An Efficient MAC for Short Messages

Sarvar Patel
Lucent Technologies
67 Whippany Road
Whippany, NJ 07981
sarvar@lucent.com

Abstract

HMAC is the internet standard for message authentication [BCK96,KBC97]. What distinguishes HMAC from other MAC algorithms is that it provides proofs of security assuming that the underlying cryptographic hash (e.g. SHA-1) has some reasonable properties. HMAC is efficient for long messages, however, for short messages the nested constructions results in a significant inefficiency. For example to MAC a message shorter than a block, HMAC requires at least two calls to the compression function rather than one.

This inefficiency may be particularly high for some applications, like message authentication of signaling messages, where the individual messages may all fit within one or two blocks. Also for TCP/IP traffic it is well known that a large number of packets (e.g. acknowledgement) have sizes around 40 bytes which fit within a block of most cryptographic hashes. We propose an enhancement that allows both short and long messages to be message authenticated more efficiently than HMAC while also providing proofs of security. In particular, for a message smaller than a block our MAC only requires one call to the compression function.

1 Introduction

Message Authentication. The design of good Message Authentication schemes is one of the important areas of cryptography. The goal in message authentication is for one party to efficiently transmit a message to another party in such a way that the receiving party can determine whether or not the message he receives has been tampered with. The setting involves two parties, Alice and Bob, who have agreed on a pre-specified secret key x. There are two algorithms used: a signing algorithm S_x and a verification algorithm V_x. If Alice wants to send a message M to Bob then she first computes a message authentication code, or MAC, $\mu = S_x(M)$. She sends (M, μ) to Bob, and upon receiving the pair, Bob computes $V_x(M, \mu)$ which returns 1 if the MAC is valid, or returns 0 otherwise. In other words, without knowledge of the secret key x, it is next to impossible for an adversary to construct a message and corresponding MAC that the verification algorithm will be accept as valid.

The formal security requirement for a Message Authentication Code was defined in [BKR94]. This definition was an analog to the formal definition of security for digital signatures [GMR88]. In particular, we say that an adversary forges a MAC if, when given oracle access to (S_x, V_x), where x is kept secret, the adversary can come up with a valid pair (M^*, μ^*) such that $V_x(M^*, \mu^*) = 1$ but the message M^* was never made an input to the oracle for S_x.

Cryptographic Hash Function Approach. One common approach to message authentication commonly seen in practice involves the use of cryptographic hash functions such as $MD5$

and SHA-1. These schemes are good because they use fast and secure cryptographic building blocks. Creating a MAC from a hash function may seem deceptively easy. It seems all one needs to do is to put a key somewhere in the hash function. A easy way to accomplish this is to prepend the key to the data before hashing. Unfortunately, this is not secure and one can forge MACs on unseen messages as we described next. To forge a MAC for the prepend key construction, we begin with a previously MACed message x and its tag $MAC_k(x)$. We create a new message z by appending any message y to x, z = (x,y). To create the tag $MAC_k(z)$ we use $MAC_k(x)$ as the chaining variable and hash y, making sure to set the length of the message to be the length of z. The result will be $MAC_k(z)$. Thus we have forged the MAC for z.

HMAC, the mandatory internet standard for message authentication, uses the cryptographic hash approach. What distinguishes HMAC from other cryptographic hash based MACs is its formal security analysis. The analysis provides a proof of security of HMAC assuming the underlying cryptographic hash is (weakly) collision resistant and that the underlying compression function is a secure MAC when both are appropriately keyed.

This Work. HMAC is efficient for long messages, however, for short messages the nested constructions results in a significant inefficiency. For example to MAC a message shorter than a block, HMAC requires at least two calls to the hash function rather than one. This inefficiency may be particularly high for some applications, like message authentication of signaling messages, where the individual messages may all fit within one or two blocks. Also for TCP/IP traffic it is well known that a large number of packets (e.g. acknowledgement) have sizes around 40 bytes which fit within a block of most cryptographic hashes. We propose an enhancement that allows both short and long messages to be message authenticated more efficiently than HMAC while also providing proofs of security. For a message smaller than a block our MAC only requires one call to the hash function.

2 Preliminaries

Cryptographic hash functions Cryptographic hash functions, $F(x)$, are public, keyless, and collision-resistant functions which map inputs, x, of arbitrary lengths into short outputs. Collision-resistance implies that it should be computationally infeasible to find two messages x_1 and x_2 such that $F(x_1) = F(x_2)$. MD5, SHA-1, and RIPE-MD are widely used cryptographic hash functions. Along with collision-resistance, the hash functions are usually designed to have other properties both inorder to use the function for other purposes and to increase the likelihood of collision-resistance.

Iterated cryptographic hash functions Most cryptographic hash functions like MD5 and SHA-1 use an iterated construction where the input message is processed block by block. The basic building block is called the compression function, f which takes two inputs of size l and b and maps into a shorter output of length l. The l sized input is called the chaining variable and the b sized input is used to actually process the message b bits at a time. The hash function $F(x)$ then is formed by iterating the compression function f over the message m using these steps:

1. Use an appropriate procedure to append the message length and pad to make the input a multiple of the block size b. The input can be broken into block size pieces $x = x_1, \ldots, x_n$.

2. $h_0 = IV$, a fixed constant.

3. For $i = 1$ to n

$h_i = f(h_{i-1}, x_i)$

4. output h_n as $F(x)$.

