This is a draft. Please do not distribute!

Keying Hash Functions for Message Authentication

MIHIR BELLARE[1] RAN CANETTI[2] HUGO KRAWCZYK[3]

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, simple, and practical 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, the security of our schemes is *tightly* related to that of the hash function.

The performance of our schemes 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.

[1] Contact Author. 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. Web Page: http://www-cse.ucsd.edu/users/mihir.

[2] Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139. Email: canetti@theory.lcs.mit.edu.

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

1 Introduction

This paper is about the authentication of information using cryptographic hash functions like MD5 and SHA, an issue brought to the fore by the security needs of the Internet.

1.1 Authenticity and MACs

Verifying the integrity and authenticity of information is a prime necessity in computer and network systems. 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. In the case in which the parties base this method on a shared secret key this mechanism is usually called 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]

Tampering of information is to be detected with very high probability even if the alteration was made by a powerful adversary, as long as this adversary is denied the secret key. 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. Typically, the adversary gets to see many pairs of messages and their corresponding authentication tags as exchanged by the parties; in the general case, the attacker may also choose or influence these messages. The quality of a MAC mechanism is quantified by its resistance to forgery as a function of the resources available to the adversary, namely, the time spent in the attack and the number of messages (or total information) authenticated by the parties and seen by the adversary. A formal definition of a MAC that captures this notion is presented in Section 2.1.

1.2 MACing with hash functions

With the development of security protocols for Internet, the need has arisen to have simple, efficient, and widely available MAC mechanisms. A strong trend in the development of such mechanisms has been to base them on *cryptographic hash functions*, primarily on the MD5 function designed by Rivest [Ri], and more recently on SHA (the US Standard Hash Algorithm [SHA]). Among the main reasons for the adoption of these functions for message authentication one can list the wide availability of software and implementations for these functions, their efficiency in software implementations, their license-free nature, and the fact that these functions are not subject to the export restriction rules of the USA and other countries.

The technical side, however, is more problematic. Hash functions were not originally designed to be used for message authentication. (A principal difficulty is that hash functions are not keyed primitives.) So special care must be taken in using them to this end. In particular, proposed constructions lack a sound and realistic security analysis. Thus there is a need for constructions which fulfill the efficiency and other requirements of the Internet but are backed by a more rigorous analysis of their security. This is what we provide.

[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.

Section 1.5 describes some background and 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. Our analysis considers a generic and complete attack on MAC schemes (rather than showing its 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, 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 the other, more standard cryptographic usages for which these functions were initially designed.

EFFICIENCY. Our constructions use the cryptographic hash functions (like MD5) 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 (see [To] for a discussion of these performance needs).

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, 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 2.2). 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.

[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.

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 constrast 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 functions through their *initial variable* (IV) (for details see Section 2.2). 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. Clearly, if the basic hash function has poor cryptographic strength, e.g. it is easily invertible or predictable, there is little one can do to build on top of it a secure message authentication function. On the other hand, if one makes very strong assumptions on the cryptographic properties of the underlying hash scheme (e.g., that it behaves as a completely random function) then it is easy to come up with authentication schemes whose security can be argued based on those very strong assumptions. 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; or, (2) The attacker is able to compute an output of the compression function even with an IV that is random, secret and unknown to the attacker. (That is, the attacker is succesful in forging with respect to the application of compression function secretly keyed and viewed as a MAC on fixed length messages.)

The feasibility of any of these attacks would contradict some of our 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 hash functions, and thus can be assumed hard to do. 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 (as is the case here) is far more difficult than finding collisions in the plain (fixed IV) hash function. This is because the former requires 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. One usually assumes that it is hard to even predict the value of a *single* bit of output; yet the task for the attacker here is to simultaneously predict *all* the bits of output, and this task is even harder.

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.

1.5 Related work

The exact security treatment of MACs began in [BKR], and we use their definitions. The CBC MAC was analyzed in [BKR], and further block cipher based constructions were provided and analyzed in [BGR]. Constructions based on universal hashing were provided and analyzed in [WC, Kr, Ro].

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 long 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 5 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.

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 [BCK] we consider how to design "pseudo-random functions" out of compression functions. We study the natural way of keying Merkle's construction (which underlines the design of iterated hash functions) and 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).

2 Basic Notions

In this section we summarize definitions for MACs, recall the iterated construction of hash functions, discuss our way of keying these hash functions, and discuss collision-resistance of the resulting keyed hash functions.

