PMAC

Proposal to NIST for a parallelizable message authentication code

Submitted on April 1, 2001
Revised on April 18, 2001

Submitter:

Phillip Rogaway

rogaway@cs.ucdavis.edu
http://www.cs.ucdavis.edu/∼rogaway

Department of Computer Science
University of California at Davis
Engineering II Building
Davis, California 95616 USA

+1 530 752 7583 office
+1 530 752 4767 fax

Department of Computer Science
Faculty of Science
Chiang Mai University
Chiang Mai 50200 Thailand

+66 1 881 8290 cell
+66 53 943 433 fax

Inventor and Owner:
Same as submitter

Auxiliary submitter:

John Black
jrb@cs.unr.edu

Department of Computer Science
University of Nevada
Reno, Nevada 89557 USA

Contents

1 Introduction **1**

2 Mathematical Preliminaries **1**

3 Specification **4**
3.1 Definition of the Mode 4
3.2 Conformance Criteria 4
3.3 An Equivalent Description 6

4 Discussion **8**
4.1 Properties 8
4.2 Design Rationale 10
4.3 Limitations 11
4.4 Related Work 11
4.5 Design History 12

5 Theorems **13**
5.1 Security Definitions 13
5.2 Theorem Statements 13
5.3 Structure of the Proof 14

6 Performance **16**

7 Intellectual Property Statement and Disclosures **17**

Acknowledgments **18**

References **18**

A Proofs **20**
A.1 Proof of the Structure Lemma (Lemma 1) 20
A.2 Proof of the Collision Bound (Lemma 2) 23

B Test Vectors **25**

C Document History **25**

1 Introduction

Popular ways to authenticate a message, like the CBC MAC [18] and HMAC [1], are inherently sequential: one cannot process the i-th message block until all previous message blocks have been processed. The sequential nature of these algorithms can limit performance. While this has not been too big an obstacle in the past, the issue can be expected to increase in importance, both for hardware and for software. Special-purpose hardware will get limited by the latency of the underlying cryptographic primitive, while performance on commodity processors will be limited by an inability to fully exploit the multiple instruction pipes provided. Thus it seems a ripe time to develop a fully parallelizable message authentication code (MAC).

This submission describes such a message authentication code, PMAC (which stands for Parallelizable MAC). Using an n-bit block cipher, PMAC authenticates an arbitrary string $M \in \{0,1\}^*$ using $\lceil |M|/n \rceil$ block-cipher calls. (The empty string is an exception; it requires one block-cipher call.) Overhead beyond the block-cipher calls is low—about 8% more than with the basic CBC MAC when one is in a non-parallelizable environment. PMAC works correctly across messages of arbitrary and varying bit lengths. It uses a single key for the underlying block cipher. The length of the computed MAC is an arbitrary number of bits $\tau \in [1..n]$, with the corresponding forging probability being about $2^{-\tau}$. PMAC is stateless and deterministic: MAC generation does not require a nonce or random value.

We prove PMAC secure, in the sense of reduction-based cryptography. Specifically, we prove that PMAC is a good variable-input-length pseudorandom function (PRF), and is therefore a good MAC, as long as the underlying block cipher is good as a pseudorandom permutation (PRP) [5, 21]. The actual results are quantitative; the security analysis is in the concrete-security paradigm.

Parallelizable MACs related to PMAC are the XOR MAC of Bellare, Guérin and Rogaway [4], the variant of this construction due to Bernstein [9], and the XECB-MAC of Gligor and Donescu [13]. For details on these and other works, see Section 4.4.

2 Mathematical Preliminaries

NOTATION. If a and b are integers, $a \le b$, then $[a..b]$ is the set $\{a, a+1, \ldots, b\}$. If $i \ge 1$ is an integer then $\mathsf{ntz}(i)$ is the number of trailing 0-bits in the binary representation of i (equivalently, $\mathsf{ntz}(i)$ is the largest integer z such that 2^z divides i). So, for example, $\mathsf{ntz}(7) = 0$ and $\mathsf{ntz}(8) = 3$.

A *string* is a finite sequence of symbols, each symbol being 0 or 1. The string of length 0 is called the *empty string* and is denoted ε. Let $\{0,1\}^*$ denote the set of all strings. Let i, n be nonnegative integers. Then 0^i and 1^i denote the strings of i 0's and 1's, respectively. Let $\{0,1\}^n$ denote the set of all strings of length n. If $A \in \{0,1\}^*$ then $|A|$ denotes the length of A, in bits, while $\|A\|_n = \max\{1, \lceil |A|/n \rceil\}$ denotes the length of A in n-bit blocks, where the empty string counts as one block. If $A, B \in \{0,1\}^*$ then $A\,B$, or $A \parallel B$, is their concatenation. If $A \in \{0,1\}^*$ and $A \ne \varepsilon$ then $\mathsf{firstbit}(A)$ is the first bit of A and $\mathsf{lastbit}(A)$ is the last bit of A. If $A \in \{0,1\}^*$ and $i \in [0..|A|]$ then $A\,[\text{first } i \text{ bits}]$ and $A[\text{last } i \text{ bits}]$ denote the strings containing the first i bits of A and the last i bits of A, respectively. Both of these values are the empty string if $\tau = 0$. 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 $a \in [0..2^n - 1]$ then $\mathsf{num2str}_n(a)$ is the n-bit string A such that $\mathsf{str2num}(A) = a$. Let $\mathrm{len}_n(A) = \mathsf{num2str}_n(|A|)$. We omit the subscript when n is understood. If $A, B \in \{0,1\}^*$ then $A \oplus B$ is the bitwise xor of $A\,[\text{first } \ell \text{ bits}]$ and $B\,[\text{first } \ell \text{ bits}]$, where $\ell = \min\{|A|, |B|\}$ (where $\varepsilon \oplus \varepsilon = \varepsilon$). So, for example, $1001 \oplus 11 = 01$. 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 $A = a_{n-1}a_{n-2}\cdots a_1a_0 \in \{0,1\}^n$ then $A{\ll}1 = a_{n-2}a_{n-3}\cdots a_1a_0 0$ is the n-bit string which is a left shift of A by 1 bit (the first bit of A disappearing and a zero coming into the last bit), while $A{\gg}1 = 0a_{n-1}a_{n-2}\ldots a_2a_1$ is the n-bit string which is a right shift of A by one bit (the last bit disappearing and a zero coming into the first 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 $1 \le i < m$."

THE FIELD WITH 2^n POINTS. Recall that a finite field is a finite set together with an addition operation and a multiplication operation, each defined to take a pair of points in the field to another point in the field. The operations must obey certain basic axioms. (For example, there must be a point 0 in the field such that $a + 0 = 0 + a = a$ for every a; there must be a point 1 in the field such that $a \cdot 1 = 1 \cdot a = a$ for every a; and for every $a \neq 0$ there must be a point a^{-1} in the field such that $a \cdot a^{-1} = a^{-1} \cdot a = 1$.) If one fixes a positive integer n, then there turns out to be a unique finite field (up to the naming of the points) that has 2^n elements. It is called the Galois field of size 2^n, and it is denoted $\mathrm{GF}(2^n)$.

Example 1 *The field* $\mathrm{GF}(2)$ *has two points,* 0 *and* 1*, and operations* $\oplus$ *(addition) and* $\cdot$ *(multiplication) are defined by* $0 \oplus 0 = 0$, $0 \oplus 1 = 1$, $1 \oplus 0 = 1$, $1 \oplus 1 = 0$, $0 \cdot 0 = 0$, $0 \cdot 1 = 0$, $1 \cdot 0 = 0$, *and* $1 \cdot 1 = 1$.

We interchangeably think of a point a in $\mathrm{GF}(2^n)$ in any of the following ways: (1) as an abstract point in a field; (2) as an n-bit string $a_{n-1}\ldots a_1a_0 \in \{0,1\}^n$; (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 the string $a \in \{0,1\}^n$ corresponds to the number $\mathsf{str2num}(a)$. For example, one can regard the string $a = 0^{125}101$ as a 128-bit string, as the number 5, as the polynomial $\mathtt{x}^2 + 1$, or as a particular point in the finite field $\mathrm{GF}(2^{128})$. 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 $a \oplus b$.