KEYED HASH FUNCTIONS Cryptographic hash functions by design are keyless. However, since message authentication requires the use of a secret key, we need a method to key the hash function. The method we use in this paper is the same as that used by NMAC [BCK96] where a secret key is used instead of the fixed and known IV. In this case the key k replaces the chaining variable in the compression function $f(chainingvariable, x)$ to form $f_k(x) = f(k, x)$ where x is of block size b. The iterated hash function $F(IV, x)$ is modified by replacing the fixed IV with the secret key k to form $F_k(x) = F(k, x)$. Collision resistance for keyed function is different because the adversary cannot evaluate $F_k(x)$ at any points without querying the user. This requirement is weaker than the standard collision requirement and hence we will call the function $F_k(x)$ to be weakly collision-resistant [BCK96] .

3 NMAC and HMAC

HMAC is a practical variant of the NMAC construction defined in [BCK96]. The formal security proofs are given for NMAC and its relation to HMAC is described. First we describe the NMAC construction and state the theorem proved in [BCK96].

3.1 The NMAC Function

The message authentication function NMAC is defined as

$$NMAC_k(x) = F_{k_1}(F_{k_2}(x)).$$

where the cryptographic hash function is first keyed with the secret key k_2 instead of IV and the message x is hashed to the short output. This output is then padded to a block size according to the padding scheme of F and then F is keyed with secret key k_1 and hashed. Thus the NMAC key k has two parts $k = (k_1, k_2)$. The following theorem about relating the security of NMAC to the security of the underlying cryptographic hash function is proved in [BCK96]:

Theorem 1 *In t steps and q queries if the keyed compression function f is an ϵ_f secure MAC, and the keyed iterated hash F is ϵ_F weakly collision-resistant then the NMAC function is $(\epsilon_f + \epsilon_F)$ secure MAC.*

3.2 Efficiency of NMAC

The NMAC construction makes two calls to F, the inner call to $F_{k_2}(x)$ has the same cost as the keyless hash function $F(x)$. Thus the outer call to $F_{k_1}()$ is an extra call beyond that required by the keyless hash function. The outer function call is basically a call to the keyed compression function $f_{k_1}()$ since the l size output of $F_{k_2}(x)$ can fit in the b size input to the compression function. For large x consisting of many blocks the cost of the extra outer compression call is not significant. However, for small sized messages x the extra compression function can in terms of percentage result in a significantly high inefficiency when compared to the unkeyed hash function. Figure 1 shows the inefficiency for small x for the SHA-1 hash function. The number of compressions calls needed by the underlying hash function and by NMAC are compared for various small x, increasing

x in 240 bit increments	# of f in $F(x)$	# of f in NMAC	% inefficiency
240	1	2	100%
480	2	3	50%
720	2	3	50%
960	3	4	33%
1200	3	4	33%
1440	3	4	33%
1680	4	5	25%
1920	4	5	25%
2160	5	6	20%
2400	5	6	20%

Figure 1: Comparison in number of compression calls for short messages of various sizes.

in 30 byte increments. The inefficiency of NMAC with respect to the underlying hash function is also noted in the figure.

As can be seen the penalty for small messages can be large. In particular, for messages which fit within a block, the penalty is 100% because two compression function calls are required in NMAC versus one call by the underlying cryptographic hash function.

3.3 The HMAC Function

HMAC is a practical variant of NMAC for those implementations which do not have access to the compression function, but can only call the cryptographic hash function F with the message. That is the key cannot be placed in the chaining variable and the function F with the fixed and known IV is called.

$$HMAC_k(x) = F(\bar{k} \oplus opad, F(\bar{k} \oplus ipad, x))$$

where $\bar{k}$ is the padding of k with zeroes to complete the b block size of the iterated hash function. Also ipad and opad are fixed constants, see [BCK96]for details. HMAC requires 4 calls to the compression function, the first time its evaluated. If the intermediate keys can be cached and loaded directly into the chaining variable then HMAC subsequently requires at least two calls. If direct loading of the keys into the chaining variable is not possible then HMAC continues to require at least 4 calls to the compression function. HMAC's security relies on an added assumption that the compression function has some pseudorandom properties.

4 ENMAC: Enhanced NMAC

We present a MAC construction which is not only significantly more efficient than NMAC for short messages but is also somewhat more efficient for longer messages. Recall that $f_k(x)$ is the compression function whose input block size is b bits and the output size is l bits, also the size of the chaining variable and hence the key size also is l bits.

$$\begin{aligned} ENMAC_k(x) &= f_{k_1}(x, pad, 1) && if\ |x| <= b-2\ bits \\ &= f_{k_1}(x_{pref}, F_{k_2}(x_{suff}), 0) && else \end{aligned}$$

where in the first case the first $b-2$ bits in the block are used to hold the message x. If the message x does not fill the block completely, then padding is required and the remaining block, except the last bit, is filled with a mandatory 1 followed by 0s, possibly none. In the case that the message is $b-2$ bits long, the $b-1th$ bit is set to 1. The last bit of the block, indicates whether a single compression call is used for ENMAC. The last bit of the block is set to 1 in the single compression call case, and is set to 0 when multiple compression calls are required which we describe next. The padding scheme for the single block case is unambiguous in the sense that every message, x, less than or equal to $b-2$ bits is uniquely mapped to a b bit string.

