The MAELSTROM-0 Hash Function

Décio Luiz Gazzoni Filho[1], **Paulo S. L. M. Barreto**[1], **Vincent Rijmen**[2]

[1] Departamento de Engenharia de Computação e Sistemas Digitais (PCS),
Escola Politécnica, Universidade de São Paulo, Brazil.

[2]Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology, Austria.

decio@decpp.net, pbarreto@larc.usp.br, vincent.rijmen@iaik.tugraz.at

Abstract. *In this paper we present* MAELSTROM-0*, an evolution of the* WHIRLPOOL *hash function with variable output length up to 512 bits. As its predecessor,* MAELSTROM-0 *is not oriented towards any particular platform, but its implementation flexibility facilitates exploiting the features of each underlying environment. On the other hand, the improved design of* MAELSTROM-0 *makes it faster and arguably more robust than its predecessor and other existing hash functions. By incorporating the state-of-the-art in the design of cryptographically secure hash functions,* MAELSTROM-0 *not only constitutes a new primitive* per se*, but also provides an initial assessment on what the minimum requirements for NIST's "Advanced Hash Standard" might be, and might serve as a valuable comparison tool for future AHS proposals in terms of security, efficiency, and flexibility.*

1. Introduction

Hash functions, intuitively speaking, are algorithms intended to generate short, (virtually) unique representatives of (virtually) arbitrarily long messages, so that these representatives can replace the messages in certain computationally expensive processes. Cryptographic hash functions are often adopted as practical instantiations of the more abstract concept of random oracles [Bellare and Rogaway 1993], which are at the core of most cryptosystems designed to provide data integrity and authentication without sacrificing efficiency. Even though some newly proposed alternatives do not rely on random oracles (see e.g. [Boneh and Boyen 2004]), these do not cover all possible needs of a security system, nor are they compatible with the conventional, currently deployed public-key infrastructure in a global scale. Besides, hash functions have long found their way into related but different applications, like confirmation of knowledge, password-based key derivation and pseudo-random number generation [Menezes et al. 1999, section 9.2.6]. It is therefore likely that hash functions will continue to play a prominent role in cryptographic applications for the foreseeable future.

Formally, hash functions map bit strings of any length less than some upper bound m to bit strings of some fixed length n. A hash function $H : \{0,1\}^{<m} \rightarrow \{0,1\}^n$ is said to be *cryptographically secure* if at least the following conditions are satisfied [Rogaway and Shrimpton 2004]:

1. (*First pre-image resistance*) Given a hash value $R \in \{0,1\}^n$, it is computationally infeasible to find a message $M \in \{0,1\}^{<m}$ such that $H(M) = R$;

2. (*Second pre-image resistance*) Given a message $M \in \{0,1\}^*$ (and, implicitly, its hash value $R = H(M)$, so that M constitutes a first pre-image of R), it is computationally infeasible to find a distinct message $M' \in \{0,1\}^{<m}$ such that $H(M') = H(M)$.
3. (*Collision resistance*) It is computationally infeasible to find two distinct messages $M, M' \in \{0,1\}^{<m}$ such that $H(M) = H(M')$, regardless of what the actual hash value is.

By "computationally infeasible" we mean that the effort to break any of these conditions should be exponential in the hash size n; typically one expects $O(2^n)$ steps to violate the first condition, $O(2^{n/2})$ steps to violate the last one, and something in between for the second condition [Kelsey and Schneier 2004, Kohno and Kelsey 2006]. Because of this, it is advisable to restrict $m \leqslant 2^{n/2}$ at most.

The recent crisis caused by the successful cryptanalysis of standardized hash functions like MD5, RIPEMD, SHA-0 and (to some extent) SHA-1, for which the last condition above was proven *not* to hold in multiblock collision attacks [Klima 2006, Wang et al. 2005a, Wang et al. 2005b, Wang and Yu 2005], has motivated a renewed interest in the design of cryptographically secure hash functions. It has also led NIST to prepare a new, soon to be announced international quest to define an "Advanced Hash Standard," similar to the Advanced Encryption Standard quest that NIST initiated nearly a decade ago. Currently, NIST is gathering feedback from the cryptological community on what the minimum requirements the candidate functions should satisfy to take part in the quest.

In this paper we describe Maelstrom-0, an evolution of the Whirlpool hash function with variable output length up to 512 bits, designed to achieve higher processing speed, particularly for longer messages. As its predecessor, Maelstrom-0 implementations on 8-bit and 64-bit processors benefit especially from the function structure, which nevertheless is not oriented towards any particular platform. On the other hand, Maelstrom-0 improves upon its predecessor with techniques not available at the time Whirlpool was designed, providing for greater flexibility and a more robust security analysis. In a sense, we feel that Maelstrom-0 represents the state-of-the-art in the design of cryptographically secure hash functions; as such, besides being a new cryptographic hash function *per se*, it provides an initial assessment on what the minimum requirements for the Advanced Hash Standard might be, and also serves as a basis of comparison for the future proposals in terms of security, efficiency, and flexibility. It thus potentially constitutes not only a feedback, but also a valuable tool for NIST in the forthcoming "AHS" process.

The remainder of this paper is organised as follows. We provide an overall view of Maelstrom-0 and define the basic mathematical notation we use in section 2. A detailed formal description of the Maelstrom-0 components, structure, and the design rationale is explained in sections 3. through 6. Estimates of the computational efficiency in software and hardware are provided in section 7. We conclude by reviewing the overall strengths and advantages of the Maelstrom-0 primitive in section 8.