Before we can say how to multiply two points we must fix some irreducible polynomial $p_n(\mathtt{x})$ having binary coefficients and degree n. (Saying that $p_n(\mathtt{x})$ is irreducible means that if $q(\mathtt{x})$ and $q'(\mathtt{x})$ are polynomials over $\mathrm{GF}(2)$ which multiply to give $p_n(\mathtt{x})$, then one of these polynomials is 1 and the other is $p_n(\mathtt{x})$.) For PMAC, choose the lexicographically first polynomial among the irreducible degree n polynomials having a minimum number of coefficients. For $n = 128$, the indicated polynomial is

$$p_{128}(\mathtt{x}) = \mathtt{x}^{128} + \mathtt{x}^7 + \mathtt{x}^2 + \mathtt{x} + 1$$

A few other $p_n(\mathtt{x})$-values are $\mathtt{x}^{64}+\mathtt{x}^4+\mathtt{x}^3+\mathtt{x}+1$ and $\mathtt{x}^{96}+\mathtt{x}^{10}+\mathtt{x}^9+\mathtt{x}^6+1$ and $\mathtt{x}^{160}+\mathtt{x}^5+\mathtt{x}^3+\mathtt{x}^2+1$ and $\mathtt{x}^{192} + \mathtt{x}^7 + \mathtt{x}^2 + \mathtt{x} + 1$ and $\mathtt{x}^{224} + \mathtt{x}^9 + \mathtt{x}^8 + \mathtt{x}^3 + 1$ and $\mathtt{x}^{256} + \mathtt{x}^{10} + \mathtt{x}^5 + \mathtt{x}^2 + 1$.

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 $\mathrm{GF}(2)$ (the coefficient of degree j in $c(\mathtt{x})$, where $j \in [0..2n-2]$, is $c_j = \oplus_{i=0}^{j}\,(b_i \cdot a_{j-i})$) and take the remainder one gets when dividing $c(\mathtt{x})$ by the polynomial $p_n(\mathtt{x})$.

By convention, the multiplication operator has higher precedence than addition operator so, for example, $\gamma_1 \cdot L \oplus R$ means $(\gamma_1 \cdot L) \oplus R$.

Example 2 *Assume* $n = 128$. *Suppose one multiplies* $a(\mathtt{x}) = \mathtt{x}^{127} + \mathtt{x} + 1$ *by* $b(\mathtt{x}) = \mathtt{x} + 1$. *The result is* $c(\mathtt{x}) = \mathtt{x}^{128} + \mathtt{x}^2 + \mathtt{x} + \mathtt{x}^{127} + \mathtt{x} + 1 = \mathtt{x}^{128} + \mathtt{x}^{127} + \mathtt{x}^2 + 1$. *If one divides* $c(\mathtt{x})$ *by* $p(\mathtt{x})$ *one gets a quotient of* $q(\mathtt{x}) = 1$ *and a remainder (which is the answer) of* $r(\mathtt{x}) = \mathtt{x}^{127} + \mathtt{x}^7 + \mathtt{x}$. *In string notation,* $10^{125}11 \cdot 0^{126}11 = 10^{119}10000010$.

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 product $a_{n-1}\mathtt{x}^n + a_{n-2}\mathtt{x}^{n-1} + a_1\mathtt{x}^2 + a_0\mathtt{x}$. Thus, if the first bit of a is 0, then $a \cdot \mathtt{x} = a{\ll}1$. If the first bit of a is 1 then we must add $\mathtt{x}^{128}$ to $a{\ll}1$. Since $\mathtt{x}^{128} + \mathtt{x}^7 + \mathtt{x}^2 + \mathtt{x} + 1 = 0$ we know that $\mathtt{x}^{128} = \mathtt{x}^7 + \mathtt{x}^2 + \mathtt{x} + 1$, so adding $\mathtt{x}^{128}$ means to xor by $0^{120}10000111$. In summary, when $n = 128$,

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

Example 3 *Let us again compute* $10^{125}11 \cdot 0^{126}11$. *Since the latter string is* $\mathtt{x} + 1$, *we should multiply the first string by* $\mathtt{x}$ *and then add it to (xor it with) the first string. As the first bit of* $10^{125}11$ *is* 1, *multiplying this point by* $\mathtt{x}$ *yields* $0^{125}110 \oplus 0^{120}10000111 = 0^{120}10000001$, *and xoring this with* $10^{125}11$ *gives a final answer of* $10^{119}10000010$, *as before.*

If $L \in \{0,1\}^n$ and $i \geq 0$, we write $L(i)$ as shorthand for $L \cdot \mathtt{x}^i$. We have an easy way to compute $L(1), L(2), \ldots, L(\mu)$-values, where μ is a small number. Namely, set $L(0) = L$ and compute $L(i) = L(i-1) \cdot \mathtt{x}$ for all $i \in [1..\mu]$.

If $a \neq 0$ is a point in $\{0,1\}^n$, we can divide a by $\mathtt{x}$, meaning that one multiplies a by the multiplicative inverse of $\mathtt{x}$ in the field: $a \cdot \mathtt{x}^{-1}$. It is easy to compute $a \cdot \mathtt{x}^{-1}$. To illustrate, again assume that $n = 128$. Then if the last bit of a is 0, then $a \cdot \mathtt{x}^{-1}$ is $a{\gg}1$. If the last bit of a is 1, then we must add (xor) to $a{\gg}1$ the value $\mathtt{x}^{-1}$. Since $\mathtt{x}^{128} = \mathtt{x}^7 + \mathtt{x}^2 + \mathtt{x} + 1$ we have $\mathtt{x}^{127} = \mathtt{x}^6 + \mathtt{x} + 1 + \mathtt{x}^{-1}$ and so $\mathtt{x}^{-1} = \mathtt{x}^{127} + \mathtt{x}^6 + \mathtt{x} + 1 = 10^{120}1000011$. In summary, for $n = 128$,

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

We point out that, for any $n = 128$, the value *huge* $= \mathtt{x}^{-1}$ will be an enormous number (when viewed as a number); in particular, *huge* starts with a 1 bit, so $2^{n-1} \leq$ *huge*. For the remainder of this submission, we will use *huge* as a synonym for $\mathtt{x}^{-1}$ whenever this seems to add to clarity. We will later assume that any messages $M = M[1] \cdots M[m]$ to be MACed has block length $m <$ *huge*, for otherwise our theorem statements assert a non-result. Thus for any message $M = M[1] \cdots M[m]$ to be MACed, each of $\gamma_1, \gamma_2, \ldots, \ldots, \gamma_m$ is different from *huge*.

Gray codes. For $\ell \geq 1$, 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 $\ell > 0$,

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

It is easy to see that γ is a Gray code. What is more, for $1 \leq i \leq 2^n - 1$, $\gamma_i = \gamma_{i-1} \oplus (0^{n-1}1{\ll}\mathsf{ntz}(i))$. This makes it easy to compute successive points.

Example 4 *The canonical Gray code with 2 points is* $\gamma^1 = \gamma_0^1 \; \gamma_1^1 = 0 \; 1$. *The canonical Gray code with 4 points is obtained by writing this once forward, then once backwards, prefixing each string in the first half by 0 and prefixing each string in the second half by 1: that is,* $\gamma^2 = \gamma_0^2 \; \gamma_1^2 \; \gamma_2^2 \; \gamma_3^2 = 00 \; 01 \; 11 \; 10 = 0 \; 1 \; 3 \; 2$. *Repeating the process, the canonical Gray code with 8 points*

is $\gamma^3 = \gamma^3_0\ \gamma^3_1\ \gamma^3_2\ \gamma^3_3\ \gamma^3_4\ \gamma^3_5\ \gamma^3_6\ \gamma^3_7 = 000\ 001\ 011\ 010\ 110\ 111\ 101\ 100 = 0\ 1\ 3\ 2\ 6\ 7\ 5\ 4$. *In PMAC we use the Gray code* $\gamma = \gamma^n$ *having* 2^n *points:* $\gamma = \gamma_0\ \gamma_1\ \gamma_2\ \gamma_3\ \cdots \gamma_{2^n-1} = 0\ 1\ 3\ 2\ 6\ 7\ 5\ 4\ 12\ \cdots\ 2^{n-1}$. *To calculate* γ_i *from* γ_{i-1}, *xor* γ_{i-1} *by* $0^{n-1}1{\ll}\mathsf{ntz}(i)$. *For example,* $\gamma_8 = 12$ *can be computed from* $\gamma_7 = 4$ *by xoring* 4 *with* $0^{n-1}1{\ll}3$.

We emphasize the following characteristics of the Gray-code values $\gamma_1, \gamma_2, \ldots, \gamma_{2^n-1}$. First, they are distinct and different from 0. Second, that $\gamma_1 = 1$. Third, that $\gamma_i \le 2i$.

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 $i \ge 2$, assume one has already produced $\gamma_{i-1} \cdot L$. Since $\gamma_i = \gamma_{i-1} \oplus (0^{n-1}1{\ll}\mathsf{ntz}(i))$ we know that

$$\begin{aligned}
\gamma_i \cdot L &= (\gamma_{i-1} \oplus (0^{n-1}1{\ll}\mathsf{ntz}(i))) \cdot L \\
&= (\gamma_{i-1} \cdot L) \oplus (0^{n-1}1{\ll}\mathsf{ntz}(i)) \cdot L \\
&= (\gamma_{i-1} \cdot L) \oplus (L \cdot \mathsf{x}^{\mathsf{ntz}(i)}) \\
&= (\gamma_{i-1} \cdot L) \oplus L(\mathsf{ntz}(i))
\end{aligned}$$

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 Specification

3.1 Definition of the Mode

Parameters. To use PMAC one must specify two parameters: a block cipher and a tag length.