In the second case where things will not fit in one block, the string x is broken into two pieces x_{pref} and x_{suff}, where

$$x_{pref} = x_1 \ldots x_{b-l-1}$$

$$x_{suff} = x_{b-l} \ldots x_{|x|}$$

First x_{suff} is hashed using k_2 to produce the l bit tag. Then an outer compression call is performed using k_1 where the first $b-l-1$ bits are set to x_{pref} and the next l bits are set to the tag $F_{k_2}(x_{suff})$, and the last bit is set to 0.

4.1 SHA-1-ENMAC

The ENMAC construction, described abstractly above, may become clearer when we look at the concrete case of using SHA-1 as the underlying cryptographic hash function.

1. If $|x| \leq 510$ bits then goto next step else goto step 7.
2. The 512 bit payload of $f_{k_1}()$ is formed by loading x into the first 510 bits.
3. Append a 1 to x.
4. Append as few 0s (possibly none) as needed to fill 511 bits. If $|x|$ is less than 510 bits than zeroes will be padded beyond the 1 or else if $|x|$ is 510 bits then no zeroes are padded and only a single 1 is added at the 511-th bit position.
5. The last 512-th bit is set to 1, to indicate the message fits in a single block.
6. Apply the keyed compression function on the payload and output the result.
7. Split x into two pieces x_{pref} and x_{suff} where $x_{pref} = x_1 \ldots x_{351}$ and $x_{suff} = x_{352} \ldots x_{|x|}$.
8. Apply the keyed hash function $F_{k_2}(x_{suff})$.
9. Form the 512 bit payload of f_{k_1} by setting the last bit to 0.
10. Set the first 351 bits to be x_{pref}.
11. Set the next 160 bits to be $F_{k_2}(x_{suff})$ calculated in step 8.
12. Apply the keyed compression function on the payload and output the result.

x in 240 bit increments	# of f in $F(x)$	# of f in ENMAC	% inefficiency
240	1	1	0%
480	2	1	-50%
720	2	2	0%
960	3	3	0%
1200	3	3	0%
1440	3	4	33%
1680	4	4	0%
1920	4	5	25%
2160	5	5	0%
2400	5	6	20%

Figure 2: Comparison in number of compression calls for short messages of various sizes.

4.2 Efficiency of ENMAC

Figure 2 below compares the number of compression calls required by the underlying hash function, SHA-1, and by ENMAC for short messages varying in sizes of 30 byte increments.

We can see a significant difference between this figure and the previous figure which compared plain NMAC. For many of the short sizes ENMAC has the same efficiency as the underlying hash function, For larger messages the efficiency of NMAC, ENMAC and the underlying hash function will not be significantly different from each other. For message of size 480 bits the entry in figure 2 surprisingly indicates that the ENMAC is more efficient than the underlying hash function! This anomaly occurs because the underlying SHA-1 function reserves 64 bits for size information while ENMAC reserves only 2 bits for messages less than 510 bits. Thus the savings resulting from using ENMAC are significant for messages that fit in one or few of blocks.

4.3 Security of ENMAC

MOTIVATION If we were to use a different key k_3 to MAC messages which fit in one block and use key $k = (k_1, k_2)$ to MAC larger messages using NMAC then we could argue the system would be secure. Essentially, this is what is being done, but instead of using a different key to create a different MAC, the trailing bit is being set to 1 if the message fits in one block and its set to 0 for the other case. Secondly, whereas NMAC pads the payload of the outer compression call with zeroes, we fit part of the message in the outer call. Our security results are similar to NMAC and we state our security theorem and prove it next.

Theorem 2 *In t steps and q queries if the keyed compression function f is an ϵ_f secure MAC, and the keyed iterated hash F is ϵ_F weakly collision-resistant then the ENMAC function is $(\epsilon_f + \epsilon_F)$ secure MAC.*

Proof: Suppose an adversary A_E is successful against ENMAC with probability ϵ_E assuming t time steps and q adaptively chosen queries to the ENMAC function. We will use this adversary to build another adversary A_f which will forge a MAC associated with the keyed compression function on a previously unqueried message. We will bound this probability of breaking the MAC in terms of ϵ_E and ϵ_F, where ϵ_F is the best probability of an adversary finding a collision in the hash function F in time t and q queries. We know that the probability of breaking the MAC in this particular way, using A_E, has to be less than the best probability of breaking the MAC in any way, ϵ_f. We use this in turn to get a bound on ϵ_E. Figure 3 outlines the algorithm A_f which we describe next.

A_f algorithm will attack the MAC f by first querying it on $x_1 \dots x_q$ and get $f_{k1}(x_1) \dots f_{k1}(x_q)$. Inside A_f, the algorithm A_E is being run; A_E itself will want to query the ENMAC function before it attacks the ENMAC. Actually there is no real ENMAC function that we are attacking, the real thing we want to attack is the MAC $f_{k1}()$. So we are simulating an ENMAC function to the A_E algorithm by answering its query correctly. We have picked a random k_2, and when A_E asks us to form ENMAC of messages x_i we will use the key k_2 to form the inside calculation part $F_{k2}()$, and for the outer part of ENMAC, which is $f_{k1}()$, we actually call the real MAC that we want to attack. When the A_E queries for $ENMAC(x_i)$ we see if x_i will fit in a single block or requires multiple block. If it fits in a single block then we just make a call to the real MAC function $f_{k1}()$ with the arguments $(x_i, pad, 1)$. The indicator bit is used here and set to 1. On the other hand if multiple blocks will be used then we first evaluate $F_{k2}(x_{i,suff})$ then we make a call to the real MAC with the arguments $(x_{i,pref}, F_{k2}(x_{i,suff}), 0)$. The indicator bit is set to zero.

