A Message Authentication Code Based on Unimodular Matrix Groups

Matthew Cary⋆[1] and Ramarathnam Venkatesan[2]

[1] University of Washington
cary@cs.washington.edu
[2] Microsoft Research
venkie@microsoft.com

Abstract. We present a new construction based on modular groups. A novel element of our construction is to embed each input into a sequence of matrices with determinant ± 1, the product of which yields the desired MAC. We analyze using the invertibility and the arithmetic properties of the determinants of certain types of matrices; this may be of interest in other applications. Performance results on our preliminary implementations show the speed of our MAC is competitive with recent fast MAC algorithms, achieving 0.5 Gigabytes per second on a 1.06 GHz Celeron.
Keywords: Message authentication, efficient MAC, hash functions.

1 Introduction

Algorithms to compute message authentication codes (MACs) are important in security applications, and the task of constructing them rigorously and efficiently has been a subject of many papers. An introduction may be found in [MvOV97].

MAC algorithms use a secret key K to select a function $H_K(X)$ and map an input X into a short binary string $h = H_K(X)$ of some fixed length. Then, h is encrypted using a block cipher. If the cipher acts as a random permutation, the encryptions of the hash values $h_1, ..., h_q$ of q distinct inputs $X_1, ..., X_q$ can not be distinguished from truly random outputs of the corresponding length, if the hash values $h_i = H_K(X_i)$ are distinct. Thus the collision properties of the hash function determines the security of the MAC. The main parameter of interest is the *collision probability* $\Pr_K[H_K(X) = H_K(X')]$ where X and X' are arbitrary and distinct inputs. If this probability is the inverse of the size of the range, the family of H_K is called *universal* [CW81]. This approach has enabled the construction of families (See [BHK$^+$99, HK97, Rog99, BCK96, Sho96, JV98, Ber]) with quantifiable collision probabilities that are quite fast in practice. In this paper, we will focus on the initial mapping $X \mapsto h$ and its collision probability, and assume for simplicity that all inputs are ℓ-bit word sequences which are the same length, and can be subdivided into blocks of convenient length t evenly.

⋆ Research done while at Microsoft Research. Partially supported by NSF grant CCR-0098066

Previous Work

To motivate our construction, we recall some earlier ones. The evaluation MAC identifies an input message $X = x_1, \dots x_t$ with a polynomial of degree t over a suitable field and computes the map $\alpha \mapsto \sum_i x_i \alpha^i$ for a random α. (See [Sho96] and [Ber] for speed-ups; the latter uses floating-point operations in a manner which may apply to our construction).

Many MAC constructions use a standard iterative rule $y_i = f_i(x_i + y_{i-1})$, where y_i are the *intermediate values* and various methods use different f_i's. In the evaluation MAC, $f_i(x) = f(x) = \alpha x$, the iteration is Horner's rule and y_m is the final value, while if one takes $f_i = f(x) = E_K(x)$ to be a block cipher, one gets the CBC MAC (see [BKR00] for an analysis). The chain and sum method in [JV98] doubles the length of the hash in a one-pass computation by outputting the pair $(y_t, \sum y_i)$. It alternates two *random* affine transformations $x \mapsto ax + b$, one for for odd i, and one for even i, each using an extra multiplier in the iteration $y_i = f(ex_i + y_{i-1})$. The invertibility of the operations allows one to combine MAC and encryption to obtain a pseudo-random permutation on X by further encrypting the intermediate values $y_1, ... y_{t-2}$ with an one-time pad derived from $(y_t, \sum y_i)$ using a stream cipher and encrypting $(y_t, \sum y_i)$ with a block cipher. We will also use the sum of intermediate values in our hash.

These methods work over a field where operations are typically expensive and using arithmetic modulo 2^ℓ is advantageous, as the fastest MACs indeed do. However, we lose invertibility (in the multiplicative sense) which is crucial for analysis. To wit, for $x \neq x'$, the function $f(x) = \alpha x + b$ over a field has a uniform *output differential* $f(x) - f(x') = \alpha(x - x')$ in the sense that it is uniformly distributed if α is randomly chosen. However, modulo 2^ℓ this changes sharply. If $2^{\ell-1} | (x - x')$ then $2^{\ell-1} | (y - y')$, and the output is distributed on a set of size 2 for a random odd α. Our goal is to construct reversible transformations that are suitable for MAC (and other applications) keeping the structure of the proof in the finite field case, except our equations involve coefficients from matrix groups.

UMAC [BHK+99] uses the iteration $y_i = y_{i-1} + f(x_{2i}, x_{2i+1}) \mod 2^{2\ell}$ where $f(x_{2i}, x_{2i+1}) = (x_{2i} + k_{2i}) \cdot (x_{2i+1} + k_{2i+1}) \mod 2^{2\ell}$, k_i are random and all variables except y_i are ℓ-bits. This allows leveraging SIMD available on today's CPU's for media processing to hash more than a byte per cycle or gigabyte per second.