- The **block cipher** E 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. Both are arbitrary, though we insist that $n \ge 64$, and we discourage $n < 128$.
- The **tag length** τ is an integer between 1 and n. By trivial means, the adversary will be able to forge a valid ciphertext with probability $2^{-\tau}$.

The popular block cipher to use with PMAC is likely to be AES, but any other block cipher is fine. As for the tag length, a suggested default of $\tau = 64$ is reasonable, though both shorter and longer tags are likely to be common. Note that tags of $\tau = 32$ bits have been standard for retail banking for many years, while tags of $\tau = 80$ bits are used in IPSec. Using a tag of more than 80 bits adds questionable security benefit, though it does entail extra bits being transmitted or stored.

With $E : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ and $\tau \in [1..n]$, we let $\mathrm{PMAC}[E,\tau]$ denote the PMAC mode of operation using block cipher E and tag length τ. This is a function from a key $K \in \mathcal{K}$ and a message $M \in \{0,1\}^*$ to a string in $\{0,1\}^\tau$.

Definition. Letting $E : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ and $\tau \in [1..n]$, the definition of the function $\mathrm{PMAC}[E,\tau]$ is given in Figure 1 and illustrated in Figure 2.

3.2 Conformance Criteria

An implementation of PMAC is said to conform to the specification if some specified subset $\mathrm{MsgSpace} \subseteq \{0,1\}^*$ of messages can be presented to be MACed, and any message M carrying a tag *Tag* is deemed invalid if $\mathrm{PMAC}_K(M) \ne \mathit{Tag}$. For example,

Algorithm $\mathrm{PMAC}_K\ (M)$

$L \leftarrow E_K(0^n)$
Partition M into $M[1] \cdots M[m]$
for $i \leftarrow 1$ **to** $m-1$ **do**
  $Y[i] \leftarrow E_K(M[i] \oplus \gamma_i \cdot L)$
$\Sigma \leftarrow Y[1] \oplus Y[2] \oplus \cdots \oplus Y[m-1] \oplus \mathsf{pad}(M[m])$
if $|M[m]| = n$ **then** $X[m] = \Sigma \oplus L \cdot \mathtt{x}^{-1}$
  else $X[m] \leftarrow \Sigma$
$Tag = E_K(X[m])$ [first τ bits]
return Tag

Figure 1: **Definition of PMAC**. Constants $\gamma_1, \gamma_2, \ldots$ and the meaning of the multiplication operator are defined in the text, but $\gamma_i \cdot L$ is simply $\gamma_{i-1} \cdot L \oplus L(\mathsf{ntz}(i))$ (where $L(j)$ is defined in the text and easily computed from L). We let $\mathsf{pad}(A) = A\,10^{n-|A|-1}$ if $|A| < n$ and $\mathsf{pad}(A) = A$ if $|A| = n$.

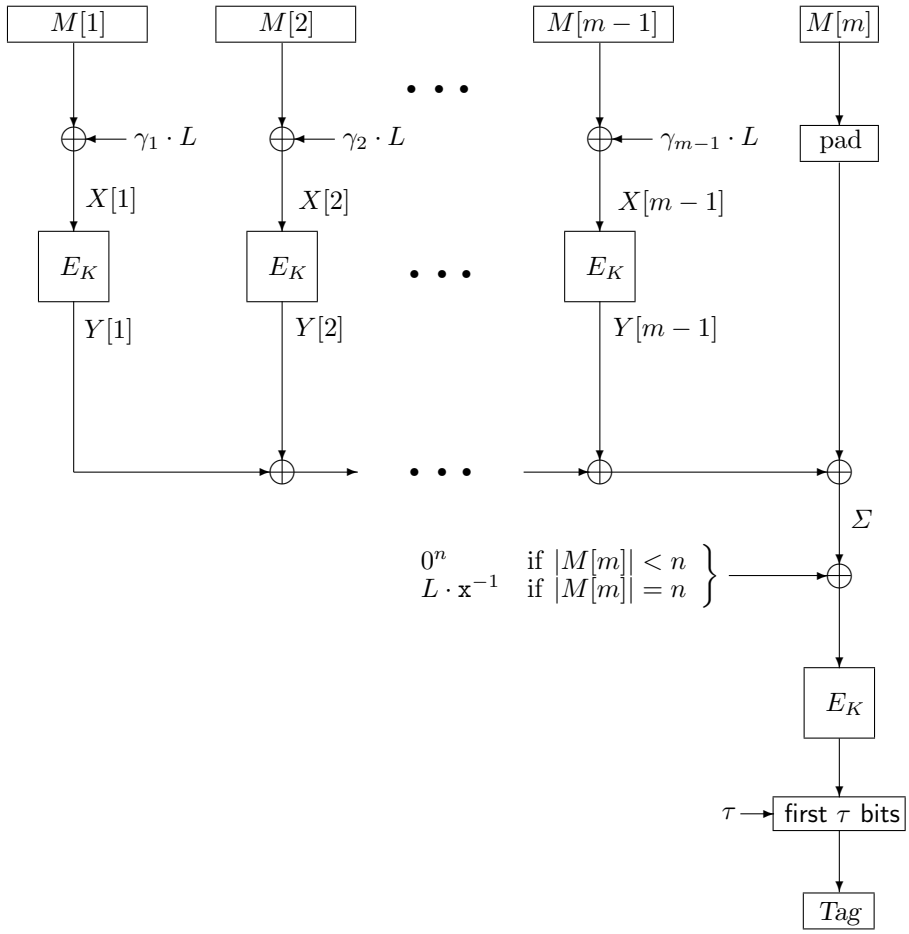

Figure 2: **Illustration of PMAC**. The message to MAC is M and the key is K. Message M is written as $M = M[1] \cdots M[m]$, where $m = \max\{1, \lceil |M|/n \rceil\}$ and $|M[1]| = |M[2]| = \cdots = |M[m-1]| = n$. Value $L = E_K(0^n)$ is derived from K.

- A conforming implementation might only be able to MAC byte strings, these strings limited to 2^{36} bytes.
- A message M might be deemed invalid for reasons which go beyond its presenting an incorrect tag. Replay detection would be the most typical reason.

3.3 An Equivalent Description

The following description of PMAC may help to clarify what a typical implementation might choose to do. In what follows, fix a block length n, block cipher $E : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$, and a tag length τ. A PMAC implementation may work as follows.

KEY GENERATION. Choose a random key $K \xleftarrow{R} \mathcal{K}$ for the block cipher. The key K is provided to both the party that generates MACs and the party that verifies them.

KEY SETUP. Once the key K is known, the following may be precomputed.

1. *Set up the block-cipher key.* 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.
2. *Precompute L.* Compute $L \leftarrow E_K(0^n)$.
3. *Precompute $L(i)$-values.* 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 $L(0) \leftarrow L$ and, for $i \in [1..\mu]$, compute $L(i) \leftarrow L(i-1) \cdot \mathtt{x}$ using a shift and a conditional xor, as described in Section 2. Compute $L(-1) \leftarrow L \cdot \mathtt{x}^{-1}$ using a shift and a conditional xor, as described in Section 2. Save the values $L(-1), L(0), L(1),\ L(2),\ \ldots,\ L(\mu)$ in a table. (Note: alternatively, if one wishes to save space, compute only the first few $L(i)$-values now, and compute any further ones only when needed, "on the fly.")

MAC GENERATION. To generate the MAC *Tag* for a message $M \in \{0,1\}^*$, do the following steps.

1. *Partition the message.* 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 $i \in [1..m-1]$.
2. *Initialize variables.* Let *Offset* $\leftarrow 0^n$. Let $\Sigma \leftarrow 0^n$.
3. *Encipher all blocks but the last one.* For $i \leftarrow 1$ to $m-1$, do the following:
   Let *Offset* $\leftarrow$ *Offset* $\oplus L(\mathsf{ntz}(i))$.
   Let $Y[i] \leftarrow E_K(M[i] \oplus \mathit{Offset})$.
   Let $\Sigma \leftarrow \Sigma \oplus Y[i]$.
4. *Compute the MAC.*
   Let $\Sigma \leftarrow \Sigma \oplus \mathsf{pad}(M[m])$.
   If $|M[m]| = n$ then let $Y[m] \leftarrow E_K(\Sigma \oplus L(-1))$ else let $Y[m] \leftarrow E_K(\Sigma)$.
   Let *Tag* be the first τ bits of $Y[m]$.
   Return *Tag* as the computed MAC.

MAC VERIFICATION. To test if (M, Tag') is authentic, do the following.

