Appears in *Advances in Cryptology – Eurocrypt '02*, Lecture Notes in Computer Science, Vol. ????, L. Knudsen, ed., Springer-Verlag. This is the full version.

A Block-Cipher Mode of Operation for Parallelizable Message Authentication

J. Black [*] P. Rogaway [†]

February 15, 2002

Abstract

We define and analyze a simple and fully parallelizable block-cipher mode of operation for message authentication. Parallelizability does not come at the expense of serial efficiency: in a conventional, serial environment, the algorithm's speed is within a few percent of the (inherently sequential) CBC MAC. The new mode, PMAC, is deterministic, resembles a standard mode of operation (and not a Carter-Wegman MAC), works for strings of any bit length, employs a single block-cipher key, and uses just $\max\{1, \lceil |M|/n \rceil\}$ block-cipher calls to MAC a string $M \in \{0,1\}^*$ using an n-bit block cipher. We prove PMAC secure, quantifying an adversary's forgery probability in terms of the quality of the block cipher as a pseudorandom permutation.

Key words: block-cipher modes, message authentication codes, modes of operation, provable security.

1 Introduction

Background. Many popular message authentication codes (MACs), like the CBC MAC [17] and HMAC [1], are inherently sequential: one cannot process the i-th message block until all previous message blocks have been processed. This serial bottleneck becomes increasingly an issue as commodity processors offer up more and more parallelism, and as increases in network speeds outpace increases in the speed of cryptographic hardware. By now there would seem to be a significant interest in having a parallelizable MAC which performs well in both hardware and software, built from a block cipher like AES.

There are several approaches to the design of such an MAC. One is to generically construct a more parallelizable MAC from an arbitrary one. For example, one could begin with breaking the message $M[1] \cdots M[2m]$ into $M' = M[1]M[3] \cdots M[2m-1]$ and $M'' = M[2]M[4] \cdots M[2m]$ then separately MAC each half. But such an approach requires one to anticipate the maximal amount of parallelism one aims to extract. In the current work we are instead interested in *fully parallelizable* MACs: the amount of parallelism that can be extracted is effectively unbounded.

One idea for making a fully parallelizable MAC is to use the Carter-Wegman paradigm [13, 23], as in [12, 16, 19], making sure to select a universal hash-function family that is fully parallelizable. In fact, most universal hash functions that have been suggested *are* fully parallelizable. This approach is elegant and can lead to a nice MAC, but constructions for fast universal hash-functions have

[*] Department of Computer Science/171, University of Nevada, Reno, Nevada, 89557, USA. E-mail: `jrb@cs.unr.edu` WWW: `www.cs.unr.edu/~jrb/`

[†] Department of Computer Science, University of California, Davis, California, 95616, USA. E-mail: `rogaway@cs.ucdavis.edu` WWW: `www.cs.ucdavis.edu/~rogaway/`

proven to be quite complex to specify or to implement well [7, 9], and may be biased either towards hardware or towards software. Twenty years after the paradigm was introduced, we still do not know of a single Carter-Wegman MAC that actually gets used. So the current work goes back to giving a conventional mode, but one designed, this time around, for serial *and* parallel efficiency.

The XOR MAC. Bellare, Guérin and Rogaway introduced a parallelizable MAC with their XOR MACs [3]. The message M is divided into pieces $M[1] \cdots M[\ell]$ of length less than the blocksize; for concreteness, think of each $M[i]$ as having 64 bits when the blocksize is $n = 128$ bits. Each piece $M[i]$ is preceded by $[i]$, the number i encoded as a 64-bit number, and to each $[i] \parallel M[i]$ one applies the block cipher E, keyed by the MAC key K. One more block is enciphered, it having a first bit of 1 and then a counter or random value in the remaining $n-1$ bits. The MAC is that counter or random value together with the XOR of all $\ell+1$ ciphertext blocks. Thus the MAC uses $\ell + 1 \approx 2m + 1$ block-cipher invocations to authenticate a message of m blocks of n bits; one has paid for parallelizability at a cost of about a factor of two in serial speed. Further disadvantages include the need for randomness or state (conventional MACs are deterministic) and in the increased length of the MAC (because of the counter or random value that has to be included).

PMAC. Unlike the XOR MAC, our new algorithm, PMAC, doesn't waste any block-cipher invocations because of block-indices (nor for a counter or random values). Also, in the spirit of [11], we optimally deal with short final blocks; we correctly MAC messages of arbitrary and varying bit lengths. The result is that PMAC makes do with just $\lceil |M|/n \rceil$ block-cipher calls to MAC a non-empty message M using an n-bit block cipher. PMAC is deterministic, freeing the user from having to provide a counter or random value, and making the MAC shorter. Overhead beyond the block-cipher calls has been aggressively optimized, so a serial implementation of PMAC runs just a few percent slower than the CBC MAC.

Besides the efficiency measures already mentioned, PMAC uses very little key-setup: one block-cipher call. (A few shifts and conditional xors are also used.) The PMAC key is a single key for the underlying blocks cipher; in particular, we forgo the need for key-separation techniques. Avoiding multiple block-cipher keys saves time because many block ciphers have significant key-setup costs.

Being so stingy with keys and block-cipher invocations takes significant care; note that even the traditional CBC MAC uses between one and four additional block-cipher calls, as well as additional key material, once it has been enriched to take care of messages of arbitrary lengths [6, 11, 17, 21]. Of course avoiding this overhead doesn't matter much on long messages, but it is significant on short ones. And in many environments, short messages are common.

We prove PMAC secure, in the sense of reduction-based cryptography. Specifically, we prove that PMAC approximates a random function (and is therefore a good MAC) as long as the underlying block cipher approximates a random permutation. The actual results are quantitative; the security analysis is in the concrete-security paradigm.

PMAC was proposed in response to NIST's call for contributions for a first modes-of-operation workshop. Earlier versions of this writeup were submitted to NIST and posted to their website (Oct 2000, Apr 2001).

Additional related work. Building on [3], Gligor and Donescu describe a MAC they call the XECB MAC [14]. That MAC is not deterministic, it uses more block-cipher invocations, and it was not designed for messages of arbitrary bit length. But, like PMAC, it goes beyond the XOR MAC by combining a message index and a message block in a way other than encoding the two. In particular, [14] combines i and $M[i]$ by adding to $M[i]$, modulo 2^n, a secret multiple i. We combine i and $M[i]$ by different means, to reduce overhead and obtain a better bound.

PMAC was also influenced by the variant of the XOR MAC due to Bernstein [8]. His algorithm is deterministic, and the way that the XOR MAC was made deterministic in [8] is similar to the way that PMAC has been made deterministic. Finally, there is also some similarity in appearance between PMAC and Jutla's IAPM encryption mode [18].

2 Mathematical Preliminaries

Notation. If $i \geq 1$ is an integer then $\mathsf{ntz}(i)$ is the number of trailing 0-bits in the binary representation of i. So, for example, $\mathsf{ntz}(7) = 0$ and $\mathsf{ntz}(8) = 3$. If $A \in \{0,1\}^*$ is a string then $|A|$ denotes its length in bits while $\|A\|_n = \max\{1, \lceil |A|/n \rceil\}$ denotes its length in n-bit blocks (where the empty string counts as one block). If $A = a_{n-1} \cdots a_1 a_0 \in \{0,1\}^n$ is a string (each $a_i \in \{0,1\}$) then $\mathsf{str2num}(A)$ is the number $\sum_{i=0}^{n-1} 2^i a_i$. If $A, B \in \{0,1\}^*$ are equal-length strings than $A \oplus B$ is their bitwise xor. If $A \in \{0,1\}^*$ and $|A| < n$ then $\mathsf{pad}_n(A)$ is the string $A\, 10^{n-|A|-1}$. If $A \in \{0,1\}^n$ then $\mathsf{pad}_n(A) = A$. With n understood we write $\mathsf{pad}(A)$ for $\mathsf{pad}_n(A)$. If $A = a_{n-1}a_{n-2} \cdots a_1 a_0 \in \{0,1\}^n$ then $A{\ll}1 = a_{n-2}a_{n-3} \cdots a_1 a_0 0$ is the n-bit string which is the left shift of A by 1 bit while $A{\gg}1 = 0a_{n-1}a_{n-2} \ldots a_2 a_1$ is the n-bit string which is the right shift of A by one bit. In pseudocode we write "Partition M into $M[1] \cdots M[m]$" as shorthand for "Let $m = \|M\|_n$ and let $M[1], \ldots, M[m]$ be strings such that $M[1] \cdots M[m] = M$ and $|M[i]| = n$ for $1 \leq i < m$."