Recently Klimov and Shamir [KS] constructed an elegant family of invertible mappings (modulo 2^ℓ) that combine arithmetic and boolean operations to get non-linear maps for use in cryptographic primitives. These functions need to be randomized and modified to have suitable differential properties, in order to be used for our and similar applications.

Our Construction

PRELIMINARIES: Recall that our inputs are broken into blocks of length t words, each of size ℓ-bits. We use a sequence 2×2 matrices (A_i) with $\det(A_i) = \pm 1$

and fixed independent of the input x_i; the sequence may be periodic so that implementations can be unrolled with small code footprint. We define invertible functions $f_i(x)$ by multiplication with odd a_i, where a_i and x are ℓ bits, and the 2ℓ bit result is viewed as a vector of two ℓ bit numbers. The a_i form the secret key of the MAC and are assumed to be random. Thus $f_i(x)$ is invertible modulo $2^{2\ell}$ and can be implemented in one instruction using the usual 2ℓ-bit result of multiplication of two ℓ-bit quantities. One may be able to use floating point arithmetic as in [Ber]. All our matrix operations are over ring of integers modulo 2^ℓ.

THE ALGORITHM: We embed a given ℓ-bit input x_i into a 3×3 matrix B_i by $x_i \mapsto \left[\begin{smallmatrix} A_i & v_i \\ 0\,0 & 1 \end{smallmatrix}\right] =: B_i$, where $v_i = f_i(x_i)$ is a vector with two elements. For each block of input, we compute the product $B = \left[\begin{smallmatrix} A & z \\ 0\,0 & 1 \end{smallmatrix}\right]$ of these matrices B_i. The output of our hash value is the pair $(z, \sum_{i=1}^t v_i)$.

We show that the collision probability is nearly $2^{-2\ell}$, by using the invertibility of A_i and the arithmetic properties of the determinants of the matrices of the form $\prod_{i=j}^k A_i - I$ over $\mathbb{Z}$ (and not modulo 2^ℓ). We believe this new approach offers simplicity and can be helpful in other applications than MACs.

Our construction can be viewed in a more general setting. Let G be the group of integer matrices with determinant ± 1, $V = \mathbb{Z}^2_{2^\ell}$ be the additive group of 2-dimensional vectors modulo 2^ℓ and $G \ltimes V$ be their semi-direct product by the natural action of G on V. Then the above maps each x_i into $G \ltimes V$, by mapping x_i to $\big(A_i, f_i(x_i)\big)$ and takes the product of the images in $G \ltimes V$. This can be generalized into higher dimensions (see the section 6), and one can also view this hashing as a walk on the associated directed Cayley graphs.

The reader may have noticed that in the above and in UMAC the block keys are random independent sequences of words. UMAC expands a short key into a longer one by using a secure pseudo-random generator, and after the first block, needs only small amount of additional keys on a per block basis. Our algorithm is similar and a closer look security and performance at reducing this key generation may be desirable for some applications.

Our current version, implemented on an Intel Celeron, while not fully optimized, is competitive or better than known algorithms but slower than UMAC. However, we believe our novel construction is interesting in its own right, and may lend itself to other applications. Further refinement may improve the speed of our algorithm, and reduce the amount of key used. Implementations on imminent architectures such as AMD with a larger number of registers and different parallel constructs may also improve our speed.

Our algorithm suggests a *semi-universal* model for checksums for files, where MAC's may be an overkill. In section 4 we show that any two inputs that collide within a block, must differ in at least two locations. The collision probability of our MAC is much smaller if the input differs in at least three locations, which we may call 3-semi-universal hashing. Such variations and generalizations may lead to more efficient constructions. We omit the details in this version.

Section 2 describes some conventions we use in this paper. Section 3 describes the construction, which is analyzed in Section 4. We give experimental performance results in Section 5, and conclude with open problems in Section 6.

2 Conventions

Fix a modulus $m = 2^\ell$, for example, $\ell = 32$. A *word* will refer to an element of $\mathbb{Z}_m = \mathbb{Z}/m\mathbb{Z}$ and a *double word* to an element of $\mathbb{Z}_{m^2}$. Hence, words can be though of as ℓ bit integers, and double words as 2ℓ bit integers. All operations will take place over words, that is, over $\mathbb{Z}_m$, unless otherwise specified. We will take advantage of the ability of modern processors to multiply two words to produce a double word in a single instruction; this operation will be denoted $\times_*$. For $x, y \in \mathbb{Z}_m$, $x \times_* y$ will be in $\mathbb{Z}_m^2$, that is, we view the result as a two word vector.

We shall assume, if necessary by padding, the input to consist integral number of words. For simplicity we assume our input consists of b blocks each of which has a fixed *block length* of t words.