1. *Re-MAC the message.* Generate the MAC *Tag* for the message M using the MAC generation procedure just described.
2. *Compare the presented and the re-computed MACs.* 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.

Security Function	**Message Authentication Code**. The computed MAC would normally be used to provide message authenticity. But, more generally, PMAC is a **pseudorandom function** (PRF) having variable input-length ($\{0,1\}^*$) and fixed output-length ($\{0,1\}^\tau$). Besides their use as MACs, PRFs can be used, for example, for key separation and within entity-authentication and key-distribution protocols.
Error Propagation	**Not applicable**. (However, any accidental modification to (M, Tag) will be detected during MAC verification with probability about $1/2^\tau$.)
Synchronization	**Not applicable**.
Parallelizability	**Fully parallelizable**. All block-cipher invocations (except the last one) may be computed at the same time.
Keying Material	**One block-cipher key**. One needs a single key, K, which keys all invocations of the underlying block cipher.
Ctr/IV/Nonce Requirements	**None.** No counter/IV/nonce is used.
Memory Requirements	**Very modest**. About $6n$ bits beyond the key are sufficient for internal calculations. Implementations may choose whether or not to save $L(i)$-values, offering some tradeoff between memory and simplicity/speed.
Pre-processing Capability	**Limited.** During key-setup the string L would typically be pre-computed (one block cipher call), as would a few $L(i)$-values. Additional pre-computation is not possible.
Message-Length Requirements	**Any bit string allowed.** Any string $M \in \{0,1\}^*$ may be MACed, including the empty string and strings which are not an integral number of bytes. The length of the string need not be known in advance.
Ciphertext Expansion	**Not applicable**. (But the MAC itself is of minimal length: τ bits are used to obtain a forging probability of approximately $1/2^\tau$.)
Other Characteristics	**Efficiency**: PMAC uses $\lceil \lvert M \rvert / n \rceil$ block-cipher calls when $M \neq \varepsilon$ (uses one block-cipher call for $M = \varepsilon$). Overhead beyond block-cipher calls is low. **Endian neutral**: Can be implemented equally efficiently on big-endian and little-endian machines (assuming the underlying block cipher has this property). **No decryption.** PMAC uses only the forward direction of the block cipher. **Incrementality.** PMAC is incremental, in the sense of [3], with respect to replacement, truncation, and concatenation. **Parsimonious PRF**. This property lets one easily construct from PMAC a parallelizable block cipher which acts on any message $M \in \{0,1\}^*$ of at least n bits. **Provable security**: PMAC provably meets its goals, assuming the underlying block cipher meets a standard cryptographic assumption (security in the sense of a pseudorandom permutation).

Figure 3: **Summary of properties of PMAC.** Further discussion of these and other properties is given in Section 4.1.

4 Discussion

4.1 Properties

PMAC has been designed to have a variety of desirable properties. These properties are summarized in Figure 3. We now expand on some of the points referenced in that table.

Security Function. PMAC is a (variable-length input, fixed-length output) pseudorandom function (PRF): as long as the underlying block cipher E is secure, an adversary will be unable to distinguish $\mathrm{PMAC}_K(\cdot)$, for a random but 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 [14], that a pseudorandom function is necessarily a good MAC. Of course the converse is not true, and PMAC, being a good PRF, has uses which go beyond its being a good MAC.

Parallelizability. In settings where there is adequate opportunity for parallelism, PMAC will be faster than the CBC MAC or HMAC, which are inherently non-parallelizable. Parallelizability is becoming important for obtaining good performance from both high-speed hardware and commodity processors. In the former case, one may want to authenticate messages at speeds in excess of 10 Gbits/second—an impossible task for the CBC MAC or HMAC (with today's technology). In the latter case, there is an architectural trend towards highly-pipelined machines with multiple instruction pipes and lots of registers. Optimally exploiting such features necessitates algorithms with plenty to do in parallel.

Keying Material. Conceptually the key is (K, L), but L is defined from the underlying key K, and then key K is still used. Normally such "lazy key-derivation" would get one in trouble. For PMAC we prove that it does not. Avoiding multiple block-cipher keys is desirable for saving on memory and key-setup time.

Ctr/IV/Nonce Requirements. Conventional MACs have always been deterministic. Maintaining this characteristic results in shorter MACs and a scheme more robust against usage errors.

Message-Length Requirements. 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. An incremental interface (as is popular for cryptographic hash functions) would be used to support this functionality.

Efficiency. Shaving off a few block-cipher calls may not seem important. But often one is dealing with short messages; for example, roughly a third of the messages on the Internet backbone are 43 bytes. If one is MACing messages of such short lengths, one should be careful about extra computational work since, by percentage, the inefficiencies can be large.

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

Incrementality. The concept of incrementality for a cryptographic primitive was introduced by [14]. For a MAC, the idea is that, having already computed the MAC *Tag* of some (possibly long) message M, if M is modified into some similar message M', the time to compute the MAC *Tag*$'$ for the new message M' should be proportional to the *amount of change* that M underwent when being modified to M' (as opposed to just computing the MAC of M' from scratch, which would typically take $\Theta(|M'|)$ time.) Assuming that $\tau = n$ (or that one retains a constant amount of extra information), PMAC is incremental with respect to three common operations for changing M to M', namely, `append`, `truncate`, and `replace`. These operations are defined as follows: $\texttt{append}(M, x) = M \parallel x$; $\texttt{truncate}(M, \mathit{ndrop}) = M\,[\text{first } |M| - \mathit{ndrop} \text{ bits}]$, where $|M| \geq \mathit{ndrop}$; and $\texttt{replace}(M, \mathit{posn}, x) = M\,[\text{first } \mathit{posn} - 1 \text{ bits}] \parallel x \parallel M[\text{last } |M| - \mathit{posn} - |x| + 1 \text{ bits}]$, where $|M| \geq \mathit{posn} + |x| - 1$). For these three operations it is easy to see how to update the MAC of M in time proportional to $|x|$, *ndrop*, or $|x|$, respectively.

Incrementality is useful when, for example, one wishes to continually maintain the MAC of a large file which is being edited, or the MAC of a file system. Various tree-based schemes offer solutions with logarithmic space overhead. For PMAC, there is zero or constant overhead.

Parsimonious PRF. Partition M into $M = M[1] \cdots M[m-1]M[m]$ and assume $|M| \geq n$. Assume a tag length of the full $\tau = n$ bits. We make the following observation: for any $i \in [1..m]$ such that $M[i]$ is a full block (that is, $|M[i]| = n$), there is a simple algorithm to recover $M[i]$ given the key K, the message $M' = M[1] \ldots M[i-1]\ M[i+1] \cdots M[m]$ which omits block $M[i]$, and the MAC $\mathit{Tag} = \mathrm{PMAC}_K(M)$. This property of a PRF was identified in [7], where a PRF having this property was said to be *parsimonious*. As shown in [7], a parsimonious PRF can be combined with a block cipher E to yield a length-preserving pseudorandom permutation (that is, a "variable-input-length block cipher") that acts on messages of any number of bits ℓ greater than or equal to n. In particular, given $K1, K2 \in \mathcal{K}$, one can encipher M under key $K = K1 \parallel K2$ by first setting $\mathrm{IV} = \mathrm{PMAC}_{K1}(M)$ and then encrypting M using CTR-mode using key $K2$ and an initial counter value of IV. This yields a ciphertext $\mathcal{C} = \mathrm{IV} \parallel C[1] \cdots C[m-1]C[m]$. One can use $\mathcal{C}$ directly, as a deterministic (but not length-preserving) encryption of M, or one can drop n of the bits from $\mathcal{C}$, the enciphered string being $C = \mathrm{IV} \parallel C[1] \ldots C[m-2]\ C[m]$, say. In this way one has constructed a length-preserving pseudorandom permutation (PRP) that is fully parallelizable and that uses a number of block-cipher calls which is roughly twice that of CBC encryption.

Provable Security. In recent years provable security has become a popular goal for practical protocols. This is for good reason; demonstrating provable security is the best way to gain assurance that a cryptographic scheme does what it is supposed to do. For a scheme which enjoys provable security, one does not need to consider attacks, since successful ones imply successful attacks on some simpler object (e.g., the algorithm of the AES). Provable security represents a major departure from iterated, attack-directed design.

Fix a block cipher E and a tag length τ. When we say that PMAC is provably secure we are asserting the existence of a particular theorem (namely, Theorem 1, or one of its corollaries). The theorem shows that if an adversary A could do a good job at distinguishing $\mathrm{PMAC}_K(\cdot)$, for a random but hidden key K, from a random function $\rho(\cdot)$, where $\rho : \{0,1\}^* \to \{0,1\}^\tau$, then there would be an adversary B, about as efficient as A, that does a good job at distinguishing block cipher $E_K(\cdot)$, for a random key K, from a random permutation $\pi(\cdot)$, where $\pi : \{0,1\}^n \to \{0,1\}^n$. A theorem of this sort is called a *reduction*. In cryptography, provable security means giving reductions (along with the associated definitions).