Now suppose the A_E algorithm has forged an ENMAC tag y on an unseen, unqueried message x and the message fits in one block. ENMAC tag on a small message x is equivalent to $f_{k1}()$ MAC tag on unseen message $(x, pad, 1)$. Now this is the crucial role about the indicator bit. The pair $(x, y = ENMAC(x))$ was never queried or requested by the A_E algorithm inside the A_f algorithm. Thus all the previous calls to the real $f_{k1}()$ MAC made on behalf of the A_E queries never made the real MAC call with argument $(x, pad, 1)$! This we see because the only calls to the real MAC that were made were either $f_{k1}(...., 1)$ on behalf of single block ENMAC queries or $f_{k1}(...., 0)$ on behalf of multiple block ENMAC queries, no other calls to the real MAC were made. So the indicator bit creates an independence from the multiple block ENMAC case, and allows us to just see that among all the single block queries, none was made with x, thus this is a new forged MAC pair. By our direct assumption that we cannot forge new message MAC pair on $f_{k1}()$, this is a contradiction. Hence there could not have been an attack on ENMAC for single block. We also note that the unambiguous padding scheme used in ENMAC guarantees that for every message x queried by $ENMAC(x)$ means there is a unique message $(x, pad, 1)$ queried by the real MAC $f_{k1}(x, pad, 1)$. If the multiple block ENMAC was what was forged first by A_E then we have to show that this allows A_f to forge new messages and MAC pair on $f_{k1}()$. We formally prove the theorem below, but first we define some useful events and their probabilities:

$$
\begin{aligned}
A_f\ forges &: \ the\ event\ where\ A_f\ correctly\ forges\ a\ mac\ of\ f_{k1}() \\
E &: \ the\ event\ where\ A_E\ correctly\ forges\ a\ mac\ of\ ENMAC() \\
\bar{E} &: \ the\ event\ where\ A_E\ fails\ to\ correctly\ forge\ a\ mac\ of\ ENMAC() \\
E_1 &: \ the\ event\ where\ A_E\ correctly\ forges\ (x, y = ENMAC(x))\ and\ message\ x \\
&\quad\ fits\ in\ a\ single\ block \\
E_+ &: \ the\ event\ where\ A_E\ correctly\ forges\ (x, y = ENMAC(x))\ and\ message\ x \\
&\quad\ requires\ multiple\ blocks \\
E_{+,pref\neq} &: \ the\ event\ where\ A_E\ correctly\ forges\ (x, y = ENMAC(x)\ and\ message\ x \\
&\quad\ requires\ multiple\ blocks\ and\ x_{pref}\ \neq\ x_{i,pref}\ of\ any\ queried\ messages \\
&\quad\ x_i \in x_1\ \dots x_q \\
E_{+,pref=} &: \ the\ event\ where\ A_E\ correctly\ forges\ (x, y = ENMAC(x)\ and\ message\ x \\
&\quad\ requires\ multiple\ blocks\ and\ x_{pref}\ =\ x_{i,pref}\ of\ some\ queried\ message
\end{aligned}
$$

Choose random k_2
For $i = 1 \ldots q$ do
    $A_E \rightarrow x_i$
    If $x_i \leq b-2$
        $A_E \leftarrow f_{k_1}(x_i, pad, 1)$
    else
        $A_E \leftarrow f_{k_1}(x_{i,pref}, F_{k_2}(x_{i,suff}), 0)$
$A_E \rightarrow (x, y)$
If $x \leq b-2$
  output $(x, pad, 1), y$
else
  output $(x_{pref}, F_{k_2}(x_{suff}), 0), y$

Figure 3: A_f algorithm to forge the keyed compression MAC

$$
\begin{array}{rcl}
 & & x_i \in x_1 \ \ldots x_q \\
\epsilon_E & : & P(E) \\
\epsilon_{E_1} & : & P(E_1) \\
\epsilon_{E_+} & : & P(E_+) \\
\epsilon_{E_+,pref\neq} & : & P(E_{+,pref\neq}) \\
\epsilon_{E_+,pref=} & : & P(E_{+,pref=})
\end{array}
$$

We note that $\epsilon_E = \epsilon_{E_1} + \epsilon_{E_+}$ where E_1 is the event and ϵ_{E_1} is the probability that ENMAC is attacked and the ENMAC message forged by A_E is about one block size, or to be precise less than $b-2$ bits. And let E_+ be the event and ϵ_{E_+} be the probability that ENMAC is attacked and the ENMAC message forged by A_E is larger than one block size. Furthermore, $\epsilon_{E_+} = \epsilon_{E_+,pref\neq} + \epsilon_{E_+,pref=}$ where $\epsilon_{E_+,pref\neq}$ is the probability that the ENMAC is forged with a multi block message and the prefix of the message does not equal the prefix of any of the messages previously queried by A_E. And $\epsilon_{E_+,pref=}$ is the probability that the ENMAC is forged with a multi block message and the prefix of the message is equal to prefix of some previously queried messages by A_E. In this case we know the suffix of the forged message has to be different than the suffix of the messages with the same prefix. We begin the proof:

$$
\begin{aligned}
P[A_f \ forges] &= P[A_f forges \ \cap \ E] + P[A_f \ forges \ \cap \ \bar{E}] && (1) \\
&= P[A_f forges \ \cap \ E] && (2)
\end{aligned}
$$