3 The Construction

Hashing One Block

We now describe construction for a map v that sends an input block $X = x_1, \ldots, x_t$ into ℓ-bit hash value $v = v(X)$. The block key consists of ℓ-bit words a_i, for $1 \le i \le t$; the same key is reused with each block. We define $f_i : \mathbb{Z}_m \to \mathbb{Z}_m^2$ by $f_i(x) = a_i \times_* x$. Our algorithm uses fixed public matrices $A_1, \ldots, A_t$. These will contain very small entries so that matrix products can be implemented very efficiently by addition and subtraction of words.

Let v_i be the column vector of two words equal to $f_i(x_i)$. Define matrices B_i, B and B_0, which have the form $\left[\begin{smallmatrix} * & * & * \\ * & * & * \\ 0 & 0 & 1 \end{smallmatrix}\right]$, where $B_0 = \left[\begin{smallmatrix} 1 & 0 & \\ 0 & 1 & z_0 \\ 0 & 0 & 1 \end{smallmatrix}\right]$, and for $i > 0$,

$$B_i := \begin{bmatrix} A_i & v_i \\ 0\ 0 & 1 \end{bmatrix}, \; B := B_0 \cdot \prod_{i=1}^{t} B_i =: \begin{bmatrix} A & z \\ 0\ 0 & 1 \end{bmatrix} \tag{1}$$

It is clear that B can be written as above; z is the first two components of the third column of B and A has determinant ± 1. z_0 is an initial value for the block. We also compute

$$\sigma = \sigma_0 + \sum_{i=1}^{t} v_i,$$

where σ_0 is another initial value for the block. The hash value is $v(X) = (z, \sigma)$.

Inter-Block Chaining

The k^{th} block is associated with two uniform hash functions $F_1^{(k)}$ and $F_2^{(k)}$ mapping double words to double words. We drop the superscript if the block number is clear from context. If (z', σ') is the output of a hashed block, we chain this to the next block by setting $\sigma_0 = F_2(\sigma')$ and

$$B_0 = \begin{bmatrix} 1\,0 & \\ 0\,1 & F_1(z') \\ 0\,0 & 1 \end{bmatrix}$$

as the initial values for the next block. These inter-block functions may be repeated to save on key length, at some cost of security, which will be detailed in the analysis. The exact definition of these functions is not important for our applications.

Doubling the Hash Value Length

The hash value length can be doubled by performing an independent hash in parallel. We use key words b_i, $1 \le i \le t$ which is independent of the a_i, and set the functions g_i, $i \le t$ to $g(x) = b_i \times_* x$. We define $u_i = g_i(x_i)$, and as above get a map $X \mapsto u(X)$ with the hash value u using

$$C_i := \begin{bmatrix} A_i & u_i \\ 0\,0 & 1 \end{bmatrix},\ C := C_0 \cdot \prod_{i=1}^{t} C_i =: \begin{bmatrix} A & w \\ 0\,0 & 1 \end{bmatrix},\ C_0 := \begin{bmatrix} 1\,0 & \\ 0\,1 & u_0 \\ 0\,0 & 1 \end{bmatrix}. \tag{2}$$

We also compute $\nu = \nu_0 + \sum_{i=1}^{t} u_i$. The overall hash will now be $\big(v(X), u(X)\big) = (z, \sigma, w, \nu)$.

Main Result

Theorem 1. *For $t \le 50$, if $H = (z, \sigma, w, \nu)$ and $H' = (z', \sigma', w', \nu')$ are the hash values computed from two distinct inputs, then*

$$\Pr[H = H'] \le 2^{-4\ell+20},$$

where the probability is taken over the choice of key.

This theorem will follow directly from Lemmas 3 and 4. We note that the theorem is not optimal, in that the choice for the matrices of Lemma 4 could be improved.

4 Analysis

Collisions in a Block

We will first concentrate on the analysis of the hash of a single block, and assume that $B_0 = I$, the 3×3 identity matrix. By repeated use of the identity

$$\begin{bmatrix} A & v \\ 0\,0 & 1 \end{bmatrix} \cdot \begin{bmatrix} B & u \\ 0\,0 & 1 \end{bmatrix} = \begin{bmatrix} AB & Au+v \\ 0\,0 & 1 \end{bmatrix}$$

in equation (1), we have that

$$z = v_1 + A_1 v_2 + A_1 A_2 v_3 + \cdots + A_1 A_2 \cdots A_{t-1} v_t. \tag{3}$$

For two (not necessarily distinct) input blocks, we write $X = x_1, \ldots, x_t$ and $X' = x'_1, \ldots, x'_t$ define $v'_i = f_i(x'_i)$, and define z' and σ' analogously.