Provable security begins with Goldwasser and Micali [15]. The style of provable security which we use here—where the primitive is a block cipher, the scheme is a usage mode, and the analysis

is concrete (no asymptotics)—is the approach of Bellare and Rogaway [2, 4, 5].

It is not really enough to know that there is a provable-security result for the scheme in question: one should also understand the definitions and the bounds. We have already sketched the definitions. When we speak of the bounds we are addressing "how effective is the adversary B in terms of the efficacy of adversary A" (where A and B are the adversaries above). For PMAC, the bounds can be summarized as follows: the maximal *advantage* an adversary can get in attacking the constructed PRF is limited to about $\sigma^2/2^n$ more than what a similarly powerful adversary can get in attacking the underlying block cipher. The advantage is a real number, between 0 and 1, with 0 meaning that the adversary is doing terribly and 1 meaning that the adversary is doing great. Here σ is the total number of messages the adversary sees the MACs of. The conclusion is that one is safe using PMAC as long as E is a good block cipher and σ is small compared to $2^{n/2}$. This is the same security degradation one observes for the CBC MAC [5, 22]. This kind of security loss was the main motivation for choosing an AES block length of $n = 128$ bits, and it is the reason that we discourage the use of PMAC with $n < 128$.

4.2 Design Rationale

Avoiding state and randomness. A message authentication code is stateful if the authenticating party maintains state, typically a counter, across MAC-generation invocations. It is probabilistic if the authenticating party uses random bits to generate a MAC. (In both cases, MAC-verification is stateless and deterministic.) As in [4], construction of a stateful message authentication code would have facilitated obtaining a better concrete security bound than PMAC delivers. A randomized scheme would also have had a better bound. But PMAC anticipates the use of a block cipher with a block length of $n \geq 128$, and so the bound that we get—the customary $\Theta(\sigma^2 2^{-n})$ bound—is quite acceptable. By insisting on a stateless and deterministic scheme one saves bits in the MAC compared to a probabilistic or stateful scheme, one eliminates the sender's need to maintain state across MAC-generation invocations, and one largely eliminates concern about usage errors, since one does not have to worry about what happens if the sender reuses a nonce or provides a non-random value where a random value is called for.

Not fixing the tag length. The number of bits that are necessary and appropriate for the tag length vary according to the application. In a context where the adversary obtains something quite valuable from a single successfully forgery, one may wish to choose a tag length of 80 bits or more. In contexts such as authenticating a video stream, where an adversary would have to forge a signficant fraction of the frames even to have a noticeable effect on the image, an 8-bit tag may be appropriate. With no universally appropriate value to choose, it is best to leave this parameter unspecified.

We comment that short tags seem to be more appropriate for PMAC than for some other MAC, particularly Carter-Wegman MACs [12, 26]. Many Carter-Wegman MACs have the property that if you can forge one message with probability δ, than you can forge an arbitrary set of (all correct) messages with probability δ. This does not appear to be true for PMAC.

Avoiding mod 2^n addition. Our earlier designs included a scheme based on modular 2^n addition ("addition" for the remainder of this paragraph) [24]. This is an interesting approach due to Gligor and Donescu [13]. Compared to our GF(2^n)-based approach ("xor" for the remainder of this paragraph), an addition-based scheme is quicker to understand a specification for, and may be easier to implement. But the use of addition (where $n \geq 128$) has signficant disadvantages:

- The bit-asymmetry of the addition operator implies that the resulting scheme will have a bias

towards big-endian architectures or little-endian architectures; there will be no way to achieve an endian-neutral scheme. The AES algorithm was constructed to be endian-neutral. We did not want to lose this nice attribute with our mode of operation.

- Modular addition of n-bit words is particularly unpleasant when programming in a high-level language, where one does not have access to the underlying add-with-carry instruction.
- Modular addition of n-bit words is not parallelizable. As a consequence, dedicated hardware will perform this operation more slowly than xor, and, correspondingly, modern processors can xor two n-bit quantities faster than they can add them.
- The concrete security bound is worse with an addition-based scheme: the degradation in the bound appears to be $\Theta(\lg m^*)$, where m^* is the maximal message length.
- A correct proof for the addition scheme is substantially more complex than a correct proof for the xor-scheme.

For all of these reasons, we eventually decided to reject the addition-based approach in favor of the xor-based one.

4.3 Limitations

We note the following limitations. None of these points are specific to PMAC—they apply to any MAC—but they are still important enough to single out.

- As with all modes of operation, the user should be careful not to use the MAC key K for any other purpose. Standard key-separation techniques should be used to derive a multiplicity of keys when keys are needed for a multiplicity of purposes. The key K itself must not be used to derive additional keys.
- A MAC does not, by itself, provide for replay detection. When replay detection is desired, it should be added, using well-known techniques, by a higher-level, protocol.
- One should be careful in combining an encryption scheme and a MAC: the keys must be distinct, and it is generally preferable to MAC the ciphertext than to encrypt the MAC (and when one wants privacy one should not transmit, in the clear, the MAC of a plaintext). See [6] for an analysis.
- The utility of a MAC is eliminated if one encrypts using an authenticated-encryption scheme, such as [13, 19, 25].

4.4 Related Work

The PMAC algorithm springs from ideas in [4], [9], and [13]. The lineage of ideas is from [4], to [9] and [13], and on to PMAC. Let us first sketch representative constructions from [4, 9, 13].

The XOR MAC. Bellare, Guérin and Rogaway defined a fully parallelizable MAC they called the XOR MAC [4]. In this MAC the message M is divided into pieces $M[1] \cdots M[\ell]$ of length less than the block length. For concreteness, think of each $M[i]$ as having 64 bits and the block length as having $n = 128$ bits. Each $M[i]$ is preceded by the number i, encoded in the usual way as a 64-bit string $\underline{i}$, and E_K is applied to $\underline{i} \parallel M[i]$. Now xor together all ℓ results to get an n-bit block $\Sigma = E_K(\underline{1} \parallel M[1]) \oplus \cdots \oplus E_K(\underline{\ell} \parallel M[\ell])$. Block cipher E_K is applied to one more block, $1 \parallel \mathrm{IV}$, where IV is a 127-bit counter or random value. This gives the ciphertext block $P = E_K(1 \parallel \mathrm{IV})$. The MAC is $(\mathrm{IV}, \Sigma \oplus P)$.

The XOR MAC requires $\ell + 1 \approx 2m + 1$ block-cipher invocations to authenticate a message of m n-bit blocks. So one gets parallelizability at the cost of about a factor of two in serial speed. One also pays in the need for randomness or state, and in the length of the MAC (which is longer

in order to communicate this randomness or state). These two limitations—the use of randomness or state and reduced serial efficiency—are overcome individually in [9] and [13], respectively.

Deterministic variants. Bernstein [9] suggested the following variant of the XOR MAC. He starts not with a block cipher but with a pseudorandom function F where, to illustrate, let us say that F_K maps 384 bits to 256 bits. In such a case, break the message M into 256-bit blocks $M[1], \ldots, M[\ell]$ and apply F_K to each of $\underline{1} \parallel M[1]$, $\underline{2} \parallel M[2]$, ..., $\underline{\ell} \parallel M[\ell]$, where $\underline{i}$ now means the 128-bit string corresponding to the number i. Now xor the resulting ciphertext blocks to get a value $\Sigma = F_K(\underline{1} \parallel M[1]) \oplus \cdots \oplus F_K(\underline{\ell} \parallel M[\ell])$. Then apply F_K to $\underline{0} \parallel \Sigma$ to give the MAC $Tag = F_K(\underline{0} \parallel \Sigma)$.

An alternative to $\underline{i} \parallel \cdot$ annotation. Gligor and Donescu [13] suggested the following approach, which they called the XECB-MAC. Let $E : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ be a block cipher and let $K \in \mathcal{K}$. Given a counter ctr, let $R = E_K(\text{ctr})$. Let $M = M[1] \cdots M[m]$ be the string we want to MAC, partitioned into n-bit words. Then encipher the following m strings using E_K: $M[1]+R$, $M[2]+2R$, ..., $M[m]+mR$. The arithmetic is done modulo 2^n. Now xor together these m ciphertexts to get a value $\Sigma = E_K(M[1]+R) \oplus \cdots \oplus E_K(M[m]+mR)$. The MAC is ctr together with $\Sigma \oplus E_K(\text{ctr})$. The advantage of this approach is that one does not "waste bits" for $\underline{i} \parallel$ -encoding; instead, symmetry has been broken by different means: by adding to the ith block of the message the ith multiple of a number unknown to the adversary.