2. Maelstrom-0 in a nutshell

Maelstrom-0 processes its input iteratively, by chaining a particular compression func-

tion based on a dedicated block cipher designed according to the Wide Trail strategy [Daemen 1995, Rijmen 1997]. The basic mathematical operations involved (either linear or nonlinear) are all defined over binary finite fields. This makes it a conservative but arguably sound proposal [Biham 2005], and since all of its components and their interactions have now been quite extensively analysed in the literature, it is our firm opinion that the result is at once more secure, flexible, and considerably faster than WHIRLPOOL, from which it differs in several important points:

- The compression mode is Davies-Meyer [Menezes et al. 1999, Algorithm 9.42] rather than Miyaguchi-Preneel [Menezes et al. 1999, Algorithm 9.43] except for very short messages;
- The chaining scheme is a member of 3C/3C+ family [Gauravaram et al. 2006], which is arguably a more robust mechanism than plain Merkle-Damgård, or MD mode [Menezes et al. 1999, Algorithm 9.25];
- The underlying block cipher features 512-bit block size but 1024-bit key size, allowing for a doubled hash rate;
- The initial value (IV) is now a fine-tuning parameter that allows for controlled truncation of hash values, and thwarts a simple but annoying attack against certain common uses of hash functions.

In what follows we define and provide a rationale for each component of MAELSTROM-0 individually.

Given a message M, we denote its bit length by $|M|$. We will represent the field $\mathbb{F}_{2^4}$ as $\mathbb{F}_2[x]/(x^4 + x + 1)$, the field $\mathbb{F}_{2^8}$ as $\mathbb{F}_2[x]/(x^8 + x^4 + x^3 + x^2 + 1)$, and the field $\mathbb{F}_{2^{512}}$ as $\mathbb{F}_2[x]/(x^{512} + x^8 + x^5 + x^2 + 1)$. In the former two cases, the moduli are the first primitive polynomials of degrees respectively 4 and 8 listed in [Lidl and Niederreiter 1997], so in either case the polynomial x is a generator of the multiplicative group. In the latter case the modulus is the first irreducible polynomial of degree 512 in lexicographical order. A polynomial $p = \sum_{i=0}^{m-1} p_i \cdot x^i \in \mathbb{F}_2[x]$ will be denoted by the numerical value $\sum_{i=0}^{m-1} p_i \cdot 2^i = p(2)$, and written in hexadecimal notation.

3. The compression function

An iterated hash function processes messages of the form $M' = M_1 M_2 \ldots M_k$ where $|M_i| = m$, maintaining an internal state that is sequentially modified by each message block M_i. Let u_i denote the h-bit internal state after block M_i has been processed; for convenience we define $u_0 = \text{IV}$. A *compression function* is a computationally one-way mapping $f : \{0,1\}^m \times \{0,1\}^h \to \{0,1\}^h$ that updates the internal state through the rule $u_i = f(M_i, u_{i-1})$ for $i = 1, \ldots, k$. Since an iterative hash scheme is defined only for messages that can be partitioned into equally sized blocks, a general message M has to be padded to fit that structure before it is actually processed.

MAELSTROM-0 pads M with Merkle-Damgård strengthening [Menezes et al. 1999, sections 9.26 and 9.32], which consists of appending to M a single '1'-bit followed by as many '0'-bits as necessary to make a string whose bitlength is a multiple of $m = 2h$ minus $h/2$ bits, then completing the remaining $h/2$ bits with the binary representation of $|M|$, left-padded if necessary with '0' bits before the most significant bit of that representation. Hence the padded message has the form $M' = M_1 M_2 \ldots M_k$ where $|M_i| = m$ for $1 \leqslant i \leqslant k$, and the last $h/2$ bits of M_k contain the binary representation of $|M|$. For MAELSTROM-0 the internal state size is always $h = 512$ bits, and the block size is always $m = 2h = 1024$ bits.

To define a compression function for MAELSTROM-0 we distinguish between "short" and "long" messages in a precise meaning. A message M is *short* if $|M| < h$, i.e. if it fits a half padded block; otherwise it is *long*.

A short padded message M' simply undergoes one application of the Miyaguchi-Preneel construction, yielding the (untruncated) hash value $f(M', \mathrm{IV}) \equiv \mathcal{M}[M'](IV) \oplus IV \oplus M'$.

The compression function adopted for long messages is Davies-Meyer, whereby $f(M_i, u_{i-1}) \equiv \mathcal{M}[M_i](u_{i-1}) \oplus u_{i-1}$ for some block cipher $\mathcal{M}[K](B)$ that encrypts a data block B under a cipher key K.

Davies-Meyer is likely the most widely employed compression function in any concrete hash proposal. But more importantly, it is the only compression function among the 12 secure constructions analysed (in the context of a single-chain iterated hash function) by Black *et al.* [Black et al. 2002] that naturally allows the underlying block cipher to accept a key size different from the block size; all the remaining 11 functions XOR the key and the data block, thus forcing either truncation or padding to cope with the different sizes, and it is unclear to what extent truncation or padding might adversely affect the security analysis.

Quantitatively, the Miyaguchi-Preneel scheme is slightly more secure (by a roughly constant factor) than Davies-Meyer, but this difference is offset by the extra flexibility in the choice of the block cipher structure, which allows for faster hashing with the use of double-length keys. We leave it as an open problem whether the more robust chaining scheme adopted in MAELSTROM-0 compensates for the slight security difference between Davies-Meyer and Miyaguchi-Preneel. We will come back to this issue later.

4. Chaining scheme

Short messages are not subjected to any chaining mechanism in MAELSTROM-0. We therefore assume for the remaining of this section that the message is long. Let $M' = M_1 M_2 \dots M_k$ be the message to hash, padded as described above. The conventional Merkle-Damgård mode maintains a single chain of intermediate values of an internal h-bit state. Let u_i be the internal state value after block M_i has been processed, i.e. $u_i = f(M_i, u_{i-1})$ with $u_0 = \mathrm{IV}$. In that mode, the (untruncated) hash value would be simply the final state u_k.

