MAME: A Compression Function with Reduced Hardware Requirements ⋆

Hirotaka Yoshida[1], Dai Watanabe[1], Katsuyuki Okeya[1], Jun Kitahara[1], Hongjun Wu[2], Özgül Küçük[2], and Bart Preneel[2]

[1] Systems Development Laboratory, Hitachi, Ltd., 1099 Ohzenji, Asao-ku, Kawasaki-shi, Kanagawa-ken, 215-0013 Japan
[2] Katholieke Universiteit Leuven, Dept. ESAT/SCD-COSIC, Kasteelpark Arenberg 10, B–3001 Heverlee, Belgium

Abstract. This paper describes a new compression function, MAME designed for hardware-oriented hash functions which can be used in applications with reduced hardware requirements. MAME takes a 256-bit message block and a 256-bit chaining variable as input and produces a 256-bit output. In the light of recent attacks on MD5 and SHA-1, our design strategy is very conservative, and we show that our compression function is secure against various kinds of widely known attacks with very large security margins. The simple logical operations and the hardware efficient S-boxes are used to achieve a hardware implementation of MAME requiring only 8.1 Kgates on 0.18 μm technology.
Keywords: hash functions, compression functions, low-resource implementation

1 Introduction

Ubiquitous systems are becoming popular in a wide variety of applications such as secure supply-chain automation, proving genuineness of goods, etc. These applications have to deal with security problems such as confidentiality, more importantly, authentication and privacy. Thus, the importance of secure cryptographic techniques in ubiquitous systems has increased significantly. However, in order to develop secure ubiquitous systems, cryptographic algorithms must be implemented under restricted source environments, such as low-cost or low-power environments. As for authentication, what has been commonly used is cryptographic hash functions and their applications.

A cryptographic hash function is an algorithm that takes input strings of arbitrary (typically very large) length and maps these to short fixed length output strings. Most hash functions proposed so far are called *iterated* hash functions, which are constructed from a compression function. They work as follows. Let h

⋆ This work was supported in part by a consignment research from the National Institute on Information and Communications Technology (NiCT), Japan. This work was supported in part by the Concerted Research Action (GOA) Ambiorics 2005/11 of the Flemish Government.

be a compression function. The message m is padded to a multiple of the block length and subsequently divided into t blocks $M_1, \ldots, M_t$. Then the hash value is taken as H_t, where $H_i = h(H_{i-1}, M_i)$ and $H_0 = IV$ is called an initial value. The values $\{H_i\}$ are called the chaining variable.

A secure cryptographic hash function has to satisfy the following requirements:

- **preimage resistance**: it is computationally infeasible to find any input which hashes to any pre-specified output.
- **second preimage resistance**: it is computationally infeasible to find any second input which has the same output as any specified input.
- **collision resistance**: it is computationally infeasible to find a collision, i.e. two distinct inputs that hash to the same result.

For an ideal hash function with an m-bit output, finding a preimage or a second preimage requires about 2^m operations and the fastest way to find a collision is the birthday attack which needs approximately $2^{m/2}$ operations.

In order to satisfy those security requirements, most iterated hash functions use the Merkle-Damgård (MD) strengthening, which fixes IV and appends the message length to the message (to prevent extension attacks).

For the last years, there has been much progress in cryptanalysis of iterated hash functions. Attacks regarding collision resistance have been reported on most widely used iterated hash functions such as MD5 [26] and SHA-1 [22]. Meanwhile, iterated hash functions with the MD strengthening were revealed susceptible to several generic kinds of attacks (independent of the specific compression functions), such as the long second preimage attack [14, 15, 11] and the attack for finding multi-collisions [13].

We argue that the design strategy of hash functions and security evaluation methods must be revisited. As for security, we limit ourselves to collision resistance because the above second preimage attack still requires more complexities than the birthday attack does. One way of viewing the collision attacks mentioned the above is that these attacks essentially apply differential cryptanalysis [5] to find collisions. One could claim that a new hash function is only taken seriously if it is accompanied with evidence that it resists differential cryptanalysis.

In order to have a secure implementation, it is highly recommended to have a chaining value of 256 bits. Thus, an implementor could use SHA-256 [22]. However, SHA-256 has a large footprint, as it was designed for 32-bit processors using XORs, shifts, and modular addition.

This motivates us to develop a new compression function MAME to be used with any domain extension algorithm in order to build a light weight hash function. MAME accepts a chaining value of 256 bits and message blocks of 256 bits. The output size is 256 bits as well.