Further afield. A completely different way for improving parallelizability is to generically construct a more parallelizable MAC from an arbitrary one. For example, one could break the message $M[1] \cdots M[2m]$ into pieces $M' = M[1]M[3]M[5] \cdots M[2m-1]$ and $M'' = M[2]M[4]M[6] \cdots M[2m]$, and separately compute $\text{MAC}_K(M') \parallel \text{MAC}_K(M'')$, and then MAC these MACs, under a separate key. But such an approach requires one to anticipate the maximal amount of parallelism that one aims to extract (the example here shows how to get a factor of two). We were not interested in such schemes; we wanted a MAC that was fully parallelizable.

Another approach for making a parallelizable MAC is the Carter-Wegman paradigm [12, 26], as in [11, 17, 20], making sure to select a universal hash-function family that is parallelizable. In fact, most universal hash-functions that have been suggested are parallelizable. This is perfectly workable, but fast constructions for universal hash-functions have proven to be complex to specify or to implement well [9, 11].

The approach. The provable-security treatment of block-cipher based MACs begins with Bellare, Kilian, and Rogaway [5]. Petrank and Rackoff [22] were the first to rigorously address issues of length variability. Black and Rogaway [10] were the first to concern themselves with optimizing efficiency on arbitrary bit strings, retaining provable security.

4.5 Design History

The initial version of PMAC, described in [24], came in three "forms"—one based on mod 2^n addition, one based on GF(2^n) addition, and one based on mod p addition. However, we eventually settled on the GF(2^n) scheme, for the reasons discussed in Section 4.2. What is described in the current submission differs from the GF(2^n)-scheme in [24] only in minor ways.

This project was carried out in parallel with the design of an authenticated-encryption scheme, OCB [25]. We have attempted to give these two algorithms a similar flavor. However, the authenticated-encryption goal turns out to be much more complex than the message-authentication goal, and the design of OCB was much harder than the design of PMAC.

5 Theorems

This section gives our security results on PMAC. The proofs of the two lemmas used are deferred to Appendix A. We begin with the requisite definitions, which are standard; see, for example, [5].

5.1 Security Definitions

A block cipher is a function $E : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ where $\mathcal{K}$ is a finite set of strings and each $E_K(\cdot) = E(K,\cdot)$ is a permutation on $\{0,1\}^n$. Let $\mathrm{Perm}(n)$ 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 string. Let A be an adversary (a probabilistic algorithm) with access to an oracle, and suppose that A always outputs a bit. Define

$$\mathbf{Adv}_E^{\mathrm{prp}}(A) \quad = \quad \Pr[K \xleftarrow{R} \mathcal{K} : \ A^{E_K(\cdot)} = 1] - \Pr[\pi \xleftarrow{R} \mathrm{Perm}(n) : \ A^{\pi(\cdot)} = 1]$$

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 $\mathrm{Perm}(n)$.

A function family from n-bits to n-bits is a map $F : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ where $\mathcal{K}$ is a finite set of strings. 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 by imagining that each function in $\mathrm{Rand}(n)$ is named by a unique string. Define

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

A function family from $\{0,1\}^*$ to $\{0,1\}^\tau$ is a map $f : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^\tau$ 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

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

5.2 Theorem Statements

We give the following information-theoretic bound on the security of PMAC.

Theorem 1 [$\mathrm{PMAC}[\mathrm{Perm}(n),\tau] \approx \mathrm{Rand}(*,\tau)$] *Fix* PMAC *parameters n and τ. Let A be an adversary with an oracle. Suppose that A asks its oracle q queries, these queries having aggregate length of σ blocks. Let $\bar{\sigma} = \sigma + 1$. Then*

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

In the theorem statement, and from now on, the aggregate length of messages $M_1, \ldots, M_q$ asked by A of its oracle 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. One gets the following. Fix PMAC parameters n and τ, and a block cipher $E : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$. Let A be an adversary with an oracle, and suppose that A asks queries of aggregate length of σ blocks, and then outputs a bit. Let $\bar{\sigma} = \sigma + 1$. Then there is an adversary B for attacking block cipher E that

achieves advantage $\mathbf{Adv}_E^{\mathrm{prp}}(B) \geq \mathbf{Adv}_{\mathrm{PMAC}[E,\tau]}^{\mathrm{prf}}(A) - \bar{\sigma}^2/2^{n-1}$. Adversary B asks at most $\bar{\sigma}$ oracle queries and has a running time which is equal to A's running time plus the time to compute E on $\bar{\sigma}$ points, plus additional time which is $cn\bar{\sigma}$, where the constant c 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 [5, 16]. Here is what the theorem would say. First a definition. Given a PRF $F : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^\tau$, let $\mathbf{Adv}_F^{\mathrm{mac}}(A)$ denote the probability that A, given an oracle $F_K(\cdot)$ for a random key $K \in \mathcal{K}$, outputs a pair (M, Tag) where $F_K(M) = Tag$ and A never asked its oracle the message M. Now fix PMAC parameters n and τ, and a block cipher $E : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$. Let A be an adversary with an oracle. Suppose that the aggregate block length of A's queries, plus the block length of A's forgery attempt M when A outputs (M, Tag), is σ. Let $\bar{\sigma} = \sigma + 1$. Then there is an adversary B for attacking block cipher E that achieves advantage $\mathbf{Adv}_E^{\mathrm{prp}}(B) \geq \mathbf{Adv}_{\mathrm{PMAC}[E,\tau]}^{\mathrm{mac}}(A) - \bar{\sigma}^2/2^{n-1} - 1/2^\tau$. Adversary B asks at most $\bar{\sigma}$ oracle queries and has a running time which is equal to A's running time plus the time to compute E on $\bar{\sigma}$ points, plus additional time which is $cn\bar{\sigma}$, where the constant c depends only on details of the model of computation.

5.3 Structure of the Proof

The proof of Theorem 1 combines two lemmas. The first lemma, the *structure lemma*, measures the pseudorandomness of PMAC in terms of two other functions: the M-collision probability, denoted $\mathrm{Mcoll}_n(\cdot)$, and the MM-collision probability, denoted $\mathrm{MMcoll}_n(\cdot,\cdot)$. Informally, $\mathrm{Mcoll}_n(m)$ measures the probability of a collision in $X[i]$-values for a single message M having m blocks. By definition, this will include collisions with 0^n. Similarly, $\mathrm{MMcoll}_n(m, \bar{m})$ measures the probability of a nontrivial collision (defined below) in $X[i]$ and $\bar{X}[j]$ values across two messages, M and $\bar{M}$, having m and $\bar{m}$ blocks. In defining these values, certain simplifications will be made compared to the "true" processing of messages under $\mathrm{PMAC}[\mathrm{Perm}(n), \tau]$. 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 messages M and $\bar{M}$, and partition them into $M[1] \ldots M[m]$ and $\bar{M}[1] \ldots \bar{M}[\bar{m}]$. Consider the experiment of Figure 4. When M is a string, let $\mathrm{Mcoll}_n(M)$ denote the probablity that *Mcoll* gets set to `true` in line 15 when the program of Figure 4 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 probablity that *MMcoll* gets set to `true` when the program of Figure 4 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.

Lemma 1 [**Structure lemma**] *Fix n and τ. Let A be an adversary who asks q queries, these having an aggregate length of σ blocks. Let $\mathrm{Mcoll}_n(\cdot)$ and $\mathrm{MMcoll}_n(\cdot,\cdot)$ denote the M- and MM-collision probablities. Then*

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

The proof of this lemma is in Appendix A.1.

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

```
10   L ←R {0,1}^n
11   for i ← 1 to m − 1 do { X[i] ← M[i] ⊕ γ_i · L;   Y[i] ←R {0,1}^n }
12   Σ ← Y[1] ⊕ ··· ⊕ Y[m − 1] ⊕ pad(M[m])
13   if |M[m]| = n then X[m] ← Σ ⊕ huge · L else X[m] ← Σ
14   𝒳 ← {X[1], ..., X[m]}
15   if there is a repetition in {0^n} ∪ 𝒳 then Mcoll ← true

16   Equal ← {i ∈ [1.. min{m, m̄} − 1] :  M[i] = M̄[i]}
17   Unequal ← [1..m̄] \ Equal
18   for i ← 1 to m̄ − 1 do
19       if i ∈ Equal then { X̄[i] ← X[i];   Ȳ[i] ← Y[i] }
20       if i ∈ Unequal then { X̄[i] ← M̄[i] ⊕ γ_i · L;   Ȳ[i] ←R {0,1}^n }
21   Σ̄ ← Ȳ[1] ⊕ ··· ⊕ Ȳ[m̄ − 1] ⊕ pad(M̄[m̄])
22   if |M̄[m̄]| = n then X̄[m̄] ← Σ̄ ⊕ huge · L else X̄[m̄] ← Σ̄
23   𝒳̄ ← {X̄[i] : i ∈ Unequal}
24   if 𝒳 ∩ 𝒳̄ ≠ ∅ then MMcoll ← true
```

Figure 4: **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 $\mathtt{x}^{-1}$.