In equation 1, $P[A_f \ forges \cap \bar{E}] = 0$ because when A_E fails and outputs a message/tag pair (x, y) where $y \neq ENMAC(x)$ then A_f will also output incorrect message/tag pair. In the case of a single block message, A_f will output the message/tag pair $([x, pad, 1], y)$ but since $y \neq ENMAC(x)$ and $ENMAC(x) = f_{k1}(x, pad, 1)$ then $y \neq f_{k1}(x, pad, 1)$, thus A_f has also output an incorrect message/tag pair. Similarly, for the multi block case if A_f will output the message/tag pair $([x_{pref}, F_{k2}(x_{suff}), 0], y)$ but since $y \neq ENMAC(x)$ and $ENMAC(x) = f_{k1}(x_{i,pref}, F_{k_2}(x_{i,suff}), 0)$ then $y \neq f_{k1}(x_{i,pref}, F_{k_2}(x_{i,suff}), 0)$, thus A_f has also output an incorrect message/tag pair.

$$\begin{aligned}
P[A_f\ forges] &= P[A_f\ forges \cap E_1] + P[A_f\ forges \cap E_+] & (3)\\
&= P[A_f\ forges \cap E_1] + P[A_f\ forges \cap E_{+,pref\neq}] + P[A_f forges \cap E_{+,pref=}] & (4)\\
&= P[A_f\ forges \mid E_1]\ P[E_1] + P[A_f\ forges \mid E_{+,pref\neq}]\ P[E_{+,pref\neq}]\\
&\quad + P[A_f\ forges \cap E_{+,pref=}] & (5)\\
&= 1 \cdot \epsilon_{E_1} + 1 \cdot \epsilon_{E_{+,pref\neq}} + P[A_f\ forges \cap E_{+,pref=}] & (6)\\
&= \epsilon_{E_1} + \epsilon_{E_{+,pref\neq}} + P[A_f\ forges \cap E_{+,pref=}] & (7)
\end{aligned}$$

In equation 5, $P[A_f\ forges \mid E_1] = 1$ because when A_E correctly outputs the message/tag pair (x, y) where $y = ENMAC(x)$ then A_f outputs $([x, pad, 1], y)$ where $y = f_{k1}(x, pad, 1)$. The indicator bit 1 means we have to only consider the previously queried single block messages since the multi block messages will have the indicator bit set to 0. Among the single block messages queried by A_E, none equal x because A_E has correctly forged a new message. Thus none of the previous queries $(x_i, pad, 1)$ to $f_{k1}(x_i, pad, 1)$ would equal $(x, pad, 1)$ and hence A_f would have correctly forged a new message/tag pair. Similarly in equation 5 $P[A_f\ forges \mid E_{+,pref\neq}] = 1$ because when A_E correctly outputs the message/tag pair (x, y) where $y = ENMAC(x)$ and x is multiple blocks long then A_f outputs the message/tag pair $([x_{pref}, F_{k2}(x_{suff}), 0], y)$ where $y = f_{k1}(x_{i,pref}, F_{k_2}(x_{i,suff}), 0)$. The indicator bit 0 means we have to only consider the previously queried multi block messages. Furthermore, the probability in question is conditioned on the multi block message being of the sort where $x_{pref} \neq x_{i,pref}$, the prefix of any previously queried message. Thus this will be a new prefix x_{pref} and independently of what the suffix is, the message/tag pair $([x_{pref}, F_{k2}(x_{suff}), 0], y = f_{k1}(x_{i,pref}, F_{k_2}(x_{i,suff}), 0))$ is new and correct.

$$\begin{aligned}
P[A_f\ forges] &= \epsilon_{E_1} + \epsilon_{E_{+,pref\neq}}\\
&\quad + P[(\forall j \epsilon S_{pref=}\ (\ F_{k2}(x_{suff}) \neq F_{k2}(x_{j,suff})\ )\ )\ \cap\ E_{+,pref=}] & (8)\\
&\quad \textit{where } S_{pref=} \textit{ is the set of } x_i \epsilon\ x_1 \ldots x_q \textit{ such that } x_{i,pref} = x_{pref}\\
&= \epsilon_{E_1} + \epsilon_{E_{+,pref\neq}} +\\
&\quad P[E_{+,pref=}] - P[(\exists j \epsilon S_{pref=}\ s.t.\ (F_{k2}(x_{suff}) = F_{k2}(x_{j,suff})))\ \cap\ E_{+,pref=}] & (9)\\
&= \epsilon_E\\
&\quad - P[(\ \exists j \epsilon S_{pref=}\ s.t.\ (\ F_{k2}(x_{suff}) = F_{k2}(x_{j,suff})\ )\ )\ \cap\ E_{+,pref=}] & (10)
\end{aligned}$$