2.1 Message Authentication Codes (MACs)

A MAC is a function which takes the secret key k (shared between the parties) and the message m to return a tag $\mathrm{MAC}_k(m)$. 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, $a_i = \mathrm{MAC}_k(m_i)$) transmitted between the parties. We consider chosen-message attacks, and think of the adversary as being able to choose the messages for which she wants to see tags. She 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 $a = \mathrm{MAC}_k(m)$. (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 success probability of the adversary is the probability that she breaks the MAC.

We adopt the definition of [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 time t available to her to attack the system. (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 t (number of operations) in the attack, and to request the value of the function MAC_k in

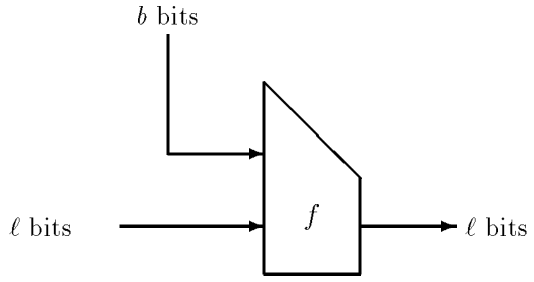

Figure 1: *A compression function.* ($b = 512$, $\ell = 128$ *in MD5.).*

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. 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.

2.2 Cryptographic Hash Functions

A function is collision-resistant if it is hard to find distinct points x, y in its domain which have the same image under the function. A cryptographic hash function takes messages of aribtrary length to strings of some fixed length, and is usually designed to be collision-resistant.

ITERATED CONSTRUCTIONS. The iterated construction methodology for the construction of collision-resistant hash functions is due to Merkle [Me] and Damgard [Da], and forms the basis of the design of the most common cryptographic hash functions like MD5 and SHA. We start with a *compression function* f which processes fixed length inputs: it takes a *chaining variable* of length ℓ and a block of data of length b to return a ℓ bit output. ($\ell = 128$ and $b = 512$ for MD5 while $\ell = 160$ and $b = 512$ for SHA. For the detailed description of the compression functions of these functions see [Ri, SHA].) See Figure 1.

A fixed value is chosen as the initial variable IV. Now a long 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, the value of the iterated function F on x is h_n where $h_0 = IV$ and $h_i = f(h_{i-1}, x_i)$ for $i = 1, 2, \ldots, n$. 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.

2.3 Keyed Hash Functions

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. As indicated before, 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 k known only to the parties. As it turns out 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 it can be achieved through the use of keys padded and prepended to data (see Section 4).

Accordingly 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) a family of (keyed) functions $\{{}_k\}_k$. Namely for $x = x_1 \ldots x_n$ define ${}_k(x)$ to be k_n where $k_i = f_{k_{i-1}}(x_i)$ for $i = 1, \ldots, n$ and $k_0 = k$. Notice that the space of keys is the same for the keyed compression functions and the keyed iterated hash functions, and it consists 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}.

We are interested in the the notion of collision-resistance for keyed functions. 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 $m_1, m_2, \ldots, m_q$ 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 much weaker than the traditional requirement of collision-resistance from (keyless) hash functions. In the latter case, it is enough to find collisions for a known and fixed IV.[3] In our case the added difficulty comes from the fact that k is chosen at random and it is *hidden from the adversary.* Most importantly, in the case of secretly keyed hash functions the adversary needs to get its "examples" (the messages $m_1, m_2, \ldots, m_q$ in the above definition) from the legal user itself who knows the key k. In the keyless 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.

[3] Some care is needed here. In principle, it could be the case that a function has easy collisions for most keys but hard-to-find ones for a particular value of an IV. However, this does not happen for iterated constructions, since the case of fixed IV can be reduced to that of a random (and known) IV by using messages which include a randomly chosen first block. Details are omitted.

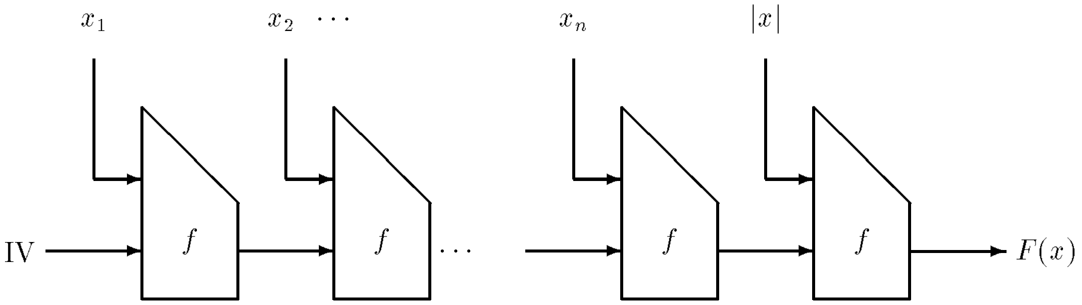

Figure 2: *The iterated construction of a hash function given a compression function f. The input $|x|$ to the last iteration illustrates the appending of the message length as defined in MD5. ($b = 512$, $\ell = 128$ in MD5.)*