The outline of this paper is as follows. In Sect. 2, we give the specification of the MAME compression function. In Sect. 3, we explain our design strategy. In Sect. 4, we evaluate the security of MAME. We then discuss the performance issues in Sect. 5. Our conclusions are given in Sect. 6.

2 Specification

2.1 Notation

The specification uses the following notations:

$\oplus$ bit-wise exclusive-or	$\ggg n$ n-bit rotation to the right (32 bit register length)
$\|\|$ concatenation	$\lll n$ n-bit rotation to the left (32 bit register length)

In the remainder of this paper, we denote the message block by M and chaining variable by H respectively for simplicity.

2.2 The Algorithm of MAME

The MAME compression function denoted by h is constructed from the block cipher f_E defined below in the following manner known as the Matyas-Meyer-Oseas (MMO) mode ([19], pp 340), $h(H, M) = f_E(H, M) \oplus M$.

Overview of the Block Cipher The structure of the block cipher $f_E(\cdot, \cdot)$ is shown in Figure 1. The block size and the key size of the block cipher f_E are both

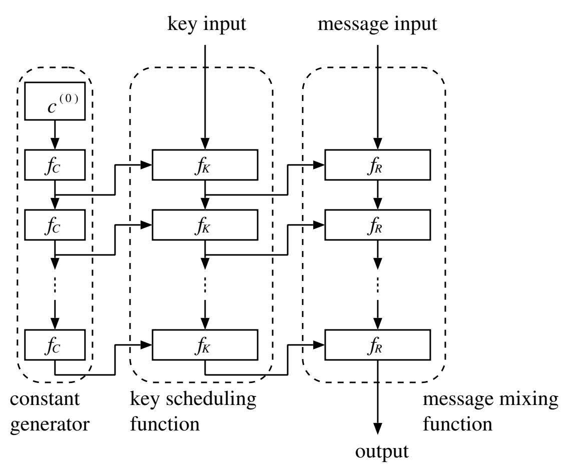

Fig. 1. The structure of the encryption function

256 bits. The cipher is a type 1 4-branch generalized Feistel network (GFN) [29] with 96 rounds. For implementation reasons, each of the branches is stored in two 32-bit words.

The cipher is broken down into three parts: the constant generation function, the key schedule function, the mixing function, each of which uses a sub-function iteratively. We denote the corresponding sub-functions by f_C, f_K, and f_R respectively.

The constant generator is initialized with the initial constant value $c^{(0)}$ and generates a round constant $C^{(r)}$ by iteratively applying the round constant generation function f_C. Together with the key, these round constants are used as input parameters to the key schedule function.

The round keys $K^{(r)}$ are calculated from the key by iteratively applying the round key generation function f_K. Each round of the key schedule function generates the round key $K^{(r)}$, which becomes the sub-key to the round function f_R. Finally, the mixing function uses the round function f_R iteratively to transform a message block into a ciphertext.

The Mixing Function The mixing function is defined by iterating the round function f_R. The input variables of f_R are $x_0, x_1, \ldots, x_7$, each a 32-bit word. The 256-bit plaintext is denoted by $P = (p_0, p_1, \ldots, p_7)$ and the 256-bit ciphertext by $E = (e_0, e_1, \ldots, e_7)$, the mixing function is defined in the following:

$$
\begin{aligned}
(x_0^{(0)}, x_1^{(0)}, \ldots, x_7^{(0)}) &= (p_0, p_1, \ldots, p_7), \\
(x_0^{(r)}, x_1^{(r)}, \ldots, x_7^{(r)}) &= f_R(x_0^{(r-1)}, x_1^{(r-1)}, \ldots, x_7^{(r-1)}), \quad 1 \leq r \leq 96, \\
(e_0, e_1, \ldots, e_7) &= (x_0^{(96)}, x_1^{(96)}, \ldots, x_7^{(96)}).
\end{aligned}
$$

The round function f_R consists of a key addition, a non-linear function F, and a word-wise permutation.