The field with 2^n points. The field with 2^n points is denoted $\mathrm{GF}(2^n)$. We interchangeably think of a point a in $\mathrm{GF}(2^n)$ in any of the following ways: (1) as an abstract point in the field; (2) as an n-bit string $a_{n-1} \ldots a_1 a_0 \in \{0,1\}^n$; (3) as a formal polynomial $a(\mathtt{x}) = a_{n-1}\mathtt{x}^{n-1} + \cdots + a_1\mathtt{x} + a_0$ with binary coefficients; (4) as a nonnegative integer between 0 and $2^n - 1$, where $a \in \{0,1\}^n$ corresponds to $\mathsf{str2num}(a)$. We write $a(\mathtt{x})$ instead of a if we wish to emphasize that we are thinking of a as a polynomial. To add two points in $\mathrm{GF}(2^n)$, take their bitwise xor. We denote this operation by $a \oplus b$. To multiply two points, fix some irreducible polynomial $p(\mathtt{x})$ having binary coefficients and degree n. To be concrete, choose the lexicographically first polynomial among the irreducible degree n polynomials having a minimum number of coefficients. To multiply points $a, b \in \mathrm{GF}(2^n)$, which we denote $a \cdot b$, regard a and b as polynomials $a(\mathtt{x}) = a_{n-1}\mathtt{x}^{n-1} + \cdots + a_1\mathtt{x} + a_0$ and $b(\mathtt{x}) = b_{n-1}\mathtt{x}^{n-1} + \cdots + b_1\mathtt{x} + b_0$, form their product $c(\mathtt{x})$ where one adds and multiplies coefficients in $\mathrm{GF}(2)$, and take the remainder when dividing $c(\mathtt{x})$ by $p(\mathtt{x})$. Note that it is particularly easy to multiply a point $a \in \{0,1\}^n$ by $\mathtt{x}$. We illustrate the method for $n = 128$, where $p(\mathtt{x}) = \mathtt{x}^{128} + \mathtt{x}^7 + \mathtt{x}^2 + \mathtt{x} + 1$. Then multiplying $a = a_{n-1} \cdots a_1 a_0$ by $\mathtt{x}$ yields

$$a \cdot \mathtt{x} = \begin{cases} a{\ll}1 & \text{if } \mathsf{firstbit}(a) = 0 \\ (a{\ll}1) \oplus 0^{120}10000111 & \text{if } \mathsf{firstbit}(a) = 1 \end{cases} \tag{1}$$

It is similarly easy to divide a by $\mathtt{x}$ (meaning to multiply a by the multiplicative inverse of $\mathtt{x}$). To illustrate, assume that $n = 128$. Then

$$a \cdot \mathtt{x}^{-1} \;=\; \begin{cases} a{\gg}1 & \text{if } \mathsf{lastbit}(a) = 0 \\ (a{\gg}1) \oplus 10^{120}1000011 & \text{if } \mathsf{lastbit}(a) = 1 \end{cases} \tag{2}$$

If $L \in \{0,1\}^n$ and $i \geq -1$, we write $L(i)$ to mean $L \cdot \mathtt{x}^i$. To compute $L(-1)$, $L(0)$, $\ldots$, $L(\mu)$, where μ is small, set $L(0) = L$ and then, for $i \in [1..\mu]$, use Equation (1) to compute $L(i) = L(i-1) \cdot \mathtt{x}$ from $L(i-1)$; and use Equation (2) to compute $L(-1)$ from L.

We point out that *huge* $= \mathtt{x}^{-1}$ will be an enormous number (when viewed as a number); in particular, *huge* starts with a 1 bit, so *huge* $> 2^{n-1}$. In the security proof this fact is relevant, so there we use *huge* as a synonym for $\mathtt{x}^{-1}$ when this seems to add to clarity.

```
Algorithm PMAC_K (M)
1.   $L \leftarrow E_K(0^n)$
2.   if $|M| > n2^n$ then return $0^\tau$
3.   Partition $M$ into $M[1]\cdots M[m]$
4.   for $i \leftarrow 1$ to $m-1$ do
5.       $X[i] \leftarrow M[i] \oplus \gamma_i \cdot L$
6.       $Y[i] \leftarrow E_K(X[i])$
7.   $\Sigma \leftarrow Y[1] \oplus Y[2] \oplus \cdots \oplus Y[m-1] \oplus \mathsf{pad}(M[m])$
8.   if $|M[m]| = n$ then $X[m] = \Sigma \oplus L \cdot \mathtt{x}^{-1}$
9.                   else $X[m] \leftarrow \Sigma$
10.  $Tag = E_K(X[m])$ [first $\tau$ bits]
11.  return $Tag$
```

Figure 1: **Definition of PMAC**. The message to MAC is M and the key is K. The algorithm depends on a block cipher E: $\mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ and a number $\tau \in [1..n]$. Constants $\gamma_1, \gamma_2, \ldots$, the meaning of the multiplication operator, and the meaning of $\mathsf{pad}()$ are all defined in the text.

Gray codes. For any $\ell \geq 1$, a Gray code is an ordering $\gamma^\ell = \gamma_0^\ell \ \gamma_1^\ell \ \ldots \ \gamma_{2^\ell-1}^\ell$ of $\{0,1\}^\ell$ such that successive points differ (in the Hamming sense) by just one bit. For n a fixed number, PMAC makes use of the "canonical" Gray code $\gamma = \gamma^n$ constructed by $\gamma^1 = 0 \ 1$ while, for $\ell > 0$,

$$\gamma^{\ell+1} = 0\gamma_0^\ell \ \ 0\gamma_1^\ell \ \ \cdots \ \ 0\gamma_{2^\ell-2}^\ell \ \ 0\gamma_{2^\ell-1}^\ell \ \ 1\gamma_{2^\ell-1}^\ell \ \ 1\gamma_{2^\ell-2}^\ell \ \ \cdots \ \ 1\gamma_1^\ell \ \ 1\gamma_0^\ell.$$

It is easy to see that γ is a Gray code. What is more, for $1 \leq i \leq 2^n - 1$, $\gamma_i = \gamma_{i-1} \oplus (0^{n-1}1 \lll \mathsf{ntz}(i))$. This makes it easy to compute successive points. Note that $\gamma_1, \gamma_2, \ldots, \gamma_{2^n-1}$ are distinct, different from 0, and $\gamma_i \leq 2i$.

Let $L \in \{0,1\}^n$ and consider the problem of successively forming the strings $\gamma_1 \cdot L$, $\gamma_2 \cdot L$, $\gamma_3 \cdot L$, …, $\gamma_m \cdot L$. Of course $\gamma_1 \cdot L = 1 \cdot L = L$. Now, for $i \geq 2$, assume one has already produced $\gamma_{i-1} \cdot L$. Since $\gamma_i = \gamma_{i-1} \oplus (0^{n-1}1 \lll \mathsf{ntz}(i))$ we know that $\gamma_i \cdot L = (\gamma_{i-1} \oplus (0^{n-1}1 \lll \mathsf{ntz}(i))) \cdot L = (\gamma_{i-1} \cdot L) \oplus (0^{n-1}1 \lll \mathsf{ntz}(i)) \cdot L = (\gamma_{i-1} \cdot L) \oplus (L \cdot \mathtt{x}^{\mathsf{ntz}(i)}) = (\gamma_{i-1} \cdot L) \oplus L(\mathsf{ntz}(i))$. That is, the ith word in the sequence $\gamma_1 \cdot L, \gamma_2 \cdot L, \gamma_3 \cdot L, \ldots$ is obtained by xoring the previous word with $L(\mathsf{ntz}(i))$.

3 Definition of PMAC

PMAC depends on two parameters: a block cipher and a tag length. The block cipher is a function E: $\mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$, for some number n, where each $E(K, \cdot) = E_K(\cdot)$ is a permutation on $\{0,1\}^n$. Here $\mathcal{K}$ is the set of possible keys and n is the block length. The tag length is an integer $\tau \in [1..n]$. By trivial means the adversary will be able to forge a valid ciphertext with probability $2^{-\tau}$. With E: $\mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ and $\tau \in [1..n]$, we let PMAC$[E, \tau]$ denote PMAC using block cipher E and tag length τ. We simplify to PMAC-E when τ is irrelevant. PMAC$[E, \tau]$ is a function taking a key $K \in \mathcal{K}$ and a message $M \in \{0,1\}^*$ and returning a string in $\{0,1\}^\tau$. The function is defined in Figure 1 and illustrated in Figure 2. We comment that line 2 of Figure 1 is simply to ensure that PMAC is well-defined even for the highly unrealistic case that $|M| > n2^n$ (by which time our security result becomes vacuous anyway). Alternatively, one may consider PMAC's message space to be strings of length at most $n2^n$ rather than strings of arbitrary length.