3 The Nested Construction NMAC

We present our basic construction NMAC and its analysis. Denote by f_k and F_k the keyed versions of a given compression function and its iterated function, as described in Section 2.3.

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

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

Notice that the outer function acts on the output of the iterated function and thus 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.

Note NMAC uses the compression function as a black box. Sometimes one doesn't want to do this, but rather wants to use only the entire iterated function as a black box. A variant of NMAC called HMAC, presented in the next section, achieves this.

Our construction, if unraveled might at first glance bear some similarity to the prepend-append construction. Actually, one can see (and we will see later) that the constructions are different.

The following result (tightly) relates the security of NMAC to the security of the hash function as a keyed collision-resistant function and the security of the compression function as a MAC on fixed length messages. The proof can be found in Appendix A.

Theorem 3.1 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 but 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. Next we present a few remarks on the above analysis.

Remark 3.2 The analysis shows a more general result than stated in the theorem. Notice that although we use the same function F 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 MD5 (keyed through its IV) for the internal iterated function, and DES-MAC-CBC for the external, etc.

Remark 3.3 One has to be careful not to attribute more security to the above function than stated in the analysis. Notice that this analysis states that NMAC is as strong, *but not stronger*, than either the keyed compression function as MAC or the keyed iterated function as collision-resistant. In particular, 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 5.

Remark 3.4 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 5), 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 3.5 One can ask whether 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 5. 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 (formally, this requires the assumption that collision-resistant functions exist in the first place). Moreover, one can show this to hold for specific proposals of MAC functions based on hash schemes.

4 HMAC: A 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 construction NMAC presented in Section 3 requires direct access to code for the compression function (rather than the hash function), and a slight modification to it in order to key its 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 an hardware implementation) as is. Here we present an adaptation of NMAC that has as its goal to provide with direct use of the iterated function (together with its defined fixed IV) as the basic black-box to build the MAC. 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. Yet with an additional assumption on the underlying compression function one can show the applicability of the analysis and security of NMAC to HMAC.

Let F be the (iterated and keyless) hash function (e.g., MD5 or SHA) 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:

$$\mathrm{HMAC}_k(x) = F(k, \mathrm{pad}_1, F(k, \mathrm{pad}_2, x))$$

where the commas represent concatenation of the information, pad_1 is formed by repeating the byte x'36' as many times as needed to pad the ℓ-bit long key k to an b-bit block, and pad_2 is defined similarly using the byte x'5c'. (For example, in the case of MD5 these bytes will be repeated 48 times, while in SHA only 44 times).

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

$$k_1 = f(k, \mathrm{pad}_1), \quad k_2 = f(k, \mathrm{pad}_2), \tag{1}$$

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 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 cannnot 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 this assumption 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. Therefore, attacks that work on HMAC and not on NMAC are possible, in principle. However, such an attack would reveal significant weaknesses of the pseudorandom properties of the underlying hash function, contradicting in a strong sense the usual assumptions on these functions.

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 is very common in practical implementations (e.g., the designers of SHA recommended using this function for pseudorandom derivation for the DSS standard [DSS]).

The above particular values of pad_1 and pad_2 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. If there is an objection to the fact that these are too "structured" pads, one can replace them by other (fixed and public) values.

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 5.

See Appendix B for some implementation considerations.

5 Attacks and Comparison to other Proposals

In Section 3 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 all known attacks 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 known ones. Here we discuss only a few well-known attacks and constructions; other attacks are discussed in Appendix C.

Birthday attacks. As shown in [PV1] and our companion work (reference omitted for anonymity), 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 the constructions mentioned above as well as to our new constructions. 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. Indeed, these attacks require the collection 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.[4] In contrast to the birthday attack on keyless 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 need 250,000 years to process all the data required by such an attack. This is in sharp contrast to birthday attacks on keyless 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 check their applicability to new constructions. It is easy to see that they do not apply to either of our constructions, where the alignment issue exploited by these attack is no applicable (see [PV2] for further details on this attack).