The 3C family of hash modes [Gauravaram et al. 2006], designed to display increase resistance against multiblock collision attacks, generalizes the Merkle-Damgård construction by maintaining two or more chains $u_i, s_i, t_i, \dots$ instead of only one. The extra chains are transformed into an extra block $M_{k+1} = g(s_k, t_k, \dots)$ for some function g, and this block is processed to update the first chain one last time, producing the (untruncated) hash value as the post-final state $u_{k+1} = f(M_{k+1}, u_k)$. This setting also prevents length-extension attacks [Menezes et al. 1999, example 9.64], as we will show at the end of this section. The second chain s_i, present in all variants of the 3C family, is a simple XOR accumulation of all intermediate compression function outputs, recursively defined as $s_0 = 0$, $s_i = u_i \oplus s_{i-1}$. The overhead to maintain this chain is thus very low compared with the cost of each compression function invocation. The 3C+ subfamily of 3C maintains a third chain t_i consisting of a different combination of the u_i states.

MAELSTROM-0 uses a linear feedback shift register (LFSR) to establish its variant of the 3C+ family, which we call 3CM. Its third chain is recursively defined as $t_0 = IV$, $t_i = u_i \oplus \zeta(t_{i-1})$ where ζ is a shift update defined by mapping its argument from $\{0,1\}^h$ to an element of $\mathbb{F}_{2^h}$, multiplying it by the primitive element $x^8 \in \mathbb{F}^*_{2^h}$, and then mapping back to $\{0,1\}^h$. Like the basic 3C+ scheme, MAELSTROM-0 takes the simplest choice for the function $g(s_k, t_k)$, namely, the concatenation of s_k and t_k.

The factor x^8 was chosen so that ζ can be implemented as a one-byte left-shift and a one-byte XOR applied to (the $\mathbb{F}_{2^{512}}$ representation of) t_{i-1}. Specifically, let the formal argument $v = \sum_{j=0}^{511} v_j x^j$ of $\zeta(v)$ with $v_j \in \mathbb{F}_2$ be represented as $v = \sum_{j=0}^{63} w_j x^{8(63-j)}$ where $w_j(x) = \sum_{\ell=0}^{7} v_{511-\ell} x^{7-\ell}$. Defining the byte values $b_j = w_j(2)$, clearly v corresponds to the byte sequence $b = (b_0, b_1, \ldots, b_{62}, b_{63})$. Then $\zeta(v) = x^8 v$ corresponds to the sequence $b = (b_1, \ldots, b_{63} \oplus c_1[b_0], c_0[b_0])$, where the 16-bit value $c(b_0) \equiv (c_1[b_0] \ll 8) \oplus c_0[b_0]$ is given by $(b_0 \ll 8) \oplus (b_0 \ll 5) \oplus (b_0 \ll 2) \oplus b_0$, which can be precomputed as two 256-entry byte-valued tables $c_1[]$ and $c_0[]$. We will see that the ζ transform is reused in the key schedule of the dedicated block cipher $\mathcal{M}$.

Although the primitive element x might look a more obvious alternative, the choice of x^8 is meant to simplify the computation of the third chain in certain important platforms (e.g. 8-bit smart cards, which naturally process data bytewise, and SSE2 128-bit registers, which have 8-bit shift instructions but lack straightforward 1-bit shifts), while keeping the overhead on other platforms, either hardware or software, as low as possible.

We conjectured above that the slight security difference between the Miyaguchi-Preneel and the Davies-Meyer compression functions that shows up in the context of plain Merkle-Damgård chaining may be compensated by the adoption of 3CM in MAELSTROM-0. That this is a real possibility is illustrated by the fact that the 3C family contains three-chain members (including 3CM) with increased resistance against multiblock collision attacks even if the underlying compression function is not collision-resistant.

The reason why omitting formal chaining when hashing short messages does not incur susceptibility to length-extension attacks is the following. Given a short message M, its hash value is $u_1 = f(M||1||0^*|||M|)$; the attacker is assumed to know u_1 and $|M|$ but not M itself. Then any message of form $M^+ = M||1||0 * |||M|||X$ for arbitrary X will be padded to $M||1||0^*|||M|||X||1||0^*|||M^+|)$ and its hash value will be $u_3^+ = f(u_2^+ \oplus u_1^+, u_2^+ \oplus \zeta(u_1^+) \oplus \zeta^2(IV), u_2^+)$ with $u_2^+ = f(X||1||0^*|||M^+|, u_1^+)$ and $u_1^+ = u_1$, which can be computed from X, u_1 and $|M|$ alone. Clearly all that is needed to thwart this extension attack is to ensure that the attacker cannot deduce u_1^+ from u_1 and $|M|$; since M is unknown to the attacker, for $|M| < h$ this goal can be easily achieved by redefining the hash value to be $u_1 \oplus (M||1||0^*)$, which means using Miyaguchi-Preneel rather than Davies-Meyer.

5. The initial value

Last, but of course not least, we must define the initial value IV used by the first and third chains. The immediate predecessor of MAELSTROM-0, WHIRLPOOL, adopted the simplest possible choice, IV = 0^{512}, but made no provision for "spicing" this value if the hash value was intended to be truncated. For comparison, SHA-224 and SHA-256 (resp. SHA-384 and SHA-512) are essentially the same algorithm with different initial values, truncating the result from 256 to 224 (resp. from 512 to 384) bits at the end of the hashing process.

MAELSTROM-0 offers the additional flexibility of defining a different IV not only for plain truncation to any desired size, but for a general reduction modulo q for some integer $q \leqslant 2^h$, namely, by defining $IV \equiv q \bmod 2^h$ (note that full-length hash uses $IV \equiv 0^h$). This flexibility is desirable whenever the hash value H is used in a cryptosystem as an exponent or scalar factor, since the actual hash value is not H but rather $H \bmod q$, which may make the scheme susceptible to an attack due to Vaudenay [Vaudenay 1996]. An alternative is to define a rule to select q and other system parameters, as is the case of (EC)DSA [NIST 2000]; however, this creates the additional overhead of checking the selection rule for both the signer and the verifier.