The following alternative description of PMAC may help to clarify what a typical implementation might choose to do. **Key generation:** Choose a random key $K \overset{R}{\leftarrow} \mathcal{K}$ for the block cipher. The

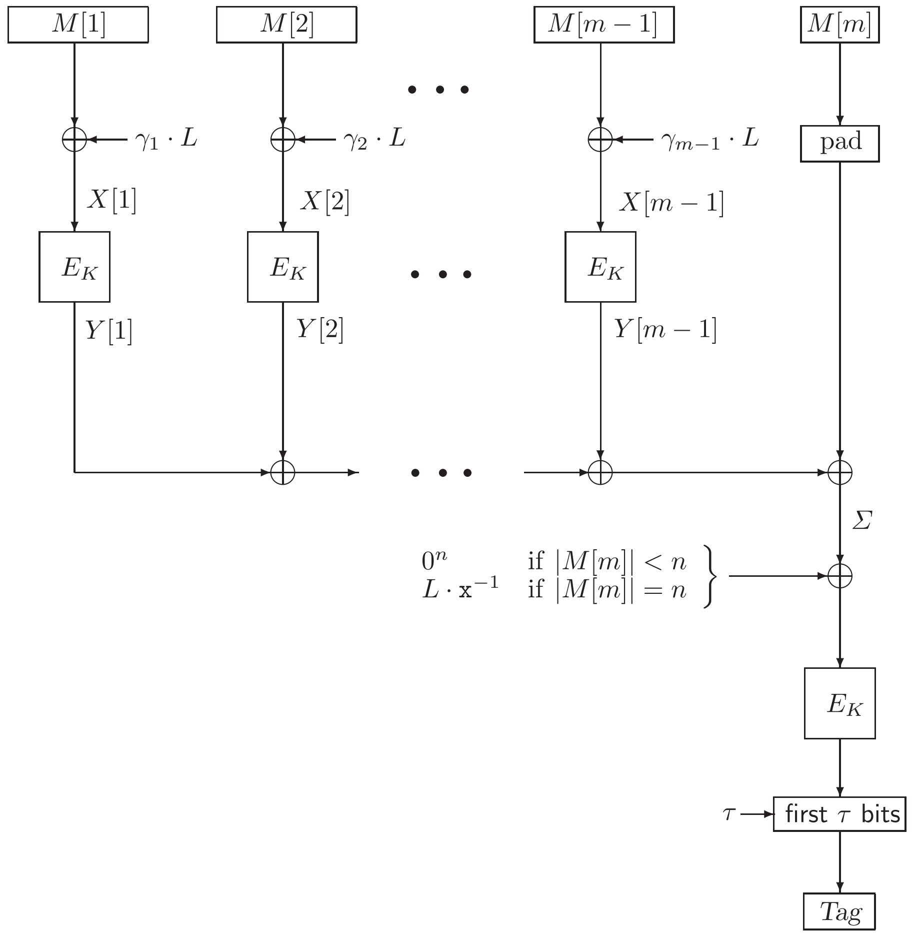

Figure 2: **Illustration of PMAC**. Message M is written as $M = M[1] \cdots M[m]$, where $m = \max\{1, \lceil |M|/n \rceil\}$ and $|M[1]| = |M[2]| = \cdots = |M[m-1]| = n$. Value $L = E_K(0^n)$ is derived from K. The meaning of the $\gamma_i \cdot L$ values is described in the text.

key K is provided to both the party that generates MACs and the party that verifies them. **Key setup:** Both the party that generates the MACs and the party that verifies the MACs do any key setup useful for applying the block-cipher in its forward direction. Let $L \leftarrow E_K(0^n)$. Let m^* bound the maximum number of n-bit blocks for any message which will be MACed. Let $\mu \leftarrow \lceil \log_2 m^* \rceil$. Let $L(0) \leftarrow L$. For $i \in [1..\mu]$, compute $L(i) \leftarrow L(i-1) \cdot \mathtt{x}$ by Equation (1), using a shift and a conditional xor. Compute $L(-1) \leftarrow L \cdot \mathtt{x}^{-1}$ by Equation (2), using a shift and a conditional xor. Save the values $L(-1)$, $L(0)$, $L(1)$, $L(2)$, $\ldots$, $L(\mu)$ in a table. (Alternatively, defer computing some or all of these $L(i)$ values until the value is actually needed.) **MAC generation:** Let $m \leftarrow \lceil |M|/n \rceil$. If $m = 0$ then let $m \leftarrow 1$. Let $M[1], \ldots, M[m]$ be strings such that $M[1] \cdots M[m] = M$ and $|M[i]| = n$ for $i \in [1..m-1]$. Let $\mathit{Offset} \leftarrow 0^n$. Let $\Sigma \leftarrow 0^n$. For $i \leftarrow 1$ to $m-1$, do the following: let $\mathit{Offset} \leftarrow \mathit{Offset} \oplus L(\mathsf{ntz}(i))$; let $Y[i] \leftarrow E_K(M[i] \oplus \mathit{Offset})$; let $\Sigma \leftarrow \Sigma \oplus Y[i]$. Let $\Sigma \leftarrow \Sigma \oplus \mathsf{pad}(M[m])$. If $|M[m]| < n$ then let $Y[m] \leftarrow E_K(\Sigma)$ else let $Y[m] \leftarrow E_K(\Sigma \oplus L(-1))$. Let *Tag* be the first τ bits of $Y[m]$. Return *Tag* as the computed MAC. **MAC verification:** Given (M, Tag'), do the following: Generate the MAC *Tag* for the message M using the MAC generation procedure just described. If $\mathit{Tag} = \mathit{Tag}'$ then regard the message M as authentic. If $\mathit{Tag} \neq \mathit{Tag}'$ then regard the message M as inauthentic.

4 Comments

As we shall soon prove, PMAC is more than a good MAC: it is good as a pseudorandom function (PRF) having variable-input-length and fixed-output-length. As long as the underlying block cipher E is secure, no reasonable adversary will be able to distinguish $\text{PMAC}_K(\cdot)$, for a random and hidden key K, from a random function ρ from $\{0,1\}^*$ to $\{0,1\}^\tau$. It is a well-known observation, dating to the introduction of PRFs [15], that a good PRF is necessarily a good MAC.

Conceptually, the key is (K, L). But instead of regarding this as the key (and possibly defining K and L from an underlying key), the value L is defined from K and then K is still used as a key. Normally such "lazy key-derivation" would get one into trouble, in proofs if nothing else. For PMAC we prove that this form of lazy key-derivation works fine.

Any string $M \in \{0,1\}^*$ can be MACed, and messages which are not a multiple of the block length are handled without the need for obligatory padding, which would increase the number of block-cipher calls.

MAC generation is "on line," meaning that one does not need to know the length of the message M in advance. Instead, the message can be MACed as one goes along, continuing until there is an indication that the message is now complete. The work of Petrank and Rackoff brought out the importance of this property [21].

In contrast to a scheme based on mod p arithmetic (for a prime p) or mod 2^n arithmetic, there is almost no endian-favoritism implicit in the definition of PMAC. (The exception is that the left shift used for forming $L(i+1)$ from $L(i)$ is more convenient under a big-endian convention, as is the right shift used for forming $L(-1) = L \cdot \mathtt{x}^{-1}$ from L.)

If $\tau = n$ (or one retains a constant amount of extra information) PMAC is incremental in the sense of [15] with respect to operations

$$\begin{aligned}
\mathtt{append}(M, x) &= M \,\|\, x, \\
\mathtt{truncate}(M, \Delta) &= M\,[\text{first } |M| - \Delta \text{ bits}], \text{for } |M| \geq \Delta, \text{and} \\
\mathtt{replace}(M, i, x) &= M\,[\text{first } i-1 \text{ bits}] \,\|\, x \,\|\, M[\text{last } |M| - i - |x| + 1 \text{ bits}], \text{for } |M| \geq i + |x| - 1.
\end{aligned}$$

For each operation it is easy to see how to update the MAC of M in time proportional to $|x|$, Δ, or $|x|$, respectively.

PMAC is parsimonious, as defined in [5]. Partition M into $M[1] \cdots M[m]$ and assume $|M| \geq n$ and $\tau = n$. For $i \in [1..m]$ such that $|M[i]| = n$, there is a simple algorithm to recover $M[i]$ from K, $M' = M[1] \cdots M[i-1]\; M[i+1] \cdots M[m]$, and $\mathit{Tag} = \text{PMAC}_K(M)$. As shown in [5], a parsimonious PRF can be combined with a parsimonious encryption scheme (eg., CTR mode) to yield a length-preserving pseudorandom permutation (a "variable-input-length block cipher") that acts on messages of any number of bits greater than or equal to n.

5 Theorems

Security definitions. We first recall the needed definitions. A block cipher is a function $E\colon \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ where $\mathcal{K}$ is a finite set and each $E_K(\cdot) = E(K, \cdot)$ is a permutation on $\{0,1\}^n$. Let $\text{Perm}(n)$ denote the set of all permutations on $\{0,1\}^n$. This set can be regarded as a block cipher by imagining that each permutation is named by a unique element of $\mathcal{K}$. Let A be an adversary (a probabilistic algorithm) with access to an oracle, and suppose that A always outputs a bit. Define

$$\mathbf{Adv}^{\text{prp}}_E(A) \;=\; \Pr[K \overset{R}{\leftarrow} \mathcal{K}:\; A^{E_K(\cdot)} = 1] - \Pr[\pi \overset{R}{\leftarrow} \text{Perm}(n):\; A^{\pi(\cdot)} = 1]$$

The above is the probability that adversary A outputs 1 when given an oracle for $E_K(\cdot)$, minus the probability that A outputs 1 when given an oracle for $\pi(\cdot)$, where K is selected at random from $\mathcal{K}$ and π is selected at random from $\mathrm{Perm}(n)$. Similarly, a function family from n-bits to n-bits is a map $F\colon \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ where $\mathcal{K}$ is a finite set. We write $F_K(\cdot)$ for $F(K,\cdot)$. Let $\mathrm{Rand}(n)$ denote the set of all functions from $\{0,1\}^n$ to $\{0,1\}^n$. This set can be regarded as a function family as above. Define

$$\mathbf{Adv}^{\mathrm{prf}}_F(A) \quad = \quad \Pr[K \overset{R}{\leftarrow} \mathcal{K} :\ A^{F_K(\cdot)} = 1] - \Pr[\rho \overset{R}{\leftarrow} \mathrm{Rand}(n) :\ A^{\rho(\cdot)} = 1]$$

Finally, a function family from $\{0,1\}^*$ to $\{0,1\}^\tau$ is a map $f\colon \mathcal{K} \times \{0,1\}^* \to \{0,1\}^\tau$ where $\mathcal{K}$ is a set with an associated distribution. We write $f_K(\cdot)$ for $f(K,\cdot)$. Let $\mathrm{Rand}(*,\tau)$ denote the set of all functions from $\{0,1\}^*$ to $\{0,1\}^\tau$. This set is given a probability measure by asserting that a random element ρ of $\mathrm{Rand}(*,\tau)$ associates to each string $x \in \{0,1\}^*$ a random string $\rho(x) \in \{0,1\}^\tau$. Define

$$\mathbf{Adv}^{\mathrm{prf}}_f(A) \quad = \quad \Pr[K \overset{R}{\leftarrow} \mathcal{K} :\ A^{f_K(\cdot)} = 1] - \Pr[g \overset{R}{\leftarrow} \mathrm{Rand}(*,\tau) :\ A^{g(\cdot)} = 1]$$

Main result. We now give an information-theoretic bound on the security of our construction.