In the key addition operation, the round subkey $K^{(r)}$ from the key schedule is XORed with x_4. The F function is a non-linear transformation with 2-word input and 2-word output. The inputs of the F function are $x_4 \oplus K^{(r)}$ and x_5. The output of the F function is XORed with x_6, x_7. We denote the most significant word of the output of the F function by F_H, and the least significant word by F_L. Figure 2 describes the round function f_R which is defined as follows:

$$
\begin{aligned}
x_0^{(r)} &= x_6^{(r-1)} \oplus F(x_4^{(r-1)} \oplus K^{(r)}, x_5^{(r-1)})_H, \\
x_1^{(r)} &= x_7^{(r)} \oplus F(x_4^{(r-1)} \oplus K^{(r)}, x_5^{(r-1)})_L, \\
x_2^{(r)} &= x_0^{(r-1)}, x_3^{(r)} = x_1^{(r-1)}, x_4^{(r)} = x_2^{(r-1)}, \\
x_5^{(r)} &= x_3^{(r-1)}, x_6^{(r)} = x_4^{(r-1)}, x_7^{(r)} = x_5^{(r-1)}.
\end{aligned}
$$

We now describe how the F function works. We denote the input words to the F function by a_H , a_L. The F function consists of two layers, the S-box layer $\mathcal{S}$, and the linear diffusion layer $\mathcal{L}$. Each of the two layers is a transformation with a 64-bit input and a 64-bit output. The F function is the composition of these two transformations: $F = \mathcal{L} \circ \mathcal{S}$.

The S-box layer was designed for bit slice implementations. It uses a substitution table S with a 4-bit input and a 4-bit output, which is defined in the following:

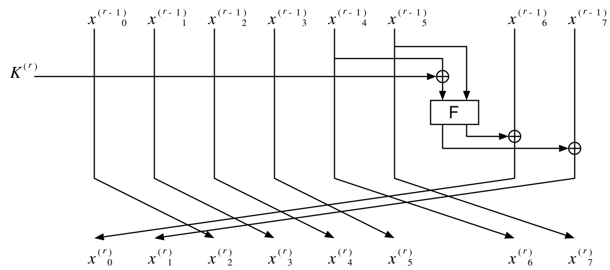

Fig. 2. The round function f_R

$$S[16] = \{\texttt{4}, \texttt{14}, \texttt{15}, \texttt{1}, \texttt{13}, \texttt{9}, \texttt{10}, \texttt{0}, \texttt{11}, \texttt{2}, \texttt{7}, \texttt{12}, \texttt{3}, \texttt{6}, \texttt{8}, \texttt{5}\}.$$

Denoting the output words by b_H, b_L, the S-box layer $\mathcal{S}$ is defined as follows:

$$b_{H,i+16}||b_{H,i}||b_{L,i+16}||b_{L,i} = S[a_{H,i+16}||a_{H,i}||a_{L,i+16}||a_{L,i}], \quad 0 \leq i < 16.$$

The linear diffusion layer $\mathcal{L}$ consists of cyclic rotations and XOR operations and is defined in the following:

$$b_L = b_L \oplus (b_H \lll 1), b_H = b_H \oplus (b_L \lll 3), b_L = b_L \oplus (b_H \lll 4),$$
$$b_H = b_H \oplus (b_L \lll 7), b_L = b_L \oplus (b_H \lll 8), b_H = b_H \oplus (b_L \lll 14).$$

The Key Schedule Function The round-key generation function f_K has the same structure as the f_R does. The difference is that f_K takes as an input the key instead of the plaintext and the subkeys are generated by the constant generation function (rather than the key schedule function).

$$\begin{aligned}
k_0^{(r)} &= k_6^{(r-1)} \oplus F(k_4^{(r-1)} \oplus C^{(r)}, k_5^{(r-1)})_H, \\
k_1^{(r)} &= k_7^{(r-1)} \oplus F(k_4^{(r-1)} \oplus C^{(r)}, k_5^{(r-1)})_L, \\
k_2^{(r)} &= k_0^{(r-1)}, k_3^{(r)} = k_1^{(r-1)}, k_4^{(r)} = k_2^{(r-1)}, \\
k_5^{(r)} &= k_3^{(r-1)}, k_6^{(r)} = k_4^{(r-1)}, k_7^{(r)} = k_5^{(r-1)}.
\end{aligned}$$

The r-th round-key $K^{(r)}$ is defined by $K^{(r)} = k_3^{(r)}$.

The Round Constants Generation The input $C^{(r)}$ to the round-key generation function f_K is generated sequentially from a fixed initial value $c^{(0)}$ by means of a simple linear transformation f_C. Starting from a fixed initial value $c^{(0)} = \texttt{0xcae1ac3f55054a96}$, The round-constant generation function f_C generates 64-bit variables $c^{(r)}$'s in the following manner:

$$\begin{aligned}
t_H||t_L &= f_L(c^{(r-1)}), \\
c^{(r)} &= t_L||t_H,
\end{aligned}$$

where f_L is a Linear feedback shift register (LFSR) defined by the polynomial $g(x)$ over GF(2) described in the Annex. The r-th round constant$C^{(r)}$ uses the 32 least significant bits of $c^{(r)}$.