Vaudenay's attack is based on the assumption that the hash function is independent of the desired q, which can then be chosen to cause a collision of the form $H(M) \equiv H(M') \pmod q$ (just vary M' until $H(M) - H(M')$ is a prime, and take this value as q). Adopting the convention that IV $= q$ thwarts this attack since M' would have to be a preimage of $H(M) \pm q$, which we assume to be as difficult as finding a preimage of the basic hash function (regardless of the IV).

A conceivable drawback of this IV convention is that a message that has to be repeatedly signed with varying q cannot be hashed only once, but we deem this scenario to be a very minor restriction since multiple or aggregate signatures most often share the same public parameters including q, or even globally defined default parameters as suggested in [NIST 2000]. We stress that the proposed IV convention remains useful under such circumstances due to its potential to reduce parameter checking overheads.

6. The underlying block cipher

MAELSTROM-0 is built upon an iterated block cipher $\mathcal{M}$ that only differs from the W underlying WHIRLPOOL in the key schedule, which takes 1024-bit rather than 512-bit keys. In simple terms, each round of $\mathcal{M}$ is a composition of a nonlinear layer, two linear layers, and an affine key addition layer, all of them operation on the internal 512-bit state. Correspondingly, each step of the key schedule consists of an affine 1024-bit key evolution transform and a nonlinear 512-bit subkey extraction function.

We briefly recall the basic concepts on which $\mathcal{M}$ is based, then describe the cipher in terms of its component transforms, designed according to the Wide Trail strategy [Daemen 1995, Rijmen 1997].

6.1. Preliminaries

A product of m distinct Boolean variables is called an m-th order product of the variables. Every Boolean function $f : (\mathbb{F}_2)^n \to \mathbb{F}_2$ can be written as a sum over $\mathbb{F}_2$ of distinct m-order products of its arguments, $0 \leqslant m \leqslant n$; this is called the algebraic normal form of f. The *nonlinear order* of f, denoted $\nu(f)$, is the maximum order of the terms appearing in its algebraic normal form. A *linear* Boolean function is a Boolean function of nonlinear order 1, i.e. its algebraic normal form only involves isolated arguments. Given $\alpha \in (\mathbb{F}_2)^n$, we denote by $l_\alpha : (\mathbb{F}_2)^n \to \mathbb{F}_2$ the linear Boolean function consisting of the sum of the argument bits selected by the bits of α:

$$l_\alpha(x) \equiv \bigoplus_{i=0}^{n-1} \alpha_i \cdot x_i.$$

A mapping $S : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}, x \mapsto S[x]$, is called a *substitution box,* or S-box for short. An S-box can also be viewed as a mapping $S : (\mathbb{F}_2)^n \to (\mathbb{F}_2)^n$ and therefore described in terms of its component Boolean functions $s_i : (\mathbb{F}_2)^n \to \mathbb{F}_2$, $0 \leqslant i \leqslant n-1$, i.e. $S[x] = (s_0(x), \ldots, s_{n-1}(x))$.

The *nonlinear order* of an S-box S, denoted ν_S, is the minimum nonlinear order over all linear combinations of the components of S:

$$\nu_S \equiv \min_{\alpha \in (\mathbb{F}_2)^n} \{\nu(l_\alpha \circ S)\}.$$

The δ*-parameter* of an S-box S is defined as

$$\delta_S \equiv 2^{-n} \cdot \max_{a \neq 0,\, b} \#\{c \in \mathbb{F}_{2^n} \mid S[c \oplus a] \oplus S[c] = b\}.$$

The value $2^n \cdot \delta$ is called the *differential uniformity* of S.

The *correlation* $c(f, g)$ between two Boolean functions f and g is defined as:

$$c(f, g) \equiv 2^{1-n} \cdot \#\{x \mid f(x) = g(x)\} - 1.$$

The extreme value (i.e. either the minimum or the maximum, whichever is larger in absolute value) of the correlation between linear functions of input bits and linear functions of output bits of S is called the *bias* of S. The λ*-parameter* of an S-box S is defined as the absolute value of the bias:

$$\lambda_S \equiv \max_{(i,j) \neq (0,0)} |c(l_i, l_j \circ S)|.$$

The Hamming distance between two vectors u and v from the n-dimensional vector space $(\mathbb{F}_{2^p})^n$ is the number of coordinates where u and v differ. The Hamming weight $\mathrm{w_h}(a)$ of an element $a \in (\mathbb{F}_{2^p})^n$ is the Hamming distance between a and the null vector of $(\mathbb{F}_{2^p})^n$, i.e. the number of nonzero components of a. A *linear* $[n, k, d]$ *code* over $\mathbb{F}_{2^p}$ is a k-dimensional subspace of the vector space $(\mathbb{F}_{2^p})^n$, where the Hamming distance between any two distinct subspace vectors is at least d (and d is the largest number with this property). A *generator matrix* G for a linear $[n, k, d]$ code C is a $k \times n$ matrix whose rows form a basis for C. A generator matrix is in *echelon* or *standard* form if it has the form $G = [I_{k\times k} \mid A_{k\times(n-k)}]$, where $I_{k\times k}$ is the identity matrix of order k. We write simply $G = [I \mid A]$ omitting the indices wherever the matrix dimensions are irrelevant for the discussion, or clear from the context. Linear $[n, k, d]$ codes obey the *Singleton bound*: $d \leqslant n - k + 1$. A code that meets the bound, i.e. $d = n - k + 1$, is called a *maximal distance separable* (MDS) code. A linear $[n, k, d]$ code C with generator matrix $G = [I_{k\times k} \mid A_{k\times(n-k)}]$ is MDS if, and only if, every square submatrix formed from rows and columns of A is nonsingular (cf. [MacWilliams and Sloane 1977, chapter 11, § 4, theorem 8]).