Theorem 1 [Security of PMAC] *Fix $n, \tau \geq 1$. Let A be an adversary with an oracle. Suppose that A asks its oracle q queries, these having aggregate length of σ blocks. Let $\bar{\sigma} = \sigma + 1$. Then*

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathrm{PMAC}[\mathrm{Perm}(n),\tau]}(A) \quad \leq \quad \frac{\bar{\sigma}^2}{2^{n-1}}$$

In the theorem statement and from now on, the aggregate length of messages $M_1, \ldots, M_q$ asked by A is the number $\sigma = \sum_{r=1}^{q} \|M_r\|_n$.

From the theorem above it is standard to pass to a complexity-theoretic analog. Fix parameters $n, \tau \geq 1$ and block cipher $E\colon \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$. Let A be an adversary with an oracle and suppose that A asks queries having aggregate length of σ blocks. Let $\bar{\sigma} = \sigma + 1$. Then there is an adversary B for attacking E that achieves advantage $\mathbf{Adv}^{\mathrm{prp}}_E(B) \geq \mathbf{Adv}^{\mathrm{prf}}_{\mathrm{PMAC}[E,\tau]}(A) - \bar{\sigma}^2/2^{n-1}$. Adversary B asks at most $\bar{\sigma}$ oracle queries and has a running time equal to A's running time plus the time to compute E on $\bar{\sigma}$ points, plus additional time of $cn\bar{\sigma}$ for a constant c that depends only on details of the model of computation.

It is a standard result that being secure in the sense of a PRF implies an inability to forge with good probability. See [4, 15].

Structure of the proof. The proof combines two lemmas. The first, the *structure lemma*, measures the pseudorandomness of PMAC using two functions: the M-collision probability, denoted $\mathrm{Mcoll}_n(\cdot)$, and the MM-collision probability, denoted $\mathrm{MMcoll}_n(\cdot,\cdot)$. The second lemma, the *collision-bounding lemma*, upperbounds $\mathrm{Mcoll}_n(m)$ and $\mathrm{MMcoll}_n(m, \bar{m})$.

We begin by defining $\mathrm{Mcoll}_n(\cdot)$ and $\mathrm{MMcoll}_n(\cdot,\cdot)$. Fix n and choose M and $\bar{M}$, partitioning them into $M[1]\cdots M[m]$ and $\bar{M}[1]\cdots\bar{M}[\bar{m}]$. Consider the experiment of Figure 3. When M is a string, let $\mathrm{Mcoll}_n(M)$ denote the probability that *Mcoll* gets set to `true` in line 15 when the program of Figure 3 is run on M. When m is a number, $\mathrm{Mcoll}_n(m)$ is the maximum value of $\mathrm{Mcoll}_n(M)$ over all strings M such that $\|M\|_n = m$. Similarly, when M and $\bar{M}$ are strings, let $\mathrm{MMcoll}_n(M, \bar{M})$ denote the probability that *MMcoll* gets set to `true` when the program of Figure 3 is run on strings $M, \bar{M}$. When m and $\bar{m}$ are numbers, let $\mathrm{MMcoll}_n(m, \bar{m})$ denote the maximum value of $\mathrm{MMcoll}_n(M, \bar{M})$ over all strings $M, \bar{M}$ such that $\|M\|_n = m$ and $\|\bar{M}\|_n = \bar{m}$. We can now state the structure lemma.

```
10  $L \xleftarrow{R} \{0,1\}^n$
11  **for** $i \leftarrow 1$ **to** $m-1$ **do** $\{\ X[i] \leftarrow M[i] \oplus \gamma_i \cdot L;\quad Y[i] \xleftarrow{R} \{0,1\}^n\ \}$
12  $\Sigma \leftarrow Y[1] \oplus \cdots \oplus Y[m-1] \oplus \mathsf{pad}(M[m])$
13  **if** $|M[m]| = n$ **then** $X[m] \leftarrow \Sigma \oplus huge \cdot L$ **else** $X[m] \leftarrow \Sigma$
14  $\mathcal{X} \leftarrow \{X[1], \ldots, X[m]\}$
15  **if** there is a repetition in $\{0^n\} \cup \mathcal{X}$ **then** $Mcoll \leftarrow$ true

16  $Equal \leftarrow \{i \in [1..\min\{m, \bar{m}\}-1] :\ M[i] = \bar{M}[i]\}$
17  $Unequal \leftarrow [1..\bar{m}] \setminus Equal$
18  **for** $i \leftarrow 1$ **to** $\bar{m}-1$ **do**
19      **if** $i \in Equal$ **then** $\{\ \bar{X}[i] \leftarrow X[i];\quad \bar{Y}[i] \leftarrow Y[i]\ \}$
20      **if** $i \in Unequal$ **then** $\{\ \bar{X}[i] \leftarrow \bar{M}[i] \oplus \gamma_i \cdot L;\quad \bar{Y}[i] \xleftarrow{R} \{0,1\}^n\ \}$
21  $\bar{\Sigma} \leftarrow \bar{Y}[1] \oplus \cdots \oplus \bar{Y}[\bar{m}-1] \oplus \mathsf{pad}(\bar{M}[\bar{m}])$
22  **if** $|\bar{M}[\bar{m}]| = n$ **then** $\bar{X}[\bar{m}] \leftarrow \bar{\Sigma} \oplus huge \cdot L$ **else** $\bar{X}[\bar{m}] \leftarrow \bar{\Sigma}$
23  $\bar{\mathcal{X}} \leftarrow \{\bar{X}[i] : i \in Unequal\}$
24  **if** $\mathcal{X} \cap \bar{\mathcal{X}} \neq \emptyset$ **then** $MMcoll \leftarrow$ true
```

Figure 3: **Defining the collision probabilities.** Functions $\mathrm{Mcoll}_n(\cdot)$ and $\mathrm{MMcoll}_n(\cdot,\cdot)$ are defined using this game. In lines 14 and 23, $\mathcal{X}$ and $\bar{\mathcal{X}}$ are understood to be multisets. The union in line 15 is a multiset union. Recall that *huge* is a synonym for $\mathtt{x}^{-1}$.

Lemma 1 [Structure lemma] *Fix $n, \tau \geq 1$. Let A be an adversary that asks q queries, these having an aggregate length of σ blocks. Then*

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathrm{PMAC}[\mathrm{Perm}(n),\tau]}(A) \leq$$
$$\max_{\substack{m_1,\ldots,m_q \\ \sigma=\sum m_i \\ m_i \geq 1}} \left\{ \sum_{1\leq r\leq q} \mathrm{Mcoll}_n(m_r) + \sum_{1\leq r<s\leq q} \mathrm{MMcoll}_n(m_r, m_s) \right\} + \frac{(\sigma+1)^2}{2^{n+1}}$$

The proof of this lemma is in Appendix A.

Explanation. Informally, $\mathrm{Mcoll}_n(m)$ measures the probability of running into trouble when the adversary asks a single question M having block length m. Trouble means a collision among the values $X[0], X[1], \ldots, X[m]$, where $X[0] = 0^n$ and each $X[i]$ is the block-cipher input associated to message block i. Informally, $\mathrm{MMcoll}_n(m, \bar{m})$ measures the probability of running into trouble across two messages, M and $\bar{M}$, having lengths m and $\bar{m}$. This time trouble means a "non-trivial" collision. That is, consider the $m+\bar{m}+1$ points at which the block cipher is applied in processing M and $\bar{M}$. There are m points $X[1], \ldots, X[m]$, another $\bar{m}$ points $\bar{X}[1], \ldots, \bar{X}[\bar{m}]$, and then there is the point 0^n (the block cipher was applied at this point to define L). Some pairs of these $m+\bar{m}+1$ points could coincide for a "trivial" reason: namely, we know that $X[i] = \bar{X}[i]$ if $i < m$ and $i < \bar{m}$ and $M[i] = \bar{M}[i]$. We say that there is a nontrivial collision between M and $\bar{M}$ if some *other* $X[i]$ and $\bar{X}[j]$ happened to coincide. Note that M-collisions include collisions with 0^n, while MM-collisions do not. Also, MM-collisions do not include collisions within a single message (or collisions with 0^n) because both of these possibilities are taken care of by way of M-collisions.