Equation 7 is transformed to equation 8 because the event $A_f\ forges \cap E_{+,pref=}$ will happen when $E_{+,pref=}$ happens and $F_{k2}(x_{suff})$ is not equal to $F_{k2}(x_{i,suff})$, the suffix of any other previously queried x_i whose prefix $x_{i,pref} = x_{pref}$. To repeat, in that case even when the prefixes are the same, $F_{k2}(x_{suff})$ is not equal to any other previously queried $F_{k2}(x_{i,suff})$. Thus the message $(x_{pref}, F_{k2}(x_{suff}), 0)$ would not have been queried before and A_f would have correctly forged a new mac $f_{k1}()$. In equation 8, the last term is really stating the probability of no collision in x_{suff} and $S_{pref=}$. In equation 9, the last term is really stating the probability of a collision in x_{suff} and $S_{pref=}$. Equation 9 can be derived from equation 8 by noting that:

$$\begin{aligned}
P[E_{+,pref=}] &= P[\textit{no collision in } x_{suff} \textit{ and } S_{pref=} \cap E_{+,pref=}]\\
&\quad + P[\textit{collision in } x_{suff} \textit{ and } S_{pref=} \cap E_{+,pref=}]\\
&\quad \textit{or more formally}\\
&= P[(\ \forall j \epsilon S_{pref=}\ (\ F_{k2}(x_{suff}) \neq F_{k2}(x_{j,suff})\ )\ )\ \cap\ E_{+,pref=}]\\
&\quad + P[(\ \exists j \epsilon S_{pref=}\ s.t.\ (\ F_{k2}(x_{suff}) = F_{k2}(x_{j,suff})\ )\ )\ \cap\ E_{+,pref=}]
\end{aligned}$$

To bound the value of $P[(\ \exists j \epsilon S_{pref=}\ s.t.\ (\ F_{k2}(x_{suff}) = F_{k2}(x_{j,suff})\ )\ ) \cap E_{+,pref=}]$ in equation 10, we will use the lemma below:

Lemma 1 $P[(\ \exists j \epsilon S_{pref=}\ (\ F_{k2}(x_{suff}) = F_{k2}(x_{j,suff})\ )\ )\ \cap\ E_{+,pref=}] < \epsilon_F$

Proof: In Appendix A.

Using the lemma to substitute in equation 10, we get the inequality in equation 11.

$$\begin{aligned} P[A_f\ forges] &> \epsilon_E - \epsilon_F &(11)\\ \epsilon_f &> P[best\ (t,q)\ algorithm\ forges\ mac\ f_{k1}()) \geq P[A_f\ forges] > \epsilon_E - \epsilon_F &(12)\\ \epsilon_f &> \epsilon_E - \epsilon_F &(13)\\ \epsilon_E &< \epsilon_f + \epsilon_F &(14) \end{aligned}$$

Equation 12 simply restates the definition of ϵ_f and notes that probability of success of A_f, a particular kind of algorithm, has to be less than or equal to probability of success by the best algorithm. This completes the proof of theorem 1. An alternate proof of theorem 1 is presented in Appendix B which more closely follows the NMAC proof style.

4.4 Practical Implementations

In the case of multiple block ENMAC, forming x_{suff}, beginning at a non-word boundary may cause us to re-align all the words in x_{suff}. We can avoid this case by using this variant (presented in the bit processing mode) of ENMAC:

$$\begin{aligned} ENMAC_k(x) &= f_{k_1}(x, pad, 1) && if\ |x| <= b-2\ bits\\ &= f_{k_1}(F_{k_2}(x_{pref}), x_{suff}, 0) && else \end{aligned}$$

where for SHA

$$\begin{aligned} x_{pref} &= x_1 \ldots x_{|x|-351}\\ x_{suff} &= x_{|x|-350} \ldots x_{|x|} \end{aligned}$$

A similar security proof described in the last section applies to this variant since the independence of the single block compression call and multiple block compression call is preserved by the last indicator bit, and in the single block case unambiguous padding is used. Other variants are possible where the single block indicator bit is placed in a different location, for example, as the first bit of the block. Also since in practice data is often processed in bytes, it may be appropriate to perform the single block case when the message $|x|$ is less than $b-8$ bits rather than the $b-2$ bits we specified.

4.5 One Key ENMAC

A variant of NMAC would be to use a keyless hash function in the inner call rather than a keyed hash function. Similar security proofs will work if one makes the assumption that the hash function is not only weakly collision resistant but also simply collision resistant.

$$\begin{aligned} One\ key\ ENMAC_k(x) &= f_{k_1}(x, pad, 1) && if\ |x| <= b-2\ bits\\ &= f_{k_1}(F(x_{pref}), x_{suff}, 0) && else \end{aligned}$$

where for SHA

$$\begin{aligned} x_{pref} &= x_1 \ldots x_{|x|-351}\\ x_{suff} &= x_{|x|-350} \ldots x_{|x|} \end{aligned}$$