3 Design Rationale

In our design of MAME, we aim to satisfy the following requirements:

- The security analysis should be simple in order to have confidence in the design.
- The security margins should be large enough to ensure long term security as a 256-bit hash function.
- It should be possible to achieve compact implementations in hardware.
- The software performance on general purpose machines should be good.

To meet these goals, we use the following design principles:

- Minimize the input/output length while achieving the required security.
- Use only known and understood building blocks such as XORs, which makes security assessment less complicated than with most previous hash functions, which use building blocks like arithmetic operations for which the full analysis is hard.
- Use a conservative estimation for the number of rounds, the choice of which considers attacks applying the input/output whitening techniques.

Parameter (input/output) Since the output length of the MAME is 256 bits, the message block length has to be at least the same size. From hardware implementation point of view, shorter input length implies that the number of required registers is smaller. Therefore we determined that the length of message block is 256 bits.

Structure We note that the SP structure is considered to be more hardware consuming than the Feistel structure. Thus, we have chosen to use Feistel over SP network. We have decided to use the unbalanced Feistel construction which allows for a more compact implementation without losing security (given sufficiently many rounds).

The Mode to Construct the Compression Function The use of the MMO mode allows the usage of the block cipher theory in understanding the security of MAME. The MMO mode is also more likely to withstand side channel attacks (e.g., when the hash function is used for key derivation) than the common Davies-Meyer [23].

The F Function The function F is the most significant component in the MAME. To reduce the area requirements, 16 small 4-bit-to-4-bit S-boxes are used in parallel. To increase the software performance, those 16 small S-boxes are identical to enable bit slice implementation. The linear diffusion layer uses simple rotations and XORs to reduce hardware and software complexity. Security-wise, we have picked the diffusion layer to have a branch number of 8.

As for the S-box, we adopted a function which is affine equivalent to the inversion function in $GF(2^4)$ for security reasons. We imposed the restriction that S has no fixed points. The S-box has the properties:

- Maximum differential and linear probabilities are 2^{-2}.
- The degree of the Boolean polynomial of every output bit is 3.
- The number of monomials of polynomial expression over $GF(2^4)$ is 14.

The Key Schedule Function and the Round Constants We use the encryption for function to derive the subkeys from the key, thus allowing for a large diffusion in the key schedule algorithm. We re-use the F function for the key schedule such that there is no extra hardware/memory requirements. The round constants introduce randomness, non-regularity, and asymmetry into the key schedule function. Thus, attacks which are based on the similarity of the rounds are easily prevented.

4 Security Analysis

Despite the fact that the most threatening attacks on hash functions at this moment are differential attacks, we evaluate the security of MAME with respect to various kinds of widely known attacks on block ciphers. These include not only differential attacks, but also linear attacks, higher order differential attacks, interpolation attacks, Square attacks.

The methods used to evaluate the compression function's resistance against these attacks are described below. In general, our analysis indicates that MAME has a large security margin against all of these attacks.

The motivation to analyze the MAME compression function with respect to attacks which do not immediately apply to hash functions as such, is that we want to ensure its security against future attacks which might borrow techniques from the field of block cipher cryptanalysis. Another motivation is that a number of block cipher based constructions, including the MMO mode, can be proved to be collision resistant if the underlying block cipher behaves as a pseudo-random function (see [25, 3]). The best way to verify this pseudo-randomness, is to apply block cipher analysis techniques to the core function f_E, and to see if this reveals any weakness or non-random behavior.

4.1 Differential and Linear Attacks

Considering the fact that the most successful attacks on hash functions are of differential nature, and that differential [5] and linear cryptanalysis [20] are two of the most powerful tools in block cipher cryptanalysis, we start our evaluation with an analysis of the resistance of f_E against differential and linear attacks.