The structure lemma provides a simple recipe for measuring the maximum advantage of any adversary that attacks the pseudorandomness of PMAC: bound the collision probabilities $\mathrm{Mcoll}_n(\cdot)$ and $\mathrm{MMcoll}_n(\cdot,\cdot)$ and then use the formula. The lemma simplifies the analysis of PMAC in two ways. First, it allows one to excise adaptivity as a concern. Dealing with adaptivity is a major complicating factor in proofs of this type. Second, it allows one to concentrate on what happens to single messages and to a fixed pair of messages. It is easier to think about what happens with one or two messages than what is happening with all q of them.

Bounding the collision probabilities. The following lemma indicates that the two types of collisions we have defined rarely occur. The proof shall be given shortly.

Lemma 2 [**Collision-bounding lemma**] *Let* $\mathrm{Mcoll}_n(\cdot)$ *and* $\mathrm{MMcoll}_n(\cdot,\cdot)$ *denote the M-collision probability and the MM-collision probability. Then*

$$\mathrm{Mcoll}_n(m) \;\le\; \binom{m+1}{2}\cdot\frac{1}{2^n} \qquad \text{and} \qquad \mathrm{MMcoll}_n(m,\bar m) \;\le\; \frac{m\bar m}{2^n}$$

Concluding the theorem. Pseudorandomness of PMAC, Theorem 1, follows by combining Lemmas 1 and 2. Namely,

$$\begin{aligned}
&\mathbf{Adv}^{\mathrm{prf}}_{\mathrm{PMAC}[\mathrm{Perm}(n),\tau]}\\
&\le \max_{\substack{m_1,\ldots,m_q\\ \sigma=\sum m_i\\ m_i\ge 1}}\left\{\sum_{1\le r\le q}\mathrm{Mcoll}_n(m_r)+\sum_{1\le r<s\le q}\mathrm{MMcoll}_n(m_r,m_s)\right\}+\frac{(\sigma+1)^2}{2^{n+1}}\\
&\le \max_{\substack{m_1,\ldots,m_q\\ \sigma=\sum m_i\\ m_i\ge 1}}\left\{\sum_{1\le r\le q}\mathrm{Mcoll}_n(m_r)\right\}\\
&\qquad\qquad+\max_{\substack{m_1,\ldots,m_q\\ \sigma=\sum m_i\\ m_i\ge 0}}\left\{\sum_{1\le r<s\le q}\mathrm{MMcoll}_n(m_r,m_s)\right\}+\frac{(\sigma+1)^2}{2^{n+1}}\\
&\le \max_{\substack{m_1,\ldots,m_q\\ \sigma=\sum m_i\\ m_i\ge 0}}\left\{\sum_{1\le r\le q}\binom{m_r+1}{2}\cdot\frac{1}{2^n}\right\}\\
&\qquad\qquad+\max_{\substack{m_1,\ldots,m_q\\ \sigma=\sum m_i\\ m_i\ge 0}}\left\{\sum_{1\le r<s\le q}\frac{m_r m_s}{2^n}\right\}+\frac{(\sigma+1)^2}{2^{n+1}}\\
&\le \frac{(\sigma+1)^2}{2^n}+\frac{(\sigma^2/2)}{2^n}+\frac{(\sigma+1)^2}{2^{n+1}} \qquad (3)\\
&\le \frac{2\,(\sigma+1)^2}{2^n}
\end{aligned}$$

where (3) follows because the first sum is maximized with a single message of length σ, while the second sum is maximized by q messages of length σ/q. (These claims can be justified using the method of Lagrange multipliers.) This completes the proof of Theorem 1.

Proof of Lemma 2. We now bound $\mathrm{Mcoll}_n(m)$ and $\mathrm{MMcoll}_n(m,\bar m)$. To begin, let $\mathit{Unequal}' = \mathit{Unequal}\setminus\{\bar X[\bar m]\}$ (multiset difference: remove one copy of $\bar X[\bar m]$) and define

$$D_1=\{0^n\} \qquad D_2=\{X[1],\ldots,X[m-1]\} \qquad D_3=\{X[m]\}$$

$$D_4 = \{\bar{X}[j] : \ j \in \mathit{Unequal}'\} \qquad D_5 = \{\bar{X}[\bar{m}]\}$$

We first show that for any two points $X[i]$ and $X[j]$ in the multiset $D_1 \cup D_2 \cup D_3$, where $i < j$ and $X[0] = 0^n$, the probability that these two points collide is at most 2^{-n}. The inequality

$$\mathrm{Mcoll}_n(m) \quad \leq \quad \binom{m+1}{2} \cdot \frac{1}{2^n}$$

follows because there are $m + 1$ points in $D_1 \cup D_2 \cup D_3$. Afterwards, we show that for any point $X[i]$ in $D_2 \cup D_3$ and any point in $\bar{X}[j]$ in $D_4 \cup D_5$, the probability that they collide is at most 2^{-n}. The inequality

$$\mathrm{MMcoll}_n(m, \bar{m}) \quad \leq \quad \frac{m\bar{m}}{2^n}$$

follows because $|D_2 \cup D_3| \cdot |D_4 \cup D_5| \leq m\bar{m}$.

To show $\mathrm{Mcoll}_n(m) \leq \binom{m+1}{2} \cdot \frac{1}{2^n}$ consider the following four cases:

Case $(D_1,\ D_2)$: $\Pr[0^n = X[i]] = \Pr[M[i] \oplus \gamma_i \cdot L = 0^n] = \Pr[L = \gamma_i^{-1} \cdot M[i]] = 2^{-n}$. We have used that γ_i is nonzero and we are working in a field. (We will continue to use this without mention.)

Case $(D_1,\ D_3)$: If $|M[m]| < n$ and $m \geq 2$ then Σ is a random n-bit string and so $\Pr[0^n = X[m]] = \Pr[0^n = \Sigma] = \Pr[0^n = Y[1] \oplus \cdots Y[m-1] \oplus \mathsf{pad}(M[m]))] = 2^{-n}$. If $|M[m]| = n$ and $m \geq 2$ then Σ is a random n-bit string that is independent of L and so $\Pr[0^n = X[m]] = \Pr[0^n = \Sigma \oplus \mathit{huge} \cdot L] = 2^{-n}$. If $|M[m]| < n$ and $m = 1$ then $\Pr[0^n = X[1]] = \Pr[0^n = \mathsf{pad}(M[m])] = 0$. If $|M[m]| = n$ and $m = 1$ then $\Pr[0^n = X[1]] = \Pr[0^n = \mathsf{pad}(M[m]) \oplus \mathit{huge} \cdot L] = 2^{-n}$.

Case $(D_2,\ D_2)$: For $i, j \in [1..m-1]$, $i < j$, $\Pr[X[i] = X[j]] = \Pr[M[i] \oplus \gamma_i \cdot L = M[j] \oplus \gamma_j \cdot L] = \Pr[M[i] \oplus M[j] = (\gamma_i \oplus \gamma_j) \cdot L] = 2^{-n}$ because $\gamma_i \neq \gamma_j$ for $i \neq j$. (Here one assumes that $j < 2^n$ because the lemma gives a non-result anyway if j were larger.)

Case $(D_2,\ D_3)$: Assume that $m \geq 2$, for otherwise there is nothing to show. Suppose first that $|M[m]| < n$. Then $\Pr[X[i] = X[m]] = \Pr[M[i] \oplus \gamma_i \cdot L = \Sigma]$. The value Σ is uniformly random and independent of L, so this probability is 2^{-n}. Suppose next that $|M[m]| = n$. Then $\Pr[X[i] = X[m]] = \Pr[M[i] \oplus \gamma_i \cdot L = \Sigma \oplus \mathit{huge} \cdot L] = \Pr[M[i] \oplus \Sigma = (\gamma_i \oplus \mathit{huge}) \cdot L]$. This value is 2^{-n} since $\gamma_i \neq \mathit{huge}$. Here we are assuming that $i < 2^{n-1}$, which is without loss of generality since a larger value of i, and therefore m, would give a non-result in the theorem statement.

Moving on, to show that $\mathrm{MMcoll}_n(m, \bar{m}) \leq \frac{m\bar{m}}{2^n}$ we verify the following four cases:

Case (D_2, D_4): Let $i \in [1..m-1]$ and $j \in \mathit{Unequal}'$ and consider $\Pr[X[i] = \bar{X}[j]] = \Pr[M[i] \oplus \gamma_i \cdot L = \bar{M}[j] \oplus \gamma_j \cdot L] = \Pr[M[i] \oplus \bar{M}[j] = (\gamma_i \oplus \gamma_j) \cdot L$. If $i \neq j$ then $\gamma_i \neq \gamma_j$ and this probability is 2^{-n}. If $i = j$ then the probability is 0 since, necessarily, $M[i] \neq \bar{M}[j]$.

Case (D_2, D_5): Suppose that $|\bar{M}[\bar{m}]| < n$. Then $\Pr[X[i] = \bar{X}[\bar{m}]] = \Pr[M[i] \oplus \gamma_i \cdot L = \bar{\Sigma}] = 2^{-n}$ because $\bar{\Sigma}$ is independent of L. Suppose that $|\bar{M}[\bar{m}]| = n$. Then $\Pr[X[i] = \bar{X}[\bar{m}]] = \Pr[M[i] \oplus \gamma_i \cdot L = \bar{\Sigma} \oplus \mathit{huge} \cdot L] = \Pr[M[i] \oplus \bar{\Sigma} = (\gamma_i \oplus \mathit{huge}) \cdot L] = 2^{-n}$ because $\bar{\Sigma}$ is independent of L and $\gamma_i \neq \mathit{huge}$.