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 in a per-message basis the MAC construction.

Collision attacks on the keyless 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 keyless 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. While the latter is totally infeasible even for hash lengths of $\ell = 128$ (see discussion on birthday attacks below), finding collisions to the plain hash function approaches feasibility (see [VW]). The reason is that such a collision attack on the plain hash function can be

[4] 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 $\mathrm{NMAC}_k(m_1) = \mathrm{NMAC}_k(m_2)$. 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 $\mathrm{NMAC}_k(m_1')$. If the collision $\mathrm{NMAC}_k(m_1) = \mathrm{NMAC}_k(m_2)$ 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.

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. Moreover, finding collisions in the keyless function has applications beyond attacking the MAC scheme (especially, for forging signatures), and then it is more likely to find organizations that would spend the resources needed to find such collisions. None of these advantages for the attacker exist when attacking NMAC. In addition, it is far more plausible to find collisions via analytical methods (e.g., [Do]) against the keyless function than against the keyed function. 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 then significantly weaker than NMAC.

Divide and conquer attacks. Consider the method, known as the "envelope" method, that combines the above prepend and append constructions, namely, $\mathrm{MAC}_{k_1,k_2}(x) = F(k_1, x, k_2)$. 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_2) that produces these collisions. Once we have the right k_2 it is straightforward to find k_1 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.

The same attack holds against NMAC. This is in no contradiction with the analysis of Section 3 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 3.1 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. This is a heuristic measure intended to counter possible weaknesses of the compression function in use, though 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.

See Appendix C for discussion of other attacks and related schemes.

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.

[BCK] M. Bellare, R. Canetti and H. Krawczyk, "How to key Merkle: Cascaded pseudorandomness and its concrete security."

[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.

[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.

[Do] H. Dobbertin, "MD4 is not collision-free," Manuscript, September 1995.

[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).

[GMR] S. Goldwasser, S. Micali and R. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM Journal of Computing,* 17(2):281–308, April 1988.

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

[Kr] Krawczyk, H., "LFSR-based Hashing and Authentication", *Advances in Cryptology – Crypto* 94 *Proceedings*, Lecture Notes in Computer Science Vol. 839, Y. Desmedt ed., Springer-Verlag, 1994.

[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," To appear in *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.

[Ro] P. Rogaway, "Bucket Hashing and its Application to Fast Message Authentication," *Advances in Cryptology – Crypto* 95 *Proceedings*, Lecture Notes in Computer Science Vol. 963, D. Coppersmith ed., Springer-Verlag, 1995.

[SHA] FIPS 180, "Secure Hash Standard", Federal Information Processing Standard (FIPS), Publication 180, National Institute of Standards and Technology, US Department of Commerce, Washington D.C., May 1993.

[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.

[WC] M.N. Wegman and J.L. Carter, "New Hash Functions and Their Use in Authentication and Set Equality", *JCSS*, 22, 1981, pp. 265-279.

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

A Proof of Theorem 3.1

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 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 $\mathrm{NMAC}_k(x_1)$, it then produces a second query x_2 for which it gets $\mathrm{NMAC}_k(x_2)$, 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 $y = \mathrm{NMAC}_k(x)$ 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

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

Figure 3: The algorithm A_f

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,

$$F_{k_2}(x) = F_{k_2}(x_i). \tag{2}$$

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 (2) 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 (2) 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 (2) 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 (2) 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 $(1 - \epsilon_N) + \epsilon_F$ or, in other words, $\epsilon_f \geq \epsilon_N - \epsilon_F$. Therefore, $\epsilon_N \leq \epsilon_f + \epsilon_F$ and the theorem follows.

B 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 (k_1, pad_1) and (k_2, pad_2)). 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. That is, these values are computed only once (as in (1)) 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, 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 5) 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.

C Further comparisons and discussion of attacks

We continue Section 5 by discussing more existing attacks and constructions. Several of the constructions mentioned below appear in [Ts]. For a survey of constructions and a detailed description of some of the attacks see [PV1] or [KR].

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}.

COMPARISON WITH RFC1828. It is particularly interesting to compare the new constructions with the MAC scheme described in RFC1828 [MS] which 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 [BCK] 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 3.1) than in the analysis in [BCK]. Another important difference is that the HMAC variant deals better with the use of a single key than done in the construction of RFC1828; in the latter, the use of the same key for prepend and append makes the analysis in [BCK] less applicable and, in particular, makes the scheme susceptible to the above mentioned key recovery attack of [PV2].