We need the following technical lemma relating the distributive law of of $\times_*$ over vector subtraction. We remark that in general it is not true that $a \times_* x - a \times_* x' = a \times_* (x - x')$ and thus the operation is not linear. However, assuming $x \neq x'$, $a \times_* x - a \times_* x'$ is nearly as likely to collide with any fixed value as $a \times_* (x - x')$.

Lemma 1. *Given any fixed words $x \neq x'$ and any fixed double word $\alpha = (\alpha_1, \alpha_2)$,*

$$\Pr_a[a \times_* x - a \times_* x' = \alpha] \leq 2^{-\ell+2},$$

where the probability is taken over uniformly chosen odd words $a \in \mathbb{Z}_m$.

Proof. For this proof we let $\cdot$ denote the usual multiplication over double words. By abusing notation we write $a \cdot x = y$ for $a, x \in \mathbb{Z}_m$ and $y \in \mathbb{Z}_{m^2}$; we note also in this case there is no overflow, so that $y = ax$ as integers. The crux of this lemma is the difference between subtraction over double words as integers modulo m^2, and subtraction over two-dimensional vectors modulo m. To make this distinction explicit, for an element $x \in \mathbb{Z}_{m^2}$ we write $[x]$ as the vector corresponding x, so that $[x] \in \mathbb{Z}_m^2$. Then for double words y and z, if $[y] - [z] = (w_1, w_2)$, then $[y - z] = (w_1 - c, w_2)$, where c is either 0 and 1 depending on whether there is a carry between the low and high words or not.

Let A be the set of all odd a that cause a collision, that is, for the fixed $\alpha = (\alpha_1, \alpha_2)$, all a such that $[a \cdot x] - [a \cdot x'] = \alpha$ for x and x' as in the statement of the lemma. Then for any $a \in A$, $[a \cdot x - a \cdot x'] = (\alpha_1 - c_a, \alpha_2)$, for $c_a = 0$ or 1. Given $a, a' \in A$ with $c_a = c_{a'}$, we have $a \cdot (x - x') = a' \cdot (x - x')$ over the integers, so that as $x \neq x'$, $a = a'$. Thus A contains at most two elements, possibly one with carry 0 and possibly one with carry 1. As there are $2^{\ell-1}$ choices for odd a, the chance of choosing one in A is at most $2 \cdot 2^{-\ell+1} = 2^{-\ell+2}$, as required. □

We now begin our analysis of the hash function proper.

Lemma 2. *If $(z, \sigma) = (z', \sigma')$ for distinct inputs X and X', then X and X' differ in at least two locations.*

Proof. Suppose not, so that $x_i = x'_i$ for all $i \neq j$, and $x_j \neq x'_j$ for some j. Then $\sigma - \sigma' = a_j \times_* x_j - a_j \times_* x'_j$. As a_j is odd and hence an invertible map from $\mathbb{Z}_m \to \mathbb{Z}_m^2$, $\sigma \neq \sigma'$, contradicting $(z, \sigma) = (z', \sigma')$. □

We now know that colliding inputs have at least two distinct words, however we do not know which words these are. This is where computing the hash as a matrix product and sum helps us. For example, if x and y are independently distributed over $\mathbb{Z}_m$, then $2x + y$ and $2y - x$ are independently distributed as well. Note, however, that $x + y$ and $x - y$ are not independently distributed; for example, they have the same parity. The difference between these two examples is that the former arises from the matrix $\left[\begin{smallmatrix} 2 & 1 \\ -1 & 2 \end{smallmatrix}\right]$, which is invertible over $\mathbb{Z}_m$, while the the matrix of the latter is $\left[\begin{smallmatrix} 1 & 1 \\ 1 & -1 \end{smallmatrix}\right]$ has determinant -2, and so is not invertible over $\mathbb{Z}_m$. The relationship between the two components of our hash pair, z and σ, is similar, so that if we pick our matrices carefully, z and σ will be independent.

Definition 1. *A sequence of matrices $A_1, \ldots, A_t$ is* k-invertible *if for any $i < j$, and Δ defined as*

$$\Delta = \det(A_i \cdots A_{j-1} - I),$$

if Δ is nonzero, and if $2^{k'} | \Delta$, then $k' \leq k$.

For any interval $\mathcal{I} = [i, j)$, the matrix $B = \prod_{\mathcal{I}} A_i - I$ formed from a k-invertible sequence of A_i is nearly invertible in the following sense. Let $\det(B) = s2^{k'}$ for odd, nonzero s and $k' \leq k$. Then $Bx = \alpha$ can be solved modulo $2^{\ell - k}$ uniquely and then there are 2^k solutions modulo 2^ℓ. Thus we need to ensure that the value k should be as small as possible.