The *branch number* $\mathcal{B}$ of a linear mapping $\theta : (\mathbb{F}_{2^p})^k \to (\mathbb{F}_{2^p})^m$ is defined as

$$\mathcal{B}(\theta) \equiv \min_{a \neq 0} \{\mathrm{w_h}(a) + \mathrm{w_h}(\theta(a))\}.$$

Given a $[k + m, k, d]$ linear code over $\mathbb{F}_{2^p}$ with generator matrix $G = [I_{k\times k} \mid M_{k\times m}]$, the linear mapping $\theta : (\mathbb{F}_{2^p})^k \to (\mathbb{F}_{2^p})^m$ defined by $\theta(a) = a \cdot M$ has branch number $\mathcal{B}(\theta) = d$; if the code is MDS, such a mapping is called an *optimal diffusion mapping* [Rijmen 1997].

Table 1. The WHIRLPOOL S-box

	00_x	01_x	02_x	03_x	04_x	05_x	06_x	07_x	08_x	09_x	$0A_x$	$0B_x$	$0C_x$	$0D_x$	$0E_x$	$0F_x$
00_x	18_x	23_x	$C6_x$	$E8_x$	87_x	$B8_x$	01_x	$4F_x$	36_x	$A6_x$	$D2_x$	$F5_x$	79_x	$6F_x$	91_x	52_x
10_x	60_x	BC_x	$9B_x$	$8E_x$	$A3_x$	$0C_x$	$7B_x$	35_x	$1D_x$	$E0_x$	$D7_x$	$C2_x$	$2E_x$	$4B_x$	FE_x	57_x
20_x	15_x	77_x	37_x	$E5_x$	$9F_x$	$F0_x$	$4A_x$	DA_x	58_x	$C9_x$	29_x	$0A_x$	$B1_x$	$A0_x$	$6B_x$	85_x
30_x	BD_x	$5D_x$	10_x	$F4_x$	CB_x	$3E_x$	05_x	67_x	$E4_x$	27_x	41_x	$8B_x$	$A7_x$	$7D_x$	95_x	$D8_x$
40_x	FB_x	EE_x	$7C_x$	66_x	DD_x	17_x	47_x	$9E_x$	CA_x	$2D_x$	BF_x	07_x	AD_x	$5A_x$	83_x	33_x
50_x	63_x	02_x	AA_x	71_x	$C8_x$	19_x	49_x	$D9_x$	$F2_x$	$E3_x$	$5B_x$	88_x	$9A_x$	26_x	32_x	$B0_x$
60_x	$E9_x$	$0F_x$	$D5_x$	80_x	BE_x	CD_x	34_x	48_x	FF_x	$7A_x$	90_x	$5F_x$	20_x	68_x	$1A_x$	AE_x
70_x	$B4_x$	54_x	93_x	22_x	64_x	$F1_x$	73_x	12_x	40_x	08_x	$C3_x$	EC_x	DB_x	$A1_x$	$8D_x$	$3D_x$
80_x	97_x	00_x	CF_x	$2B_x$	76_x	82_x	$D6_x$	$1B_x$	$B5_x$	AF_x	$6A_x$	50_x	45_x	$F3_x$	30_x	EF_x
90_x	$3F_x$	55_x	$A2_x$	EA_x	65_x	BA_x	$2F_x$	$C0_x$	DE_x	$1C_x$	FD_x	$4D_x$	92_x	75_x	06_x	$8A_x$
$A0_x$	$B2_x$	$E6_x$	$0E_x$	$1F_x$	62_x	$D4_x$	$A8_x$	96_x	$F9_x$	$C5_x$	25_x	59_x	84_x	72_x	39_x	$4C_x$
$B0_x$	$5E_x$	78_x	38_x	$8C_x$	$D1_x$	$A5_x$	$E2_x$	61_x	$B3_x$	21_x	$9C_x$	$1E_x$	43_x	$C7_x$	FC_x	04_x
$C0_x$	51_x	99_x	$6D_x$	$0D_x$	FA_x	DF_x	$7E_x$	24_x	$3B_x$	AB_x	CE_x	11_x	$8F_x$	$4E_x$	$B7_x$	EB_x
$D0_x$	$3C_x$	81_x	94_x	$F7_x$	$B9_x$	13_x	$2C_x$	$D3_x$	$E7_x$	$6E_x$	$C4_x$	03_x	56_x	44_x	$7F_x$	$A9_x$
$E0_x$	$2A_x$	BB_x	$C1_x$	53_x	DC_x	$0B_x$	$9D_x$	$6C_x$	31_x	74_x	$F6_x$	46_x	AC_x	89_x	14_x	$E1_x$
$F0_x$	16_x	$3A_x$	69_x	09_x	70_x	$B6_x$	$D0_x$	ED_x	CC_x	42_x	98_x	$A4_x$	28_x	$5C_x$	$F8_x$	86_x

6.2. The nonlinear layer γ

Function $\gamma : \mathrm{GL}_8(\mathbb{F}_{2^8}) \to \mathrm{GL}_8(\mathbb{F}_{2^8})$ consists of the parallel application of a nonlinear S-box $S : \mathbb{F}_{2^8} \to \mathbb{F}_{2^8}$, $x \mapsto S[x]$ to all bytes of the argument individually:

$$\gamma(a) = b \Leftrightarrow b_{ij} = S[a_{ij}],\ 0 \leqslant i, j \leqslant 7.$$

MAELSTROM-0 uses the same S-box as its predecessor WHIRLPOOL (see [ISO/IEC 2004]). This S-box satisfies the following properties:

- The δ-parameter is 8×2^{-8}.
- The λ-parameter is 14×2^{-6}.
- The non-linear order ν is maximum, namely, 7.

The detailed microstructure and the rationale on the choice of that particular S-box is found in [Barreto and Rijmen 2000]. For completeness, we list that S-box in table 1.