5 EHMAC: Enhanced HMAC

HMAC, as previously described, is a variant of NMAC which does not require the direct loading of the key in to the chaining variable of the compression function, but only makes call to the hash function. A straightforward adaptation of ENMAC to EHMAC would be to prepend HMAC type preprocessing, but we have to now change the message size that can be processed by a single block because the SHA-1 hash function appends its own padding and length values. We specify the straightforward EHMAC function using SHA-1 hash function:

$$EHMAC_k(x) = F(\bar{k} \oplus opad, x, pad, 1) \qquad if\ |x|\ <=\ 445\ bits$$

$$= F(\bar{k} \oplus opad, F(\bar{k} \oplus ipad, x_{pref}), x_{suff}, 0) \qquad else$$

where for SHA

$$x_{pref} = x_1 \dots x_{|x|-286}$$

$$x_{suff} = x_{|x|-285} \dots x_{|x|}$$

The key k is appended with enough zeroes to expand k to 512 bit value $\bar{k}$. The 512 bit values of opad and ipad are different from each other and for example are specified in [BCK96]. Since the underlying hash function SHA-1 appends its own 64 bit length value and a mandatory padding bit, the largest possible string that can be fed to F in the SHA-1 case is 447 bits. As specified for the single block case of ENMAC, we reserve the last 447th bit as a single block indicator bit which is set to 1 for the single block case and set to 0 for the multiple block case. Also the single block padding scheme "pad" described in the ENMAC case is used here, hence one more bit is reserved for padding. Thus if the message is less than or equal to 445 bits then we treat it as a single block case. Even for the single block case two calls to the compression are needed, first to process the 512 bit value $\bar{k} \oplus opad$ and then the actual (445 or less) message x. If we look upon $k_1 = f(IV, \bar{k} \oplus opad)$ then the relationship of EHMAC to ENMAC seems more apparent.

For the multiple block case similar HMAC preprocessing is added, but the length of x_{suff} that can be loaded in the outer call has to be reduced to accommodate the length and pad appending of the underlying SHA-1 hash function.

A variant of EHMAC which is slightly more efficient for the single block case is possible where instead of specifying a pad scheme, a "1" is appended to x in the single block case and a "0" is appended in the multiple block case. The security is preserved because SHA-1 hash function itself has an unambiguous method of padding.

6 Conclusion

NMAC and HMAC are efficient for long messages, however, for short messages the nested constructions results in a significant inefficiency. For example to MAC a message shorter than a block, HMAC requires at least two calls to the hash function rather than one. We proposed an enhancement that allows both short and long messages to be message authenticated more efficiently than HMAC while also providing proofs of security. For a message smaller than a block our MAC only requires one call to the hash function. We discuss various variants of our constructions.

7 Acknowledgements

I would like to thank Phil MacKenzie for providing helpful comments on an earlier version of the draft, and Frank Quick for suggesting to include presentation of security proofs (Appendix B) which closely follow the presentation in the HMAC paper.

8 References

[BCK96] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In Proc. CRYPTO 96.

[KBC97] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-hash functions for message authentication, IETF RFC-2104, Feb 1997.

[BKR94] M. Bellare, J. Kilian and P. Rogaway. The security of cipher block chaining. CRYPTO 94

[FIPS180] National Bureau of Standards, FIPS publication 180-1: Secure Hash Standard, 1995. Federal Information Processing Standards Publications 180-1.

[GMR88] O. Goldwasser, S. Micali, and R. Rivest. A digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. Siam Journal of Computing, 1988, pg 281-308.

[PV95] B. Preneel and P.C. van Oorschot. MD-x MAC and building fast MACs from hash functions," CRYPTO 95.

[R92] R. Rivest. The MD5 message digest algorithm. IETF RFC-1321, 1992.

A Lemma 1

Lemma 1 states that the joint probability of the event that $E_{+,pref=}$ happens where A_E forges a multi block message with its prefix, x_{pref}, being equal to prefix of some previously queried message, $x_{i,pref}$, and there is a collision of x_{suff} and a message in set $S_{pref=}$ is less than ϵ_F. Suppose that the probability of $P[(\ \exists j \epsilon S_{pref=}\ s.t.\ (\ F_{k2}(x_{suff}) = F_{k2}(x_{j,suff})\ )\ )\ \cap\ E_{+,pref=}]$ is significant then we can, similar to A_f, create an algorithm A_F which will find collisions on the hash function $F_{k2}()$. This particular way of finding collisions we know is bounded above by ϵ_F, the probability that the best algorithm finds a collision. We now describe the algorithm A_F shown in figure 4.

Instead of choosing a random k_2 as in A_f, we choose a random key k_1 since we are trying to break the collision resistant function F_{k2} and not the mac function f_{k1}. The algorithm A_E will ask to see the ENMAC on various messages x_i. We answer these queries correctly, as was done in A_f, so to A_E it looks like the answers are coming from a true ENMAC function. To calculate ENMAC on x_i we check if its a single block message, if it is then A_F itself calculates $f_{k1}(x_i, pad, 1)$ since it knows the key k_1. If x_i is a multi block message then A_F first makes a query call to the real hash function $F_{k2}()$ with the argument $x_{i,suff}$. A_F takes this answer and uses key k_1 to form the mac of $f_{k1}(x_{i,pref}, F_{k2}(x_{i,suff}), 0)$ and gives to A_E. When A_E gives a forgery (x, y), we see if its a multi block message and whether its x_{pref} equals $x_{i,pref}$ for some previously queried x_i. If it doesn't then A_F says no collision found. If the set $S_{pref=}$ is not empty then we query $F_{k2}(x_{suff})$ and see if it equals $F_{k2}(x_{j,suff})$ for some j in $S_{pref=}$. If it does then we output the pair x_{suff} and $x_{j,suff}$ as collision points and stop.