Lemma 3. *Assume that the sequence $A_1, \ldots, A_t$ is k-invertible. Then for distinct inputs $X \neq X'$, $\Pr_{\{a_i\}}[(z, \sigma) = (z', \sigma')] \leq 2^{-2\ell+4+k}$, where $f_i(x) = a_i \times_* x$.*

Proof. Let $\delta x_i = x_i - x'_i$ and $\delta v_i = f(x_i) - f(x'_i) = a_i \times_* x_i - a_i \times_* x'_i$. By Lemma 2, we can assume that there exist $i < j$ such that $\delta x_i \neq 0$ and $\delta x_j \neq 0$. Our analysis now is in terms of matrix equations over $\mathbb{Z}_m$ involving A_i's and δv_i ; the inputs x_i and x'_i are involved implicitly in a non-linear way which will by Lemma 1 will cost us a factor of 2. By fixing all a_r for $r \neq i, j$, we have that

$$\begin{aligned} &\Pr_{a_i, a_j}[(z, \sigma) = (z', \sigma')] = \\ &\qquad \Pr_{a_i, a_j}\left[A_1 \cdots A_{i-1} \delta v_i + A_1 \cdots A_{j-1} \delta v_j = \alpha, \delta v_i + \delta v_j = \beta\right], \end{aligned} \tag{4}$$

for appropriate fixed α and β. Rearranging (4), we have that for some fixed α', it is equivalent to

$$\Pr_{a_i, a_j}\left[(A_i \cdots A_{j-1} - I)\delta v_j = \alpha', \delta v_i + \delta v_j = \beta\right].$$

Let $B = (A_i \cdots A_{j-1} - I)$, and let $\Delta = \det B$. As the sequence $A_i, \ldots, A_{j-1}$ is k-invertible, $\Delta = s \cdot 2^{k'}$ for some odd s and $k' \leq k$. As remarked above, $B\delta v_j = \alpha'$ iff

$2^{k'}\delta v_j = \alpha^*$ in $\mathbb{Z}_m$, for some fixed α^* depending on α' and B. As from Lemma 1 $\Pr_{a_j}[\delta v_j = \gamma] \le 2^{-\ell+2}$ for any fixed γ, $\Pr_{a_j}[2^{k'}\delta v_j = \alpha^*] \le 2^{-\ell+2+k'} \le 2^{-\ell+2+k}$ (recall all operations are performed over $\mathbb{Z}_m$). Finally, if the event $2^k\delta v_j = \alpha^*$ occurs, then $\Pr_{a_i}[\delta v_i + \delta v_j = \beta] \le 2^{-\ell+2}$ (and can be possibly zero), as δv_i depends only on a_i, independently from v_j. Multiplying these probabilities gives the lemma. □

Inter-Block Chaining

We now consider the operation of the hash over several blocks. Let (z_k, σ_k) be the output of the k^{th} block, so that the initial values for the $k+1$ block are $F_1^{(k)}(z_k)$ and $F_2^{(k)}(\sigma_k)$. If the keys for the pair $(F_1^{(k)}, F_2^{(k)})$ are new at each block, then the initial positions at each block are independent, using the uniformity of the F_i. Given two messages $X_1, \ldots, X_n$ and $X'_1, \ldots, X'_n$, let i be the largest index of different blocks, so that $X_i \neq X'_i$ and $X_j = X'_j$ for $j > i$. Then $H(X_1, \ldots, X_n) = H(X'_1, \ldots, X'_n)$ iff $(z_i, \sigma_i) = (z'_i, \sigma'_i)$. If $H(X_1, \ldots, X_{i-1}) = H(X'_1, \ldots, X'_{i-1})$, then the probability that $(z_i, \sigma_i) = (z'_i, \sigma'_i)$ is given in Lemma 3. Otherwise, by fixing all key bits but those for $F_r^{(i-1)}$, $r = 1, 2$, the probability that $(z_i, \sigma_i) = (z'_i, \sigma'_i)$ is equal to that of a collision in the $F_r^{(i-1)}$, which is smaller than that of Lemma 3. If we wish to save on key size, the $F_j^{(i)}$ can be reused. A standard union-bound shows that the bit-security of the hash decreases linearly with the frequency of reuse.

Choice of Matrices

The choice of the sequence $A_1, \ldots, A_t$ can be tailored to implementation requirements. Obviously there is a trade-off between finding k-invertible matrices for minimum k while ensuring that the matrix-vector products of the hashing algorithm can be efficiently computed. Our implementations in §5 use the families below. We caution that if the order of the matrices is changed, the determinants of interest may be identically zero!