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 maximal advantage of an adversary who attacks the pseudorandomness of PMAC: namely, 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 collisions rarely occur. Its proof is given in Appendix A.2.

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. The pseudorandomness of PMAC follows by combining Lemmas 1 and 2. Namely,

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

where (1) 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. This completes the proof of Theorem 1.

6 Performance

Abstract accounting. PMAC uses $\lceil |M|/n \rceil$ block-cipher invocations for any nonempty message M. (The empty string takes one block-cipher invocation). We compare with the CBC MAC:

- The "basic" CBC MAC, which assumes that the message is a nonzero multiple of the block length and which is only secure when all messages to be MACed are of one fixed length, uses the same number of block cipher calls: $|M|/n$.
- The version of the CBC MAC described in [10], which removes the two restrictions just mentioned, uses the same number of calls as PMAC, $\lceil |M|/n \rceil$.
- Obligatory padding (to support short-final-block messages) and standard methods to process the final block (double or triple encryption, to achieve security across variable-length messages) can raise the number of block-cipher calls to as much as $\lceil |M+1|/n \rceil + 2$.

Thus PMAC saves between 0 and 3 block-cipher calls compared to the various versions of the CBC MAC.

As with any mode, there is further overhead beyond the block-cipher calls. Per block, this overhead is about three n-bit xor operations plus associated logic. The work for this associated logic will vary according to whether or not one precomputed $L(i)$-values, whether or not there is a native $\mathsf{ntz}()$ instruction available, and on other factors.

Though some of the needed $L(i)$-values are likely to be pre-computed, calculating these values "on the fly" is not too 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. So half the time we use L itself; a quarter of the time we use $2\cdot L$; one eighth of the time we use $4\cdot L$; and so forth. Thus the expected number of $a\cdot \mathtt{x}$-operations to compute an offset is at most $\sum_{i=1}^{\infty} i/2^{i+1} = 1$. Each $a\cdot \mathtt{x}$ instruction requires

Algorithm	16 B	128 B	2 KB
PMAC	**22.1**	**18.7**	**18.4**
CBC MAC	18.9	17.4	17.1

Figure 5: **Performance results.** Numbers are in cycles per byte, on a Pentium 3, for three message lengths, the code written in assembly. The underlying block cipher is AES-128. CBC MAC refers to the "basic" CBC MAC—no padding is performed and nothing extra is done to get security across messages of varying lengths.

an n-bit xor and a conditional 32-bit xor. Said differently, for any $m > 0$, the total number of $a \cdot \mathtt{x}$ operations needed to compute $\gamma_1 \cdot L$, $\gamma_2 \cdot L$, $\ldots$, $\gamma_m \cdot L$ is $\sum_{i=1}^{m} \mathsf{ntz}(i)$, which is less than m. The above assumed that one does not retain or precompute any $L(i)$ value beyond $L = L(0)$. Suppose that one retains a few values: $L(0), L(1), L(2), L(3)$. Computing and storing these three additional values is less overhead than computing L itself, which required an application of E_K. But now the desired multiple of L has already been computed $1/2 + 1/4 + 1/8 + 1/16 \approx 94\%$ of the time. When it has not been pre-computed it must be calculated, starting from $8 \cdot L$, so the amortized number of doubling steps has thus been reduced from 1 to $\sum_{i=1}^{\infty} = i/2^{i+4} = 0.125$.

Experimental results. Our colleague Ted Krovetz implemented PMAC using AES-128 as the underlying block cipher. He compared its performance in an entirely sequential setting to that of the CBC MAC. By the CBC MAC we mean the "basic" CBC MAC—nothing extra 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 assembly. The OS was Windows 2000 sp1 and the compiler was Visual C++ 6.0 sp4. All data fit into L1 cache.

Note that the basic CBC MAC needs to be modified to correctly handle length-variability and the possibility of short final blocks, and doing this is often done in a way that entails additional block-cipher calls. Ignoring these factors tends to make the performance comparison conservative.

Disregarding the one-block message in Figure 6, we see that PMAC adds about 8% overhead compared to the basic CBC MAC. We emphasize that this is for an entirely serial execution environment with a limited number of registers. In an environment with plenty of registers and multiple instruction pipes, PMAC, properly implemented, will of course be faster than the CBC MAC.

7 Intellectual Property Statement and Disclosures

Patent applications covering the ideas of this proposal were filed (Rogaway as inventor) on 13 September 2000, 12 October 2000, and 9 February 2001.

The inventor hereby releases IP rights covering PMAC for all non-commercial, non-governmental applications. For commercial applications, the inventor will license PMAC under a non-exclusive license, on a non-discriminatory basis, based on reasonable terms and conditions.

IBM has a patent covering the XOR MAC of [4]; this is US Patent #5,673,318 (September 30, 1997). Though the submitter is in no position to render a legal opinion, it appears to him that the claims of US Patent #5,673,318 do not read upon the use of PMAC. Virgil Gligor indicates that he has made patent filings on 31 January 2000, 31 March 2000, and 24 August 2000. We know no more details.

The IP status specified in this submission will be updated when more is known.

Acknowledgments

Thanks to Virgil Gligor, who described [13] to Rogaway at CRYPTO '00. This conversation inspired the development of PMAC. Special thanks to Ted Krovetz, who wrote reference code, prepared test vectors, collected performance data, and gave valuable interaction. Thanks to Michael Amling for a careful proofreading.

The first NIST modes-of-operation workshop, held in October 2000, was the motivation for putting together our first draft of PMAC [24]. The present writeup was prepared in response to a later call for proposals by NIST. We are pleased with NIST's leadership in promoting the development of modern modes of operation, and thank Elaine Barker, Morris Dworkin, and Jim Foti for their work in this connection.

This submission was prepared while Rogaway was on leave of absence from UC Davis, visiting the Department of Computer Science, Faculty of Science, Chiang Mai University. No external funding was used for this particular piece of research, but the submitter gratefully acknowledges a recent gift by Cisco Systems.

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. `http://www-cse.ucsd.edu/users/mihir/`

[2] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption: Analysis of the DES modes of operation. *Proceedings of 38th Annual Symposium on Foundations of Computer Science* (FOCS 97), IEEE, 1997. `http://www.cs.ucdavis.edu/~rogaway/`

[3] 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. `http://www.cs.ucdavis.edu/~rogaway/`

[4] 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, D. Coppersmith, Ed., pp. 15–28, 1995. `http://www.cs.ucdavis.edu/~rogaway/`

[5] 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.) `http://www.cs.ucdavis.edu/~rogaway/`

[6] M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *Advances in Cryptology – ASIACRYPT '00*. Lecture Notes in Computer Science, T. Okamoto., ed., Springer-Verlag, 2000. `http://www-cse.ucsd.edu/users/mihir/`

[7] 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, T. Okamoto., ed., Springer-Verlag, 2000. `http://www.cs.ucdavis.edu/~rogaway/`

[8] 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.

[9] D. Bernstein. How to stretch random functions: the security of protected counter sums. *Journal of Cryptology*, 12(3): pp. 185–192 (1999). `http://cr.yp.to/djb.html`

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

[11] 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. `http://www.cs.ucdavis.edu/~rogaway/`

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

[13] V. Gligor and P. Donescu. Fast encryption and authentication: XCBC encryption and XECB authentication modes. Manuscript, August 18, 2000. Available at `http://www.eng.umd.edu/~gligor/`. There is also an October 2000 version on `http://csrc.nist.gov/encryption/aes/modes/`

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

[15] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, vol. 28, April 1984, pp. 270–299.

[16] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal of Computing*, vol. 17, no. 2, April 1988, pp. 281–308.

[17] S. Halevi and H. Krawczyk. MMH: Software message authentication in the Gbit/second rates. *Proceedings of the 4th Workshop on Fast Software Encryption*, Lecture Notes in Computer Science, vol. 1267, Springer-Verlag, pp. 172–189, 1997. `http://www.research.ibm.com/people/s/shaih/`

[18] Iso/Iec 9797-1. Information technology – Security techniques – Data integrity mechanism using a cryptographic check function employing a block cipher algorithm. Second edition. International Organization for Standards (ISO), Geneva, Switzerland, 1999.

[19] C. Jutla. Encryption modes with almost free message integrity. Manuscript. August 1, 2000. `http://eprint.iacr.org/` (reference number 2000/039). Proceedings version to appear in Eurocrypt 2001.

[20] 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.

[21] M. Luby and C. Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM J. Computation*, vol. 17, no. 2, April 1988.

[22] E. Petrank and C. Rackoff. CBC MAC for real-time data sources. *J. of Cryptology*, 13(3), pp. 315–338, Nov 2000. `http://www.cs.technion.ac.il/~erez/publications.html`. Earlier version as 1997/010 in the Cryptology ePrint archive, `http://eprint.iacr.org/`

[23] 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.