The probability of A_F finding a collision this way has to be less than the probability of finding collision by the best algorithm which is ϵ_F. But we also see that the probability of A_F finding a collision is the same as the event in A_f of $(\ \exists j \epsilon S_{pref=}\ s.t.\ (\ F(x_{suff}) = F(x_{j,suff})\ )\ )\ \cap\ E_{+,pref=}$. Thus this will also be bounded by ϵ_F.

Choose random k_1
For $i = 1 \dots q$ do
    $A_E \rightarrow x_i$
    If $x_i \leq b - 2$
        $A_E \leftarrow f_{k_1}(x_i, pad, 1)$
    else
        $A_E \leftarrow f_{k_1}(x_{i,pref}, F_{k_2}(x_{i,suff}), 0)$
$A_E \rightarrow (x, y)$
If $(x > b - 2)$ and (x_{pref} equals some $x_{i,pref}$)
  query $F_{k2}(x_{suff})$
  If $F_{k2}(x_{suff}$ equals $F_{k2}(x_{j,suff})$ for some j in $S_{pref=}$
    output x_{suff} and $x_{j,suff}$ as collision points and stop
output no collisions found

Figure 4: A_F algorithm to find collisions in keyed hash $F_{k2}()$

B Alternate Security Proof of ENMAC

In this alternate proof we proceed from the probability of A_f failing to forge a correct mac of $f_{k1}()$ rather than the $P[A_f\ forges]$ as done in the main proof.

$$\begin{aligned} P[A_f\ fails] &= P[A_f\ fails \cap E] + P[A_f\ fails \cap \bar{E}] && (15) \\ &= P[A_f\ fails \cap E] + P[A_f\ fails \mid \bar{E}]\ P[\bar{E}] && (16) \\ &= P[A_f\ fails \cap E] + 1 \cdot (1 - \epsilon_E) && (17) \\ &= P[A_f\ fails \cap E] + 1 - \epsilon_E && (18) \end{aligned}$$

In equation 17, the $P[A_f\ fails \mid \bar{E}]$ is set to 1 because whether in the single block or multi block case, if A_E was unsuccessful in forging a valid ENMAC on x then A_f output would also not be a valid mac.

$$\begin{aligned} P[A_f\ fails] &= P[A_f\ fails \cap E_1] + P[A_f\ fails \cap E_+] + 1 - \epsilon_E && (19) \\ &= P[A_f\ fails \mid E_1]\ P[E_1] + P[A_f\ fails \cap E_+] + 1 - \epsilon_E && (20) \\ &= 0 \cdot P[E_1] + P[A_f\ fails \cap E_+] + 1 - \epsilon_E && (21) \\ &= P[A_f\ fails \cap E_{+,pref\neq}] + P[A_f\ fails \cap E_{+,pref=}] + 1 - \epsilon_E && (22) \\ &= P[A_f\ fails \mid E_{+,pref\neq}]\ P[E_{+,pref\neq}] + P[A_f\ fails \cap E_{+,pref=}] + 1 - \epsilon_E && (23) \\ &= 0 \cdot P[E_{+,pref\neq}] + P[A_f\ fails \cap E_{+,pref=}] + 1 - \epsilon_E && (24) \\ &= P[A_f\ fails \cap E_{+,pref=}] + 1 - \epsilon_E && (25) \end{aligned}$$

In equation 20, $P[A_f\ fails \mid E_1]$ is zero because if A_E has forged a correct $(x, y = ENMAC(x))$ pair then A_f would have forged a correct $([x, pad, 1], f_{k1}(x, pad, 1))$ pair. Similarly in equation 23 the $P[A_f\ fails \mid E_{+,pref\neq}]$ is zero because if a correct pair was forged with a prefix different then any previously queried then this prefix will be a new input to the message/tag pair output by A_f which is $(x_{pref}, F_{k2}(x_{suff}), 0)$ as message and $f_{k1}(x_{pref}, F_{k2}(x_{suff}), 0)$ as tag.

$$
\begin{aligned}
P[A_f\ fails] &= P[(\ \exists j \epsilon S_{pref=}\ s.t.\ (\ F_{k2}(x_{suff}) = F_{k2}(x_{j,suff})\ )\ )\ \cap\ E_{+,pref=}] + 1 - \epsilon_E \quad (26)\\
P[A_f\ fails] &< \epsilon_F + 1 - \epsilon_E \quad (27)\\
1 - \epsilon_f &< P[best\ (t,q)\ algorithm\ fails\ in\ forging\ mac\ f_{k1}()]\\
&\leq\ P[A_f\ fails] < \epsilon_F + 1 - \epsilon_E \quad (28)\\
1 - \epsilon_f &< \epsilon_F + 1 - \epsilon_E \quad (29)\\
\epsilon_E &< \epsilon_f + \epsilon_F \quad (30)
\end{aligned}
$$

Equation 26 equates the $P[A_f\ fails \cap E_{+,pref=}]$ in equation 25 with the probability of finding a collision in $F_{k2}(x_{suff})$ and another $F_{k2}(x_{j,suff})$ where x_j is from the set $S_{pref=}$. Lemma 1 deals exactly with this probability and we use Lemma 1 to bound this collision probability in equation 27. Equation 28 restates the fact that even the best algorithm will fail to forge with probability greater than $1 - \epsilon_f$. And in turn the particular algorithm A_f will fail to forge with probability greater than or equal to that. In equation 30 we complete the alternate proof of theorem 1.