Lemma 4. *Define the following integer matrices of determinant* ± 1*.*

$$A'_1 = \begin{pmatrix} -1 & 1 \\ 1 & -2 \end{pmatrix},\ A'_2 = \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix},\ \textit{and}\ A'_3 = \begin{pmatrix} 1 & 3 \\ 1 & 2 \end{pmatrix}.$$

We now extend this periodically into a longer sequence: $\mathcal{A}_t = (A_1, \ldots, A_t)$ *where* $A_{i+3s} = A'_i$*. Then* $\mathcal{A}_{19}$ *is 4-invertible, and* $\mathcal{A}_{50}$ *is 6-invertible.*

Proof. This can be verified by direct computation. A plot of the the k-invertibility of $\mathcal{A}_{50}$ is shown in Figure 1. □

It would be interesting to see if the noticeable structure in the plot can be exploited. We now present another family of matrices whose near-invertibility is not as good; however these have entries from $\{\pm 1, 0\}$ yielding more efficient

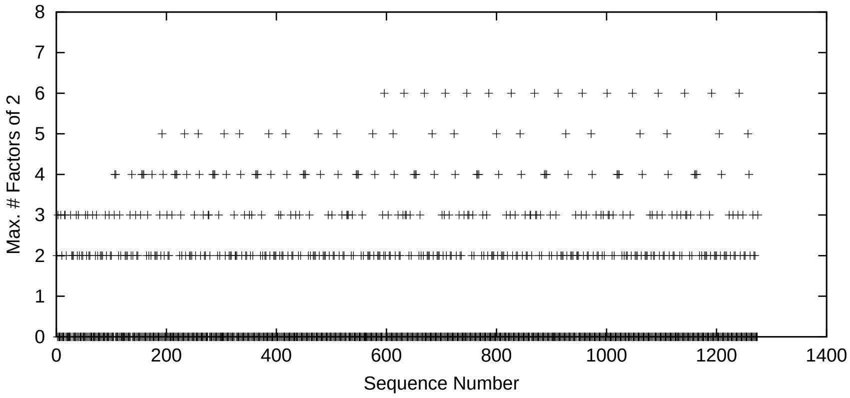

Fig. 1. The 6-invertibility of $\mathcal{A}_{50}$. The y-axis is the largest $k \geq 0$ such that $2^k | \det\Big(\big(\prod_i^j A_s\big) - I\Big)$, where the interval $\{i \dots j\}$ is given by the sequence number. The determinant is nonzero in all cases.

implementations. Preliminary implementations suggest a 15% speed-up when using these simpler matrices. Of perhaps more interest, we can also show the determinants of interest are non-zero, if not nearly odd.

Lemma 5. *Define the following matrices.*

$$B_1' = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, B_2' = \begin{pmatrix} -1 & -1 \\ 0 & -1 \end{pmatrix}, B_3' = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}, \textit{ and } B_4' = \begin{pmatrix} -1 & 0 \\ -1 & -1 \end{pmatrix}.$$

Set $B_i = B'_{(i \bmod 4)+1}$ *and* $\mathcal{B}_t = (B_1, \dots, B_t)$. *Then for any* $1 \leq i \leq j \leq t$, *if* $M = \prod_i^j B_s$, $\det(M - I) \neq 0$.

This is a necessary condition for k-invertibility, though clearly is insufficient in general. Experimentally, $\mathcal{B}_t$ is roughly $\log_{1.5} t$-invertible. For $t \sim 50$, they are not as invertible as $\mathcal{A}_{50}$, so we have not used them in our implementation. Figure 2 shows the growth of the k-invertibility of $\mathcal{B}_t$ as t is increased.

Proof. For a matrix A, we write $A \geq 0$ if each entry of A is at least 0. We write $A \leq 0$ if $-A \geq 0$, and $A \geq A'$ if $A - A' \geq 0$. We also write $|A|$ to denote the matrix whose entries are the absolute value of those of A.

In the notation of Lemma 5, note that

$$X_1 = B_1'B_2' = B_2'B_3' = \begin{pmatrix} -1 & -2 \\ -1 & -1 \end{pmatrix} \text{ and } X_2 = B_3'B_4' = B_4'B_1' = \begin{pmatrix} -1 & -1 \\ -2 & -1 \end{pmatrix}.$$

By examination we have for all $1 \leq s \leq 4$, $\det(B_s' - I) \in \{-1, 4\}$ and hence nonzero, and $\mathrm{Tr}(B_s') \in \{1, -1\}$ and at least 1 in absolute value. For $r = 1, 2$, $\det(X_r - I) = 2 \neq 0$ and $\mathrm{Tr}(X_r) = -2$. Finally, $\det(B_s'X_r - I) \in \{-4, -3, 6\}$.