Case (D_3, D_4): Suppose that $|M[m]| < n$. Then $\Pr[X[m] = \bar{X}[j]] = \Pr[\Sigma = \bar{M}[j] \oplus \gamma_j \cdot L] = 2^{-n}$ because Σ is independent of L. Suppose that $|M[m]| = n$. Then $\Pr[X[m] = \bar{X}[j]] = \Pr[\Sigma \oplus \mathit{huge} \cdot L = \bar{M}[j] \oplus \gamma_j \cdot L] = \Pr[\Sigma \oplus \bar{M}[j] = (\gamma_j \oplus \mathit{huge}) \cdot L] = 2^{-n}$ because $\gamma_j \neq \mathit{huge}$.

Case (D_3, D_5): Suppose that $|M[m]| < n$ and $|\bar{M}[\bar{m}]| < n$. If $m > \bar{m}$ then $\Pr[X[m] = \bar{X}[\bar{m}]] = \Pr[\Sigma = \bar{\Sigma}] = 2^{-n}$ because of the contribution of $Y[m-1]$ in Σ—a random variable that is not

Algorithm	16 B	128 B	2 KB
PMAC-AES128	22.1	18.7	18.4
CBCMAC-AES128	18.9	17.4	17.1

Figure 4: **Performance results.** Numbers are in cycles per byte (cpb) on a Pentium 3, for three message lengths, the code written in assembly.

used in the definition of $\bar{\Sigma}$. If $m < \bar{m}$ then $\Pr[X[m] = \bar{X}[\bar{m}]] = \Pr[\Sigma = \bar{\Sigma}] = 2^{-n}$ because of the contribution of $\bar{Y}[\bar{m}-1]$ in $\bar{\Sigma}$—a random variable that is not used in the definition of Σ. If $m = \bar{m}$ and there is an $i < m$ such that $M[i] \neq \bar{M}[i]$ then $\Pr[X[m] = \bar{X}[\bar{m}]] = \Pr[\Sigma = \bar{\Sigma}] = 2^{-n}$ because of the contribution of $\bar{Y}[i]$ in $\bar{\Sigma}$—a random variable that is not used in the definition of Σ. If $m = \bar{m}$ and for every $i < m$ we have that $M[i] = \bar{M}[i]$, then, necessarily, $M[m] \neq \bar{M}[m]$. In this case $\Pr[\Sigma = \bar{\Sigma}] = 0$, as the two checksums differ by the nonzero value $\mathsf{pad}(M[m]) \oplus \mathsf{pad}(\bar{M}[m])$.

Suppose that $|M[m]| = n$ and $|\bar{M}[\bar{m}]| = n$. Then $X[m]$ and $\bar{X}[m]$ are offset by the same amount, $\mathit{huge} \cdot L$, so this offset is irrelevant in computing $\Pr[X[m] = \bar{X}[\bar{m}]$. Proceed as above.

Suppose that $|M[m]| < n$ and $|\bar{M}[\bar{m}]| = n$. Then $\Pr[X[m] = \bar{X}[m]] = \Pr[\Sigma = \bar{\Sigma} \oplus \mathit{huge} \cdot L] = 2^{-n}$ since Σ and $\bar{\Sigma}$ are independent of L. Similarly, if $|M[m]| = n$ and $|\bar{M}[\bar{m}]| < n$, then $\Pr[X[m] = \bar{X}[m]] = 2^{-n}$. This completes the proof.

6 Performance

A colleague, Ted Krovetz, implemented PMAC-AES128 and compared its performance in an entirely sequential setting to that of CBCMAC-AES128. The latter refers to the "basic" CBC MAC; nothing is done to take care of length-variability or the possibility of strings which are not a multiple of the block length. The code was written in modestly-optimized assembly under Windows 2000 sp1 and Visual C++ 6.0 sp4. All data fit into L1 cache. Disregarding the one-block message in Figure 4 we see that, in a serial environment, PMAC-AES128 was about 8% slower than CBCMAC-AES128. A more aggressively optimized implementation of CBCMAC-AES128, due to Helger Lipmaa, achieves 15.5 cpb for 1 KByte message lengths [20]. Adding the same 8%, we expect that this code could be modified to compute PMAC at a rate of about 16.7 cpb. In general, differences in implementation quality would seem to be a more significant in determining implementation speed than the algorithmic difference between PMAC and the CBC MAC.

Though some or all of the $L(i)$-values are likely to be pre-computed, calculating all of these values "on the fly" is not expensive. Starting with 0^n we form successive offsets by xoring the previous offset with L, $2 \cdot L$, L, $4 \cdot L$, L, $2 \cdot L$, L, $8 \cdot L$, and so forth, making the expected number of $a \cdot \mathtt{x}$-operations to compute an offset at most $\sum_{i=1}^{\infty} i/2^{i+1} = 1$. For $n = 128$, each $a \cdot \mathtt{x}$ instruction requires a 128-bit shift and a conditional 32-bit xor.

Acknowledgments

We would like to thank several people for helpful comments on drafts of this manuscript: Michael Amling, Mihir Bellare, Johan Håstad, Ted Krovetz, David McGrew, David Wagner, and the Eurocrypt program committee. Special thanks to Ted, who wrote reference code, prepared test vectors, and collected performance data. Virgil Gligor described [14] to Rogaway at Crypto '00 and that event, along with the announcement of a NIST modes-of-operation workshop in October 2000, inspired the current work.

This paper was written while Rogaway was on leave from UC Davis, visiting the Department of Computer Science, Faculty of Science, Chiang Mai University.

References

[1] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. *Advances in Cryptology — CRYPTO '96.* Lecture Notes in Computer Science, vol. 1109, Springer-Verlag, pp. 1–15, 1996. Available at URL www-cse.ucsd.edu/users/mihir

[2] M. Bellare, S. Goldwasser, and O. Goldreich. Incremental cryptography and applications to virus protection. *Proceedings of the 27th Annual ACM Symposium on the Theory of Computing* (STOC '95). ACM Press, pp. 45–56, 1995. Available at URL www.cs.ucdavis.edu/∼rogaway

[3] M. Bellare, R. Guérin and P. Rogaway. "XOR MACs: New methods for message authentication using finite pseudorandom functions." *Advances in Cryptology — CRYPTO '95.* Lecture Notes in Computer Science, vol. 963, Springer-Verlag, pp. 15–28, 1995. Available at URL www.cs.ucdavis.edu/∼rogaway

[4] M. Bellare, J. Kilian, and P. Rogaway. The security of the cipher block chaining message authentication code. *Journal of Computer and System Sciences*, vol. 61, no. 3, Dec 2000. (Full version of paper from *Advances in Cryptology — CRYPTO '94.* Lecture Notes in Computer Science, vol. 839, pp. 340–358, 1994.) Available at URL www.cs.ucdavis.edu/∼rogaway

[5] M. Bellare and P. Rogaway. Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient encryption. *Advances in Cryptology — ASIACRYPT '00.* Lecture Notes in Computer Science, vol. 1976, Springer-Verlag, 2000. Available at URL www.cs.ucdavis.edu/∼rogaway

[6] A. Berendschot, B. den Boer, J.P. Boly, A. Bosselaers, J. Brandt, D. Chaum, I. Damgård, M. Dichtl, W. Fumy, M. van der Ham, C.J.A. Jansen, P. Landrock, B. Preneel, G. Roelofsen, P. de Rooij, and J. Vandewalle. Integrity primitives for secure information systems, Final report of RACE integrity primitives evaluation (RIPE-RACE 1040). Lecture Notes in Computer Science, vol. 1007, Springer-Verlag, 1995.

[7] D. Bernstein. Floating-point arithmetic and message authentication. Unpublished manuscript. Available at URL http://cr.yp.to/papers.html#hash127

[8] D. Bernstein. How to stretch random functions: the security of protected counter sums. *Journal of Cryptology*, vol. 12, no. 3, pp. 185–192 (1999). Available at URL cr.yp.to/djb.html

[9] J. Black, S. Halevi, H. Krawczyk, T. Krovetz, and P. Rogaway. UMAC: Fast and secure message authentication. *Advances in Cryptology — CRYPTO '99.* Lecture Notes in Computer Science, Springer-Verlag, 1999. Available at URL www.cs.ucdavis.edu/∼rogaway

[10] J. Black and P. Rogaway. A block-cipher mode of operation for parallelizable message authentication. Full version of this paper. Available at URL www.cs.ucdavis.edu/∼rogaway

[11] J. Black and P. Rogaway. CBC MACs for arbitrary-length messages: The three-key constructions. Full version of paper from *Advances in Cryptology — CRYPTO '00.* Lecture Notes in Computer Science, vol. 1880, pp. 197–215, 2000. Available at URL www.cs.ucdavis.edu/∼rogaway

[12] G. Brassard. On computationally secure authentication tags requiring short secret shared keys. *Advances in Cryptology — CRYPTO '82.* Plenum Press, pp. 79–86, 1983.

