* 37th *Symposium on Foundations of Computer Science*, IEEE, 1996.

[BGR] M. Bellare, R. Guérin and P. Rogaway, "XOR MACs: New methods for message authentication using finite pseudorandom functions," *Advances in Cryptology – Crypto* 95 *Proceedings*, Lecture Notes in Computer Science Vol. 963, D. Coppersmith ed., Springer-Verlag, 1995.

[BKR] M. Bellare, J. Kilian and P. Rogaway. "The security of cipher block chaining." *Advances in Cryptology – Crypto* 94 *Proceedings*, Lecture Notes in Computer Science Vol. 839, Y. Desmedt ed., Springer-Verlag, 1994.

[BGV] A. Bosselaers, R. Govaerts, J. Vandewalle, "Fast hashing on the Pentium," *Advances in Cryptology – Crypto* 96 *Proceedings*, Lecture Notes in Computer Science Vol. 1109, N. Koblitz ed., Springer-Verlag, 1996.

[Da] I. Damgård, "A design principle for hash functions," *Advances in Cryptology – Crypto* 89 *Proceedings*, Lecture Notes in Computer Science Vol. 435, G. Brassard ed., Springer-Verlag, 1989.

[Do1] H. Dobbertin, "Cryptanalysis of MD4," *Fast Software Encryption Workshop*, Lecture Notes in Computer Sciences, vol. 1039, Springer Verlag, 1996, pp. 53-69.

[Do2] H. Dobbertin, "MD5 is not collision-free," Manuscript, 1996.

[Do3] H. Dobbertin, " The Status of MD5 After a Recent Attack", RSA Labs' CryptoBytes, Vol. 2 No. 2, Summer 1996. `http://www.rsa.com/rsalabs/pubs/cryptobytes.html`

[DSS] National Institute for Standards and Technology, "Digital Signature Standard (DSS)", Federal Register, Vol. 56, No. 169, August, 1991

[GGM] O. Goldreich, S. Goldwasser and S. Micali, "How to construct random functions," *Journal of the ACM,* Vol. 33, No. 4, 210–217, (1986).

[KR] B. Kaliski and M. Robshaw, "Message Authentication with MD5", *RSA Labs' CryptoBytes,* Vol. 1 No. 1, Spring 1995.

[KBC] H. Krawczyk, M. Bellare and R. Canetti, Internet draft draft-ietf-ipsec-hmac-md5-txt.00, March 1996.

[MS] P. Metzger and W. Simpson, "IP Authentication using Keyed MD5", IETF Network Working Group, RFC 1828, August 1995.

[Me] R. Merkle, "One way hash functions and DES," *Advances in Cryptology – Crypto* 89 *Proceedings*, Lecture Notes in Computer Science Vol. 435, G. Brassard ed., Springer-Verlag, 1989. (Based on unpublished paper from 1979 and his Ph. D thesis, Stanford, 1979).

[Ne] J. Nechvatal, "Public Key Cryptography," in Contemporary Cryptography, The Science of Information Integrity, G. Simmons ed., IEEE Press, 1992.

[PV1] B. Preneel and P. van Oorschot, "MD-x MAC and building fast MACs from hash functions," *Advances in Cryptology – Crypto* 95 *Proceedings*, Lecture Notes in Computer Science Vol. 963, D. Coppersmith ed., Springer-Verlag, 1995.

[PV2] B. Preneel and P. van Oorschot, "On the security of two MAC algorithms," *Advances in Cryptology – Eurocrypt* 96 *Proceedings*, Lecture Notes in Computer Science Vol. ??, U. Maurer ed., Springer-Verlag, 1996.

[Ri] R. Rivest, "The MD5 message-digest algorithm," IETF Network Working Group, RFC 1321, April 1992.

[SHA] FIPS 180-1. Secure Hash Standard. Federal Information Processing Standard (FIPS), Publication 180-1, National Institute of Standards and Technology, US Department of Commerce, Washington D.C., April 1995.

[To] J. Touch, "Performance Analysis of MD5", *Proceedings of Sigcomm '95,* pp. 77-86. (See also RFC 1810).

[Ts] G. Tsudik, "Message authentication with one-way hash functions," *Proceedings of Infocom 92.*

[VW] P. van Oorschot and M. Wiener, "Parallel Collision Search with Applications to Hash Functions and Discrete Logarithms", Proceedings of the 2nd ACM Conf. Computer and Communications Security, Fairfax, VA, November 1994.

[X9.9] ANSI X9.9, "American National Standard for Financial Institution Message Authentication (Wholesale)," American Bankers Association, 1981. Revised 1986.