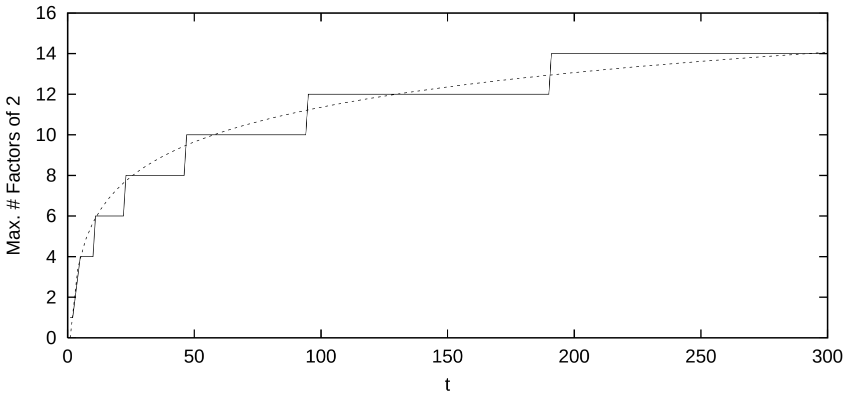

Fig. 2. The k-invertibility of $\mathcal{B}_t$ (solid line) plotted against $\log_{1.5} t$ (dashed line). Here the y-axis is the largest k such that $2^k | \det\Big(\big(\prod_i^j B_s\big) - I\Big)$, for all $1 \leq i \leq j \leq t$, for the specified t.

Hence we can proceed by induction, and assume $j - i > 2$. Set $M' = \prod_{s=i}^{j-2} B_s$ and fix r so that $M = M'X_r$, and by induction we can assume that $\big|\mathrm{Tr}(M')\big| \geq 2$

Since $\det(M) = \pm 1$, $\det(M - I) = \det(M) + 1 - \mathrm{Tr}(M)$, and $\det(M) + 1 =$ 0 or 2, it will be enough to show that $|\,\mathrm{Tr}(M)| > 2$. Note that $M \geq 0$ or $M \leq 0$, for $B_s = \pm 1 \cdot |B_s|$, so that $M = \pm 1 \cdot \prod_i^j |B_s|$, and $\prod |B_s| \geq 0$. As $M' \geq 0$ or $M' \leq 0$, using the same argument as for M, by examining X_r we see that $|M| \geq |M'|$.

One can label the off-diagonal elements of M' by x and y, so that

$$\mathrm{Tr}(M) = \mathrm{Tr}(M'X_r) = -\big(\big|\mathrm{Tr}(M')\big| + 2|x| + |y|\big),$$

if necessary by exchanging x and y. In a similar way as showing $|M| \geq |M'|$, one can show $|M'| > 0$, so thus $\big|\mathrm{Tr}(M)\big| \geq \big|\mathrm{Tr}(M')\big| + 1 \geq 3$, using the inductive assumption on M'. Hence $\det(M - I) \neq 0$, as required. $\square$

5 Preliminary Implementation

Our hash design is mindful of operating constraints of modern processors. In particular it admits parallelization which is useful now that SIMD operations are standard on most computers. For example, the MMX instruction set standard on Intel Pentium II and later processors can operate simultaneously on 32-bit words with a throughput of 2 per cycle.

We mention some caveats. As written above, our test version is not optimized and availability of extra registers and parallelism in imminent machines must be taken in to account. The comparison numbers are to be taken as indicative

Algorithm	Security (Bits)	Peak Rate (cycles/byte)	Key Size (8 Kbyte Message)
Ours (two streams)	108	3.7	13.6 Kbits
Ours (one stream)	54	2.0	6.8 Kbits
UMAC	60	0.98	8 Kbits
SHA-1	80	12.6	512 bits

Data for other algorithms taken from [BHK+99, BHK+00].

Fig. 3. MAC Comparisons

of the parameters chosen in the test, and the results may depend on it. The comparison of key size generation was not closely looked at since this would mean re-coding and testing other algorithms with different parameters. The numbers presented on key sizes here may offer a different picture than what one would get if aggressive optimizations were made.

For brevity, we say that a hash or MAC has s bits of security if the collision probability (over the choice of keys) on two distinct fixed messages is $\leq 2^{-s}$. Using $\mathcal{A}_{50}$, by Lemma 3 each hash gives $2 \cdot 32 - 4 - 6 = 54$ bits of security, using 30 32-bit words of key per MAC per stream, plus the key for the inter block chaining. As two MACs are computed, the total security is 108 bits. Using MMX instructions on a 1.06 GHz Celeron, this MAC was computed at a peak rate of 3.7 cycles per byte. We have not implemented an optimized SSE2 algorithm to determine if the extended instruction set would benefit our algorithm. We also implemented the hash using a single stream, which gives 54 bits of security. This achieved a peak rate of 2.0 cycles per byte.

Our algorithm is also competitive with UMAC on the length of the generated key. To maintain the security bounds of Lemma 3, each inter-block hash needs four 32-bit words of key per hash stream. Each of our blocks then requires $50 \cdot 2$ 32-bit words of key. Thus, for an 8 Kbyte message, 42 inter-block hashes are required, for 5376 bits of key per hash stream. The total for an 8 Kbyte message and two hash streams is 13.6 Kbits of key. This compares with the current UMAC implementation [BHK+00] which requires 8 Kbits of generated key to hash a message of any length to 60 bits of security.