[13] L. Carter and M. Wegman. Universal hash functions. *J. of Computer and System Sciences.* vol. 18, pp. 143–154, 1979.

[14] V. Gligor and P. Donescu. Fast encryption and authentication: XCBC encryption and XECB authentication modes. *Fast Software Encryption*, Lecture Notes in Computer Science, Springer-Verlag, April 2001. Available at URL www.eng.umd.edu/∼gligor

[15] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM,* vol. 33, no. 4, pp. 210–217, 1986.

[16] S. Halevi and H. Krawczyk. MMH: Software message authentication in the Gbit/second rates. *Fast Software Encryption* (FSE 4), Lecture Notes in Computer Science, vol. 1267, Springer-Verlag, pp. 172–189, 1997. Available at URL www.research.ibm.com/people/s/shaih

[17] Iso/Iec 9797. Information technology – Security techniques – Data integrity mechanism using a cryptographic check function employing a block cipher algorithm. International Organization for Standards (ISO), Geneva, Switzerland, 1994 (second edition).

[18] C. Jutla. Encryption modes with almost free message integrity. *Advances in Cryptology — EUROCRYPT 2001.* Lecture Notes in Computer Science, vol. 2045, B. Pfitzmann, ed., Springer-Verlag, 2001.

[19] H. Krawczyk. LFSR-based hashing and authentication. *Advances in Cryptology — CRYPTO '94.* Lecture Notes in Computer Science, vol. 839, Springer-Verlag, pp 129–139, 1994.

[20] H. Lipmaa. Personal communication, July 2001. Further information available at www.tcs.hut.fi/∼helger

[21] E. Petrank and C. Rackoff. CBC MAC for real-time data sources. *Journal of Cryptology*, vol. 13, no. 3, pp. 315–338, Nov 2000. Available at URL www.cs.technion.ac.il/∼erez/publications.html. Earlier version as 1997/010 in the Cryptology ePrint archive, eprint.iacr.org

[22] B. Preneel. Cryptographic primitives for information authentication — State of the art. *State of the Art in Applied Cryptography*, COSIC '97, LNCS 1528, B. Preneel and V. Rijmen, eds., Springer-Verlag, pp. 49–104, 1998.

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

A Proof of the Structure Lemma

Let A be an adversary that attacks PMAC[Perm(n), τ]. Since A is computationally unbounded, there is no loss of generality to assume that A is deterministic. One can imagine A interacting with a PMAC[Perm(n), τ] oracle as A playing a certain game, Game 1, as defined in Figure 5. This game perfectly simulates the behavior of PMAC[Perm(n), τ]. It does so in a somewhat unusual way, sometimes setting a flag *bad* to `true`. We observe that if the flag *bad* is *not* set to `true` in an execution of the game, then the value Tag_r returned by the game at line 33 is a random one—the first τ bits of the string randomly selected at line 29. It follows that $\mathbf{Adv}_{\mathrm{PMAC}[\mathrm{Perm}(n),\tau]}(A)$ is at most the probability that *bad* gets set to `true` in Game 1. The rest of the proof is devoted to bounding this probability.

We first consider the probability that *bad* gets set to `true` in line 24 or 30. In both cases, we have just chosen a random n-bit string and then we are testing it for membership in a set. The

```
**Initialization**
10    $L \xleftarrow{R} \{0,1\}^n$;   $\pi(0^n) \leftarrow L$

**When $A$ makes its $r$-th query, $M_r = M_r[1]\cdots M_r[m_r]$, where $r \in [1..q]$**
20    for $i \leftarrow 1$ to $m_r - 1$ do
21        $X_r[i] \leftarrow M_r[i] \oplus \gamma_i \cdot L$
22        if $X_r[i] \in \text{Domain}(\pi)$ then $Y_r[i] \leftarrow \pi(X_r[i])$
23        else $Y_r[i] \xleftarrow{R} \{0,1\}^n$
24            if $Y_r[i] \in \text{Range}(\pi)$ then { $bad \leftarrow$ true;  $Y_r[i] \xleftarrow{R} \overline{\text{Range}}(\pi)$ }
25            $\pi(X_r[i]) \leftarrow Y_r[i]$
26    $\Sigma_r \leftarrow Y_r[1] \oplus Y_r[2] \oplus \cdots \oplus Y_r[m_r - 1] \oplus \mathsf{pad}(M_r[m_r])$
27    if $|M_r[m_r]| = n$ then $X_r[m_r] \leftarrow \Sigma_r \oplus huge \cdot L$ else $X_r[m_r] \leftarrow \Sigma_r$
28    if $X_r[m_r] \in \text{Domain}(\pi)$ then { $bad \leftarrow$ true;   $TAG_r \xleftarrow{R} \overline{\text{Range}}(\pi)$ }
29    else $TAG_r \xleftarrow{R} \{0,1\}^n$
30        if $TAG_r \in \text{Range}(\pi)$ then { $bad \leftarrow$ true;   $TAG_r \xleftarrow{R} \overline{\text{Range}}(\pi)$ }
31        $\pi(X_r[m_r]) \leftarrow TAG_r$
32    $Tag_r \leftarrow TAG_r$  [first $\tau$ bits]
33    return $Tag_r$
```

Figure 5: **Game 1.** This game accurately simulates PMAC[Perm(n), τ].

size of this set starts at 1 (after executing line 10) and grows one element at a time until, by the time just before the last addition of a point, it has size σ. Thus we have that

$$\Pr_1[bad \text{ gets set in lines 24 or 30 }] \;\le\; \frac{1+2+\ldots+\sigma}{2^n} \;\le\; \frac{(\sigma+1)^2}{2^{n+1}} \qquad (4)$$

Here the subscript of 1 in the probability reminds us that we are considering the behavior of Game 1.

We can now modify Game 1 by changing the behavior when and only when *bad* is set, and adding as a compensating factor the bound given by Equation (4). In particular, we may simply omit lines 24 and 30, and the second statement in the compound statement of line 28 along with the following **else**. The modified game is rewritten in Figure 6. At this point we know that

$$\mathbf{Adv}^{\text{prf}}_{\text{PMAC[Perm}(n),\tau]}(A) \;\le\; \Pr_2[bad \text{ gets set }] + \frac{(\sigma+1)^2}{2^{n+1}} \qquad (5)$$

Here the subscript of 2 in the probability reminds us that we are considering the behavior of Game 2.

Notice in Game 2 that the value Tag_r returned in response to a query M is always a random τ-bit string. But of course the game does more than just return these strings: it also chooses L at random, fills in π-values, and sets *bad* under certain conditions. We can defer doing all those things, and just return the random strings $Tag_1, \ldots, Tag_q$. This does not change the view of the adversary that interacts with the game, nor will it change the probability that *bad* is set to `true`. The modified game is called Game 3, and it is depicted in Figure 7.

We need to bound the probability that *bad* gets set to `true` in Game 3. This probability is over the random TAG_r-values selected at line 10, the random value of L selected at line 20, and the random $Y_r[i]$-values selected at line 25. We want to show that, over these random values,

```
Initialization
10    $L \xleftarrow{R} \{0,1\}^n$;   $\pi(0^n) \leftarrow L$

When $A$ makes its $r$-th query, $M_r = M_r[1] \cdots M_r[m_r]$, where $r \in [1..q]$
20    for $i \leftarrow 1$ to $m_r - 1$ do
21        $X_r[i] \leftarrow M_r[i] \oplus \gamma_i \cdot L$
22        if $X_r[i] \in \mathrm{Domain}(\pi)$ then $Y_r[i] \leftarrow \pi(X_r[i])$
23        else { $Y_r[i] \xleftarrow{R} \{0,1\}^n$;   $\pi(X_r[i]) \leftarrow Y_r[i]$ }
24    $\Sigma_r \leftarrow Y_r[1] \oplus Y_r[2] \oplus \cdots \oplus Y_r[m_r - 1] \oplus \mathsf{pad}(M_r[m_r])$
25    if $|M_r[m_r]| = n$ then $X_r[m_r] \leftarrow \Sigma_r \oplus \mathit{huge} \cdot L$ else $X_r[m_r] \leftarrow \Sigma_r$
26    if $X_r[m_r] \in \mathrm{Domain}(\pi)$ then $\mathit{bad} \leftarrow$ true
27    $\mathit{TAG}_r[m_r] \xleftarrow{R} \{0,1\}^n$
28    $\pi(X_r[m_r]) \leftarrow \mathit{TAG}_r[m_r]$
29    $\mathit{Tag}_r \leftarrow \mathit{TAG}_r$ [first $\tau$ bits]
30    return $\mathit{Tag}_r$
```