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** $Y_r[i] \xleftarrow{R} \{0,1\}^n$

24 **if** $Y_r[i] \in \mathrm{Range}(\pi)$ **then** { $bad \leftarrow \texttt{true}$; $Y_r[i] \xleftarrow{R} \overline{\mathrm{Range}}(\pi)$ }

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** $X_r[m_r] \in \mathrm{Domain}(\pi)$ **then** { $bad \leftarrow \texttt{true}$; $TAG_r \xleftarrow{R} \overline{\mathrm{Range}}(\pi)$ }

29 **else** $TAG_r \xleftarrow{R} \{0,1\}^n$

30 **if** $TAG_r \in \mathrm{Range}(\pi)$ **then** { $bad \leftarrow \texttt{true}$; $TAG_r \xleftarrow{R} \overline{\mathrm{Range}}(\pi)$ }

31 $\pi(X_r[m_r]) \leftarrow TAG_r$

32 $Tag_r \leftarrow TAG_r$ [first τ bits]

33 **return** Tag_r

Figure 6: **Game 1.** This game accurately simulates PMAC[Perm$(n), \tau$].

[24] P. Rogaway. PMAC: A parallelizable message authentication code. Contribution to NIST, October 16, 2000. Earlier version of this paper. Available at `http://csrc.nist.gov/encryption/modes/workshop1/`

[25] P. Rogaway (submitter), and M. Bellare, J. Black, and T. Krovetz (auxiliary submitters). OCB Mode. Contribution to NIST. April 1, 2001. Available at `http://csrc.nist.gov/encryption/modes/`

[26] 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 Proofs

A.1 Proof of the Structure Lemma (Lemma 1)

Let A be an adversary that attacks PMAC[Perm$(n), \tau$]. 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$(n), \tau$] oracle as A playing a certain game, Game 1, as defined in Figure 6. This game perfectly simulates the behavior of PMAC[Perm$(n), \tau$]. 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 size of this set starts at 1 (after executing line 10) and grows one element at a time until, by 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** $X_r[i] \in \mathrm{Domain}(\pi)$ **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** $\mathit{bad} \leftarrow \texttt{true}$
27 $TAG_r[m_r] \xleftarrow{R} \{0,1\}^n$
28 $\pi(X_r[m_r]) \leftarrow TAG_r[m_r]$
29 $Tag_r \leftarrow TAG_r$ [first τ bits]
30 **return** Tag_r

Figure 7: **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 $\mathbf{Adv}^{\mathrm{prf}}_{\mathrm{PMAC[Perm}(n),\tau]}$.

time just before the last addition of a point, it has size σ. Thus we have that

$$\begin{aligned} \Pr_1[\mathit{bad} \text{ gets set in lines 24 or 30 }] &\leq \frac{1+2+\ldots+\sigma}{2^n} \\ &\leq \frac{(\sigma+1)^2}{2^{n+1}} \end{aligned} \tag{2}$$

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 (2). 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 7. At this point we know that

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathrm{PMAC[Perm}(n),\tau]}(A) \leq \Pr_2[\mathit{bad} \text{ gets set }] + \frac{(\sigma+1)^2}{2^{n+1}} \tag{3}$$

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 8.

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, *bad* will rarely be set. In fact, we show something stronger: that even if one arbitrarily fixes the values of $TAG_1, \ldots, TAG_q \in \{0,1\}^n$ (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

```
When A makes its r-th query, M_r = M_r[1] ··· M_r[m_r], where r ∈ [1..q]
10      TAG_r ←R {0,1}^n
11      return TAG_r [first τ bits]

When A is done making its q queries
20      L ←R {0,1}^n;   π(0^n) ← L
21      for r ← 1 to q do
22          for i ← 1 to m_r − 1 do
23              X_r[i] ← M_r[i] ⊕ γ_i · L
24              if X_r[i] ∈ Domain(π) then Y_r[i] ← π(X_r[i])
25              else { Y_r[i] ←R {0,1}^n;   π(X_r[i]) ← Y_r[i] }
26          Σ_r ← Y_r[1] ⊕ Y_r[2] ⊕ ··· ⊕ Y_r[m_r − 1] ⊕ pad(M_r[m_r])
27          if |M_r[m_r]| = n then X_r[m_r] ← Σ_r ⊕ huge · L else X_r[m_r] ← Σ_r
28          if X_r[m_r] ∈ Domain(π) then bad ← true
29          π(X_r[m_r]) ← TAG_r
```

Figure 8: **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 ←R {0,1}^n;   π(0^n) ← L
11      for r ← 1 to q do
12          for i ← 1 to m_r − 1 do
13              X_r[i] ← M_r[i] ⊕ γ_i · L
14              if X_r[i] ∈ Domain(π) then Y_r[i] ← π(X_r[i])
15              else { Y_r[i] ←R {0,1}^n;   π(X_r[i]) ← Y_r[i] }
16          Σ_r ← Y_r[1] ⊕ Y_r[2] ⊕ ··· ⊕ Y_r[m_r − 1] ⊕ pad(M_r[m_r])
17          if |M_r[m_r]| = n then X_r[m_r] ← Σ_r ⊕ huge · L else X_r[m_r] ← Σ_r
18          if X_r[m_r] ∈ Domain(π) then bad ← true
19          π(X_r[m_r]) ← TAG_r
```

Figure 9: **Game 4**$[\mathcal{C}]$. This game depends on constants $\mathcal{C}$ which specify: q, $TAG_1, \ldots, TAG_q \in \{0,1\}^n$, and $M_1 = M_1[1] \cdots M_1[m_1], \ldots, M_q = M_q[1] \cdots M_q[m_q]$.

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 9. It depends on constants $\mathcal{C} = (q,\ TAG_1, \ldots, TAG_q,\ M_1, \ldots, M_q)$. At this point in the proof we have that

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

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 $TAG_1, \ldots, TAG_q \in \{0,1\}^n$.

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,

```
10    L ←R {0,1}^n;   π(0^n) ← L
11    for r ← 1 to q do
12        for i ← 1 to m_r − 1 do
13            X_r[i] ← M_r[i] ⊕ γ_i · L;   Y_r[i] ←R {0,1}^n
14            if M_r[i] = M_s[i] for some s < r and i < m_s then Y_r[i] ← π(X_s[i])
15            else if X_r[i] ∈ Domain(π) then bad ← true
16            π(X_r[i]) ← Y_r[i]
17        Σ_r ← Y_r[1] ⊕ Y_r[2] ⊕ ··· ⊕ Y_r[m_r − 1] ⊕ pad(M_r[m_r])
18        if |M_r[m_r]| = n then X_r[m_r] ← Σ_r ⊕ huge · L else X_r[m_r] ← Σ_r
19        if X_r[m_r] ∈ Domain(π) then bad ← true
20        π(X_r[m_r]) ← 0^n
```

Figure 10: **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.

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 signficance 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 $Y_1 = Y_1[1]\cdots Y_1[m_1]$, ..., $Y_q = Y_q[1]\cdots Y_q[m_q]$, 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 possiblities:

- 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 probablity 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 $\sum_{1\le r\le q} \mathrm{Mcoll}_n(m_r)$.

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 $\sum_{1\le s<u\le q} \mathrm{MMcoll}_n(m_s, m_u)$.

This completes the proof.

A.2 Proof of the Collision Bound (Lemma 2)

Let $\mathit{Unequal}' = \mathit{Unequal} \setminus \{\bar{X}[\bar{m}]\}$ (multiset difference: remove one copy of $\bar{X}[\bar{m}]$) and define

- $D_1 = \{0^n\}$

- $D_2 = \{X[1], \ldots, X[m-1]\}$
- $D_3 = \{X[m]\}$
- $D_4 = \{\bar{X}[j] : j \in \mathit{Unequal}'\}$
- $D_5 = \{\bar{X}[\bar{m}]\}$

We first show that for any two points $X[i]$ and $X[j]$ in the multiset $D_1 \cup D_2 \cup D_3$ ($i < j$, $X[0] = 0^n$) 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 $\mathrm{Mcoll}_n(m) \leq \binom{m+1}{2} \cdot \frac{1}{2^n}$ we 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 the fact 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.

Similarly, 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 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 being offset by the same amount, $\mathit{huge} \cdot L$, so this offset is irrelevant in computing $\Pr[X[m] = \bar{X}[\bar{m}]$ and one proceeds exactly 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.

B Test Vectors

PMAC-AES test vectors and reference code, prepared by Ted Krovetz (`tdk@acm.org`), are available at `http://www.cs.ucdavis.edu/~rogaway/`

C Document History

- October 16, 2000. First version of the PMAC document [24]. Submitted to NIST.
- April 1, 2001. Second version of the PMAC document. Submitted to NIST.
- April 18, 2001. Revised the sections formerly marked "draft."