6.3. The cyclical permutation π

The permutation $\pi : \mathrm{GL}_8(\mathbb{F}_{2^8}) \to \mathrm{GL}_8(\mathbb{F}_{2^8})$ cyclically shifts each column of its argument independently, so that column j is shifted downwards by j positions:

$$\pi(a) = b \Leftrightarrow b_{ij} = a_{(i-j) \bmod 8, j},\ 0 \leqslant i, j \leqslant 7.$$

The purpose of π is to disperse the bytes of each row among all rows.

6.4. The linear diffusion layer θ

The MAELSTROM-0 diffusion layer $\theta : \mathrm{GL}_8(\mathbb{F}_{2^8}) \to \mathrm{GL}_8(\mathbb{F}_{2^8})$ is directly inherited from its predecessor, WHIRLPOOL (see [ISO/IEC 2004]). It is a linear mapping based on the $[16, 8, 9]$ MDS code with generator matrix $G_C = [I \mid C]$ where C is the circulant matrix

$\mathrm{cir}(\mathtt{01}_x, \mathtt{01}_x, \mathtt{04}_x, \mathtt{01}_x, \mathtt{08}_x, \mathtt{05}_x, \mathtt{02}_x, \mathtt{09}_x)$, so that $\theta(a) = b \Leftrightarrow b = a \cdot C$. The effect of θ is to mix the bytes in each state row. The detailed rationale on this choice of θ is found in [Barreto and Rijmen 2000, revised May 2003]; for completeness, we list here the C matrix:

$$C = \begin{bmatrix} \mathtt{01}_x & \mathtt{01}_x & \mathtt{04}_x & \mathtt{01}_x & \mathtt{08}_x & \mathtt{05}_x & \mathtt{02}_x & \mathtt{09}_x \\ \mathtt{09}_x & \mathtt{01}_x & \mathtt{01}_x & \mathtt{04}_x & \mathtt{01}_x & \mathtt{08}_x & \mathtt{05}_x & \mathtt{02}_x \\ \mathtt{02}_x & \mathtt{09}_x & \mathtt{01}_x & \mathtt{01}_x & \mathtt{04}_x & \mathtt{01}_x & \mathtt{08}_x & \mathtt{05}_x \\ \mathtt{05}_x & \mathtt{02}_x & \mathtt{09}_x & \mathtt{01}_x & \mathtt{01}_x & \mathtt{04}_x & \mathtt{01}_x & \mathtt{08}_x \\ \mathtt{08}_x & \mathtt{05}_x & \mathtt{02}_x & \mathtt{09}_x & \mathtt{01}_x & \mathtt{01}_x & \mathtt{04}_x & \mathtt{01}_x \\ \mathtt{01}_x & \mathtt{08}_x & \mathtt{05}_x & \mathtt{02}_x & \mathtt{09}_x & \mathtt{01}_x & \mathtt{01}_x & \mathtt{04}_x \\ \mathtt{04}_x & \mathtt{01}_x & \mathtt{08}_x & \mathtt{05}_x & \mathtt{02}_x & \mathtt{09}_x & \mathtt{01}_x & \mathtt{01}_x \\ \mathtt{01}_x & \mathtt{04}_x & \mathtt{01}_x & \mathtt{08}_x & \mathtt{05}_x & \mathtt{02}_x & \mathtt{09}_x & \mathtt{01}_x \end{bmatrix}.$$

6.5. The key addition $\sigma[k]$

The affine key addition $\sigma[k] : \mathrm{GL}_8(\mathbb{F}_{2^8}) \rightarrow \mathrm{GL}_8(\mathbb{F}_{2^8})$ consists of the bitwise addition (XOR) of a key matrix $k \in \mathrm{GL}_8(\mathbb{F}_{2^8})$:

$$\sigma[k](a) = b \Leftrightarrow b_{ij} = a_{ij} \oplus k_{ij},\ 0 \leqslant i, j \leqslant 7.$$

6.6. The round function $\rho[k]$

The r-th round function is the composite mapping $\rho[k] : \mathrm{GL}_8(\mathbb{F}_{2^8}) \rightarrow \mathrm{GL}_8(\mathbb{F}_{2^8})$, parametrised by the key matrix $k \in \mathrm{GL}_8(\mathbb{F}_{2^8})$ and given by:

$$\rho[k] \equiv \sigma[k] \circ \theta \circ \pi \circ \gamma.$$

6.7. The round constants c^r

The round constant for the r-th round, $r \geqslant 0$, is a matrix $c^r \in \mathrm{GL}_8(\mathbb{F}_{2^8})$ defined as

$$\begin{aligned} c^r_{3j} &\equiv S[16r + j],\ 0 \leqslant j \leqslant 7; \\ c^r_{7j} &\equiv S[16r + 8 + j],\ 0 \leqslant j \leqslant 7; \\ c^r_{ij} &\equiv 0,\ i \neq 3, 7;\ 0 \leqslant j \leqslant 7. \end{aligned}$$

6.8. The key schedule

The key schedule expands the 1024-bit cipher key K onto a sequence of 512-bit round keys $K^0, \ldots, K^R$, with $K^r \in \mathrm{GL}_8(\mathbb{F}_{2^8})$. The bit sequence $K = (v_0, \ldots, v_{1023})$ is initially mapped to a column-vector $\mathcal{K}_{-1} \equiv (\kappa^{-2}, \kappa^{-1}) \in \mathbb{F}_{2^{512}} \times \mathbb{F}_{2^{512}}$ by the rules $\kappa_{-2} = \sum_{j=0}^{511} v_j x^{511-j}$, $\kappa_{-1} = \sum_{j=0}^{511} v_{j+512} x^{511-j}$.