Figure 6: **Game 2**. A simplification of Game 1. Bounding the probability that *bad* gets set in this game, and then adding a correction factor, serves to bound $\mathbf{Adv}^{\mathrm{prf}}_{\mathrm{PMAC}[\mathrm{Perm}(n),\tau]}$.

```
When $A$ makes its $r$-th query, $M_r = M_r[1] \cdots M_r[m_r]$, where $r \in [1..q]$
10    $\mathit{TAG}_r \xleftarrow{R} \{0,1\}^n$
11    return $\mathit{TAG}_r$ [first $\tau$ bits]

When $A$ is done making its $q$ queries
20    $L \xleftarrow{R} \{0,1\}^n$;   $\pi(0^n) \leftarrow L$
21    for $r \leftarrow 1$ to $q$ do
22        for $i \leftarrow 1$ to $m_r - 1$ do
23            $X_r[i] \leftarrow M_r[i] \oplus \gamma_i \cdot L$
24            if $X_r[i] \in \mathrm{Domain}(\pi)$ then $Y_r[i] \leftarrow \pi(X_r[i])$
25            else { $Y_r[i] \xleftarrow{R} \{0,1\}^n$;   $\pi(X_r[i]) \leftarrow Y_r[i]$ }
26        $\Sigma_r \leftarrow Y_r[1] \oplus Y_r[2] \oplus \cdots \oplus Y_r[m_r - 1] \oplus \mathsf{pad}(M_r[m_r])$
27        if $|M_r[m_r]| = n$ then $X_r[m_r] \leftarrow \Sigma_r \oplus \mathit{huge} \cdot L$ else $X_r[m_r] \leftarrow \Sigma_r$
28        if $X_r[m_r] \in \mathrm{Domain}(\pi)$ then $\mathit{bad} \leftarrow$ true
29        $\pi(X_r[m_r]) \leftarrow \mathit{TAG}_r$
```

Figure 7: **Game 3**. Like Game 2, but we defer all but the selection of TAG_r-values. This does not change the view of the adversary or the chance that *bad* will be set to `true`.

```
10    $L \xleftarrow{R} \{0,1\}^n$;   $\pi(0^n) \leftarrow L$
11    **for** $r \leftarrow 1$ **to** $q$ **do**
12        **for** $i \leftarrow 1$ **to** $m_r - 1$ **do**
13            $X_r[i] \leftarrow M_r[i] \oplus \gamma_i \cdot L$
14            **if** $X_r[i] \in \text{Domain}(\pi)$ **then** $Y_r[i] \leftarrow \pi(X_r[i])$
15            **else** { $Y_r[i] \xleftarrow{R} \{0,1\}^n$;   $\pi(X_r[i]) \leftarrow Y_r[i]$ }
16        $\Sigma_r \leftarrow Y_r[1] \oplus Y_r[2] \oplus \cdots \oplus Y_r[m_r - 1] \oplus \mathsf{pad}(M_r[m_r])$
17        **if** $|M_r[m_r]| = n$ **then** $X_r[m_r] \leftarrow \Sigma_r \oplus \mathit{huge} \cdot L$ **else** $X_r[m_r] \leftarrow \Sigma_r$
18        **if** $X_r[m_r] \in \text{Domain}(\pi)$ **then** $\mathit{bad} \leftarrow \texttt{true}$
19        $\pi(X_r[m_r]) \leftarrow \mathit{TAG}_r$
```

Figure 8: **Game 4**[$\mathcal{C}$]. This game depends on constants $\mathcal{C}$ which specify: q, $\mathit{TAG}_1, \ldots, \mathit{TAG}_q \in \{0,1\}^n$, and $M_1 = M_1[1] \cdots M_1[m_1], \ldots, M_q = M_q[1] \cdots M_q[m_q]$.

```
10    $L \xleftarrow{R} \{0,1\}^n$;   $\pi(0^n) \leftarrow L$
11    **for** $r \leftarrow 1$ **to** $q$ **do**
12        **for** $i \leftarrow 1$ **to** $m_r - 1$ **do**
13            $X_r[i] \leftarrow M_r[i] \oplus \gamma_i \cdot L$;   $Y_r[i] \xleftarrow{R} \{0,1\}^n$
14            **if** $M_r[i] = M_s[i]$ for some $s < r$ and $i < m_s$ **then** $Y_r[i] \leftarrow \pi(X_s[i])$
15            **else if** $X_r[i] \in \text{Domain}(\pi)$ **then** $\mathit{bad} \leftarrow \texttt{true}$
16            $\pi(X_r[i]) \leftarrow Y_r[i]$
17        $\Sigma_r \leftarrow Y_r[1] \oplus Y_r[2] \oplus \cdots \oplus Y_r[m_r - 1] \oplus \mathsf{pad}(M_r[m_r])$
18        **if** $|M_r[m_r]| = n$ **then** $X_r[m_r] \leftarrow \Sigma_r \oplus \mathit{huge} \cdot L$ **else** $X_r[m_r] \leftarrow \Sigma_r$
19        **if** $X_r[m_r] \in \text{Domain}(\pi)$ **then** $\mathit{bad} \leftarrow \texttt{true}$
20        $\pi(X_r[m_r]) \leftarrow 0^n$
```

Figure 9: **Game 5**[$\mathcal{C}$]. This game sets *bad* at least as often as Game 4[$\mathcal{C}$] does. It is this game that is related back to the one that defines the collision probabilities Mcoll_n and MMcoll_n.

bad will rarely be set. In fact, we show something stronger: that even if one arbitrarily fixes the values of $\mathit{TAG}_1, \ldots, \mathit{TAG}_q \in \{0,1\}^n$ (and takes the probability over just the remaining values), still the probability that *bad* will be set to `true` is small. Since the oracle responses have now been fixed, and since the adversary itself is deterministic, the queries $M_1, \ldots, M_q$ that the adversary will generate have likewise been fixed. Interaction and the adversary itself are essentially gone at this point, replaced by universal quantification. The new game is show in Figure 8. It depends on constants $\mathcal{C} = (q,\ \mathit{TAG}_1, \ldots, \mathit{TAG}_q,\ M_1, \ldots, M_q)$. At this point in the proof we have that

$$\mathbf{Adv}^{\text{prf}}_{\text{PMAC}[\text{Perm}(n),\tau]}(A) \leq \max_{\mathcal{C}}\{\Pr[\ \mathit{bad} \text{ gets set to } \texttt{true} \text{ in game } 4[\mathcal{C}]\} + \frac{(\sigma+1)^2}{2^{n+1}} \qquad (6)$$

where, if A is limited to q queries of aggregate length σ, then $\mathcal{C}$ specifies q, strings $M_1, \ldots, M_q$ of aggregate block length σ, and $\mathit{TAG}_1, \ldots, \mathit{TAG}_q \in \{0,1\}^n$.

The next step is to modify Game 4 so that the new game, Game 5, sets *bad* every time that Game 4 does, plus some additional times. Look at line 14 in Game 4. The value $X_r[i]$ could have been in the domain of π for the "trivial" reason that $M_r[i] = M_s[i]$ for some $s < r$ and where $i < m_s$, or for some other, "non-trivial" reason: $X_r[i] = 0^n$, or $X_r[i] = X_s[j]$ for some $s < r$ and

$j \neq i$, or $X_r[i] = X_r[j]$ for some $j < i$. If $X_r[i]$ was in the domain of π for a non-trivial reason, we effectively give up, setting *bad* to `true`. Thus in Game 5, line 15, we set *bad* if $X_r[i]$ is already in the domain of π and it is not due to the trivial cause (which is tested for in line 14). We also modify the last line, setting $X_r[m_r]$ to some particular value, say 0^n, instead of to TAG_r. The only significance of this assignment was to make π defined at the point $X_r[m_r]$; the particular value associated to this point is not used unless *bad* has already been set to `true`.

The coins used in Game 5 are L and $Y_1 = Y_1[1] \cdots Y_1[m_1]$, $\ldots$, $Y_q = Y_q[1] \cdots Y_q[m_q]$, where $Y_s[i]$ are either random coins or are a synonym $Y_u[i]$ where u is the least number such that $u < s$ and $i < m_u$ and $M_s[i] = M_u[i]$ (whenever such a u exists). Run Game 5 on $M_1, \ldots, M_q$ and the indicated vector of coins. Suppose that *bad* gets set to `true` on this execution. Let $s \in [1..q]$ be the particular value of r when *bad* first got set to `true`. We differentiate the following possibilities:

- Case 1. When *bad* was first set to `true`, this happened because $X_s[i] = 0^n$ for some s, or this happened because $X_s[i] = X_s[j]$ for some $j \in [1..i-1]$.
- Case 2. When *bad* got set to `true` this happened because $X_s[i] = X_u[j]$ for some $u \in [1..s-1]$ (and if $i \neq j$ then $M_s[i] \neq M_u[j]$).

In the first case, if we had run game 5 using coins L and Y_s, dropping line 14 and restricting the execution of line 12 to $r = s$, then *bad* still would have been set to `true`. This exactly coincides with the game that defines $\mathrm{Mcoll}_n(M_s)$. Thus the probability that case 1 occurs for M_s, is at most $\mathrm{Mcoll}_n(M_s)$, and, by the sum bound, the probability that case 1 occurs is at most $\sum_{1 \leq r \leq q} \mathrm{Mcoll}_n(m_r)$.

In the second case, if we had run game 5 using coins L, Y_s and Y_u, restricting the execution of line 12 to $r \in \{s, u\}$, then *bad* still would have been set to `true`. This exactly coincides with the game that defines $\mathrm{Mcoll}_n(M_s, M_u)$. Thus the probability that case 2 occurs for M_s, M_u, is at most $\mathrm{MMcoll}_n(M_s, M_u)$, and the probability that case 2 occurs due to (M_s, M_u) is at most $\mathrm{MMcoll}_n(M_s, M_u)$ and, by the sum bound, the probability of an MM collision is at most $\sum_{1 \leq s < u \leq q} \mathrm{MMcoll}_n(m_s, m_u)$. This completes the proof.