We summarize this information with context from other algorithms in Figure 3.

6 Future Work

k-INVERTIBLE MATRICES: The proof k-invertibility of our matrix sequences is computational and analytical proofs are desirable. It is not necessary for such sequences to be periodic, as ours are. More complex families may improve the speed and the security of our hash. For example, we have found a periodic sequence of 4×4 matrices of length 80 which is 4-invertible. The larger matrices can be used to consume twice as much input per iteration, and the longer sequence length means the inter-block chaining is less frequent, improving efficiency. Preliminary implementations show this is 17% faster than the matrices of Lemma 4, and 2% faster than the matrices of Lemma 5, while providing more security than the other sequences.

Other Hardware Platforms: Both our construction and UMAC benefit from the media processing instructions found on Pentium CPUs. Other platforms, such as those of AMD, or Intel's Itanium CPU, have different advantages, including larger register files. These details may change the relative performance between our MAC and UMAC, as well as motivate new directions in MAC design.

Combined MAC and Encryption: Since our operations are invertible, we may try to combine authentication and encryption with stream ciphers. The idea is rather simple: use the final hash value to define a key for a stream cipher to generate a one-time pad. Instead of encrypting the input sequence x_i, one encrypts $y_i = a_i x_i + b_i$, where a_i and b_i are random key words (the first quantity is the lower half of a v_i in a step of our MAC). As before we further need to encrypt the hash value. One needs to exercise caution here: if the addition with b_i were omitted, one could still observe correlations. This would be the case if the inputs x_i end in many zeroes and RC4 is used. It would be interesting to see if such minor known or future correlations in RC4 [Gol97, Mir02, MT98] can be masked.

Key Size Reduction: The inter-block chaining seems to contain some slack in the use of key. Almost twice as much key is used in inter-block hashing as is used for the blocks. Key reuse techniques such as a Toplitz shift (see [BHK$^+$99]) may be able to address this problem. The use of a single pairwise independent hash may be sufficient, but our proof of that is incomplete.

References

[ALW01] Noga Alon, Alexander Lubotzky, and Avi Wigderson. Semi-direct product in groups and Zig-zag product in graphs: Connections and applications. In *FOCS 2001*, pages 630–637. IEEE, 2001.

[BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. *Lecture Notes in Computer Science*, 1109, 1996.

[Ber] D. Bernstein. Floating-point arithmetic and message authentication. draft available as http://cr.yp.to/papers/hash127.dvi.

[BHK$^+$99] J. Black, S. Halevi, H. Krawczyk, T. Krovetz, and P. Rogaway. UMAC: Fast and secure message authentication. *Lecture Notes in Computer Science*, 1666:216–233, 1999.

[BHK$^+$00] J. Black, S. Halevi, H. Krawczyk, T. Krovetz, and P. Rogaway. UMAC home page, 2000. URL: `http://www.cs.ucdavis.edu/~rogaway/umac`.

[BKR00] Mihir Bellare, Joe Kilian, and Phillip Rogaway. The security of the cipher block chaining message authentication code. *Journal of Computer and System Sciences*, 61(3):362–399, 2000.

[CW81] Carter and Wegman. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 22(3):265–279, 1981.

[Gol97] J. Golic. Linear statistical weaknesses in alleged RC4 keystream generator. In *Advances in Cryptology — EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 226–238. Springer-Verlag, 1997.

[HK97] Shai Halevi and Hugo Krawczyk. MMH: Software message authentication in the Gbit/second rates. In *Fast Software Encryption*, pages 172–189, 1997.

[JV98] Mariusz H. Jakubowski and Ramarathnam Venkatesan. The chain and sum primitive and its applications to MACs and stream ciphers. In *Advances in Cryptology — EUROCRYPT '98*, volume 1403 of *Lecture Notes in Computer Science*, pages 281–293. Springer-Verlag, 1998.

[KS] Alexander Klimov and Adi Shamir. A new class of invertible mappings. Crypto 2001 Rump Session.

[Mir02] Ilya Mironov. Not so random shuffles of RC4. In *Advances in Cryptology — CRYPTO 2002*, Lecture Notes in Computer Science. Springer-Verlag, 2002.

[MvOV97] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1997.

[MT98] Serge Mister and Stafford E. Tavares. Cryptanalysis of RC4-like Ciphers In *Selected Areas in Cryptography*, 131–143, 1998.

[Rog99] Phillip Rogaway. Bucket hashing and its application to fast message authentication. *Journal of Cryptology: the Journal of the International Association for Cryptologic Research*, 12(2):91–115, 1999.

[Sho96] Victor Shoup. On fast and provably secure message authentication based on universal hashing. *Lecture Notes in Computer Science*, 1109, 1996.