At each step $s \geqslant 0$ the column-vector $\mathcal{K}_{s-1} = (\kappa_{2s-2}, \kappa_{2s-1})$ is transformed into a new vector by the *key evolution transform* $\mathcal{K}_s = (\kappa_{2s}, \kappa_{2s+1}) \equiv \mathcal{E} \cdot \mathcal{K}_{s-1} + C_s$, where $\mathcal{E} \in \mathrm{GL}_2(\mathbb{F}_{2^{512}})$ is defined as

$$\mathcal{E} \equiv \begin{bmatrix} 1 & 1 \\ x^8 & x^8 + 1 \end{bmatrix},$$

and the elements of the column-vector $C_s \equiv (c_{2s}, c_{2s+1}) \in \mathbb{F}_{2^{512}} \times \mathbb{F}_{2^{512}}$ correspond to the round constants c^{2s} and c^{2s+1}, respectively. This matrix generates a multiplicative

subgroup of $\mathrm{GL}_2(\mathbb{F}_{2^{512}})$ of size $O(2^{510})$. Besides, it and at least its first 100000 powers are all MDS.

Recall that multiplication by $x^8 \in \mathbb{F}_{2^{512}}$ is accomplished by the ζ transform defined in the context of the third 3CM chain. Thus each key evolution transform costs only one ζ invocation plus two $\mathbb{F}_{2^{512}}$ additions.

The actual round keys are given by $K^r = \psi(\kappa_r)$, where the *key extraction function* ψ consists of mapping κ_r to a matrix $\kappa^r \in \mathrm{GL}_8(\mathbb{F}_{2^8})$, then applying the S-box to the lines κ_i^r such that c_i^r is nontrivial (i.e. does not consist of a sequence of null $\mathbb{F}_{2^8}$ elements) obtaining the modified matrix $\tilde{\kappa}^r$, and finally applying the linear diffusion layer to those same lines exclusively, $K_i^r = \tilde{\kappa}_i^r \cdot C$, the remaining lines of K^r coinciding with the corresponding lines of κ^r.

Notice that, since R is even, the key evolution transform as described produces one extra element κ_{R+1}, which is discarded.

This scheduling scheme is lighter than that of Whirlpool, yet its nonlinearity is higher than that of the AES [NIST 2001]. The present choice ensures that it is not possible to choose a difference in the key input such that the input differences to all the S-box applications due to the key extraction function are zero, i.e. it prevents an attacker from completely avoiding the nonlinearity in the key schedule. To achieve this, the minimum total number of 8×8 S-box applications for a 1024-bit key along its evolution must be $1024/8 = 128$ (the actual number of S-box applications in the Maelstrom-0 key schedule is $16(R + 1) = 176$) and the mapping from the key to the S-boxes inputs must be bijective (as it indeed is, since the key evolution transform is affine and all powers of $\mathcal{E}$ are nonsingular).

6.9. The complete internal cipher $\mathcal{M}$

Given a sequence of functions $f_m, f_{m+1}, \ldots, f_{n-1}, f_n$, $m \leqslant n$, we use the notation $\bigcirc_{r=m}^{n} f_r \equiv f_m \circ f_{m+1} \circ \cdots \circ f_{n-1} \circ f_n$, and $\bigcirc_{m}^{r=n} f_r \equiv f_n \circ f_{n-1} \circ \cdots \circ f_{m+1} \circ f_m$; if $m > n$, both expressions stand for the identity mapping.

The dedicated R-round block cipher $\mathcal{M}[K] : \mathrm{GL}_8(\mathbb{F}_{2^8}) \times (\mathbb{F}_2)^{1024} \to \mathrm{GL}_8(\mathbb{F}_{2^8})$ is defined as

$$\mathcal{M}[K] = \left(\bigcirc_{1}^{r=R} \rho[K^r] \right) \circ \sigma[K^0],$$

where the round keys $K^0, \ldots, K^R$ are derived from K by the key schedule.

7. Efficiency

All implementation techniques proposed for components shared between Whirlpool and Maelstrom-0 remain valid [Barreto and Rijmen 2000, section 7]. Techniques specific to the few new components like the ζ transform have already been pointed out in the text.

Using the Whirlpool reference implementation in C as basis for a direct (non-optimized) implementation of Maelstrom-0 in IA-32 assembly language, we observe the results listed in table 2 for messages 1500 bytes long, the typical size of an IP packet. The benchmark platform is an Intel Pentium M 1.40 GHz processor. The core operation consists of the underlying cipher operating in the proper chaining mode and compression function. The full timings take buffering, padding, and other kinds of overhead into

Table 2. Efficiency (cycles/byte)

	WHIRLPOOL	MAELSTROM-0
Core	126	30
Full	157	103

Table 3. Extrapolation (cycles/byte)

SHA-512	WHIRLPOOL	SHA-256	MAELSTROM-0
40	36	22	14

account. All figures refer to the full 512-bit hash value to avoid extra costs needed for truncations or modular reductions.

We remark that, for the full timings, virtually the same buffer processing code was used for WHIRLPOOL and MAELSTROM-0, namely the reference code for WHIRLPOOL with a few changes to account for MAELSTROM-0's differing block size. When originally written, this code didn't require optimization as it wasn't considered a bottleneck for WHIRLPOOL, but the full timing figures indicate MAELSTROM-0 is severely affected by the inefficiency of this code; unfortunately, due to time constraints, we were unable to optimize it. We are confident that a properly optimized implementation of buffer processing will achieve considerably better figures.

Extrapolating to MAELSTROM-0 the best results on WHIRLPOOL available to us in software (exploiting the use of assembly language instructions on an IA-32 platform) [Nakajima and Matsui 2002], we anticipate the figures in table 3 for messages of the same size above or longer, concentrating on the core operation.

Similar extrapolation factors may be expected in throughput-oriented FPGA devices [Kitsos and Koufopavlou 2004] with only a modest increase in hardware resources (say, about 15%). That would produce a throughput of nearly 12.8 Gbit/s at a frequency of about 90 MHz, using roughly 6500 CLB slices. Although the expected number of CLB slices is nearly six times that of SHA-512, the corresponding throughput is almost 27 times higher for MAELSTROM-0. Alternatively, one may be interested in minimizing the required hardware resources [Pramstaller et al. 2006]; this might produce a throughput of nearly 1 Gbit/s at a frequency of about 130 MHz, using roughly 1700 CLB slices. This is less than the expected resources needed for either SHA-512 or SHA-384, yet the throughput is 2–3 times higher.

We stress that these are only estimates; a complete hardware project will include the still experimental (and therefore omitted) special handling of very short messages.

8. Conclusion

We have presented MAELSTROM-0, an evolution of WHIRLPOOL that incorporates the state-of-the-art in the design of cryptographically secure hash functions. The overall structure of MAELSTROM-0 makes it essentially platform-independent, yet facilitates exploiting the features of each particular environment. Still, its improved design makes it faster and arguably more robust than its predecessor and other comparable hash functions like SHA-512.

We hope that MAELSTROM-0 fulfils its role not merely as a new hash function, but as a feedback to NIST on which minimum requirements the future "Advanced Hash Standard" should satisfy, and also as a valuable comparison tool for AHS candidates in terms of security, efficiency, and flexibility.

References

Barreto, P. S. L. M. and Rijmen, V. (2000). The WHIRLPOOL hashing function. In *First Open NESSIE Workshop*, Leuven, Belgium. NESSIE Consortium.

Bellare, M. and Rogaway, P. (1993). Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, Fairfax, USA. ACM Press.

Biham, E. (2005). Recent advances in hash functions: The way to go. In *ECRYPT Conference on Hash Functions*. European Network of Excellence for Cryptology – ECRYPT. Invited talk, `http://www.cs.technion.ac.il/~biham/Reports/Slides/hash-func-krakow-2005.ps.gz`.

Black, J., Rogaway, P., and Shrimpton, T. (2002). Black-box analysis of the block-cipher-based hash-function constructions from PGV. In *Advances in Cryptology – Crypto'2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 320–335. Springer.

Boneh, D. and Boyen, X. (2004). Efficient selective-id secure identity based encryption without random oracles. In *Advances in Cryptology – Eurocrypt'2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 223–238. Springer.

Daemen, J. (1995). *Cipher and Hash Function Design*. Ph.D. thesis, Katholieke Universiteit Leuven.

Gauravaram, P., Millan, W., Dawson, E., and Viswanathan, K. (2006). Constructing secure hash functions by enhancing Merkle-Damgård construction. In *Australasian Conference on Information Security and Privacy – ACISP'2006*, Lecture Notes in Computer Science. Springer. To appear.

ISO/IEC (2004). *ISO/IEC 10118-3:2004 Standard – Information technology – Security techniques – Hash-functions – Part 3: Dedicated hash-functions*. ISO/IEC.

Kelsey, J. and Schneier, B. (2004). Second preimages on n-bit hash functions for much less than 2^n work. In *Advances in Cryptology – Eurocrypt'2004*, volume 3494 of *Lecture Notes in Computer Science*, pages 474–490. Springer.

Kitsos, P. and Koufopavlou, O. (2004). Efficient architecture and hardware implementation of the Whirlpool hash function. *IEEE Transactions on Consumer Electronics*, 50:208–213.

Klima, V. (2006). Tunnels in hash functions: MD5 collisions within a minute. Cryptology ePrint Archive, Report 2006/105. `http://eprint.iacr.org/2006/105`.

Kohno, T. and Kelsey, J. (2006). Herding hash functions and the nostradamus attack. In *Advances in Cryptology – Eurocrypt'2006*, Lecture Notes in Computer Science. Springer. To appear.

Lidl, R. and Niederreiter, H. (1997). *Finite Fields*. Number 20 in Encyclopedia of Mathematics and its Applications. Cambridge University Press, Cambridge, UK, 2nd edition.

MacWilliams, F. J. and Sloane, N. J. A. (1977). *The theory of error-correcting codes*, volume 16. North-Holland Mathematical Library.

Menezes, A. J., van Oorschot, P. C., and Vanstone, S. A. (1999). *Handbook of Applied Cryptography*. CRC Press, Boca Raton, USA.

Nakajima, J. and Matsui, M. (2002). Performance analysis and parallel implementation of dedicated hash functions. In *Advances in Cryptology – Eurocrypt'2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 165–180. Springer.

NIST (2000). *Federal Information Processing Standard (FIPS 186-2) – Digital Signature Standard (DSS)*. National Institute of Standards and Technology – NIST.

NIST (2001). *Federal Information Processing Standard (FIPS 197) – Advanced Encryption Standard (AES)*. National Institute of Standards and Technology – NIST.

Pramstaller, N., Rechberger, C., and Rijmen, V. (2006). A compact FPGA implementation of the hash function Whirlpool. In *International Symposium on Field Programmable Gate Arrays – FPGA'2006*, pages 159–166. ACM.

Rijmen, V. (1997). *Cryptanalysis and design of iterated block ciphers*. Ph.D. thesis, Katholieke Universiteit Leuven.

Rogaway, P. and Shrimpton, T. (2004). Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In *Fast Software Encryption – FSE'2004*, volume 3017 of *Lecture Notes in Computer Science*, pages 371–388. Springer.

Vaudenay, S. (1996). Hidden collisions on DSS. In *Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 83–88, Santa Barbara, USA. Advances in Cryptology – Crypto'96, Springer-Verlag.

Wang, X., Lai, X., Feng, D., Chen, H., and Yu, X. (2005a). Cryptanalysis of the hash functions MD4 and RIPEMD. In *Advances in Cryptology – Eurocrypt'2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 1–18. Springer.

Wang, X., Yin, Y. L., and Yu, H. (2005b). Efficient collision search attacks on SHA-0. In *Advances in Cryptology – Crypto'2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 1–16. Springer.

Wang, X. and Yu, H. (2005). How to break MD5 and other hash functions. In *Advances in Cryptology – Eurocrypt'2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35. Springer.