In order to estimate the strength of f_E with respect to differential and linear attacks, we will try to compute upper bounds on the probabilities of differential and linear characteristics. As is commonly done in block cipher cryptanalysis, we will make abstraction of the exact differences or masks used in these characteristics, and just consider patterns of active S-boxes. More precisely, instead of analyzing how a full 256-bit difference (or mask) at the input of f_E propagates over different rounds, we only consider a 4×16-bit pattern whose bits indicate which of the 64 S-boxes in the first four rounds are active, and analyze to which patterns it can possibly propagate in all subsequent sequences of four consecutive rounds. In order to simplify notations in the remainder of this section, we will denote by $\tilde{x}$ the 16-bit pattern of active S-boxes that correspond with a 64-bit difference or mask x at the input of the function F. Once we have found a bound on the total number of active S-boxes in a characteristic, we can apply the following theorem:

Theorem 1. *Let D_{min} and L_{min} be lower bounds on the total number of active S-boxes in a differential/linear characteristic. Then, the maximum probabilities of the differential/linear characteristics are upper bounded by $p_s^{D_{min}}$ and $q_s^{L_{min}}$, respectively, where p_s and q_s denote the maximum differential/linear probabilities of the S-box, and are defined as follows:*

$$p_s = \max_{\Delta x \neq 0, \Delta y} \Pr[S(x) \oplus S(x \oplus \Delta x) = \Delta y]$$

$$q_s = \max_{\Gamma y \neq 0, \Gamma x} (2 \Pr[x \cdot \Gamma x = S(x) \cdot \Gamma y] - 1)^2$$

Hereafter, we only explain our method of evaluating the security against differential cryptanalysis as we can apply a similar method regarding linear cryptanalysis because of its duality to differential cryptanalysis [6].

In the case of MAME, we estimate the lower bounds of the number of active S-boxes by applying the Viterbi algorithm often used in the error correction codes. This algorithm considers a set of states where each of two states has distance and then, it exhaustively searches for paths with minimum distance. In our case, each state is defined as the intermediate state of f_E after certain round, the distance between a state at round r and a state at round $r+1$ is measured by the number of active S-box which has been increased through an application of r-th round.

However, we had a problem of too large memory requirement of 2^{64} in the the Viterbi algorithm.

To solve this, we consider the Hamming weight of a 64-bit difference rather than 64-bit difference itself. For such a 16-bit word $\tilde{x_i}$, $\mathrm{Ham}(\tilde{x})$ ranges from 0 to 16 and it can be represented as a 5-bit string. In the end, we manage to truncate the 64-bit space into the 20-bit space, which results in a practical usage of memory 2^{20} in carrying out the Viterbi algorithm.

Carrying out the Viterbi algorithm requires us to construct some table representing the propagation of the weight of the differences through F which is shown in Table 1 where 0 indicates the corresponding transition of Hamming weight of a difference is not possible, otherwise we put 1. For the row i, the column j, the element $a_{i,j}$ in Table 1 is determined in the following way:

- Case 1($i \leq 6$): For any 64-bit x such that $\mathrm{Ham}(x) = i$, compute $\mathrm{Ham}(\widetilde{\mathcal{L}(x)})$. If there exists x such that $\mathrm{Ham}(\widetilde{\mathcal{L}(x)}) = j$, then let $a_{i,j}$ be 1. Otherwise let $a_{i,j}$ be 0.
- Case 2($j \leq 6$): For any 64-bit y such that $\mathrm{Ham}(y) = j$, compute $\mathrm{Ham}(\widetilde{\mathcal{L}^{-1}(y)})$. If there exists y such that $\mathrm{Ham}(\widetilde{\mathcal{L}^{-1}(y)}) = i$, then let $a_{i,j}$ be 1. Otherwise let $a_{i,j}$ be 0.
- Case 3(Otherwise): Let $a_{i,j}$ be 1.

It took us several hours on a PC to perform experiments for each case. Table 1 tells us that the branch number of $\mathcal{L}$ is equal to 8, which is defined as follows:

Definition 1. *The branch number $B_{\mathcal{L}}$ of linear transformation $\mathcal{L}$ is defined by*

$$B_{\mathcal{L}} = \min_{x \neq 0}(Ham(\tilde{x}) + Ham(\widetilde{\mathcal{L}(x)})),$$

where we denote the Hamming weight of y by $Ham(y)$.

In the Viterbi algorithm, for an input difference, one standard way of computing the Hamming weight of the output difference is to use the branch number. In this way, we estimate that the value D_{min} is more than the required number 131 for MAME reduced to 80 rounds.

In order to improve the precision of estimation, we next used the Table 1 instead of the branch number when we performed the Viterbi algorithm. In addition, we captured information on how the weights of the differences change through two applications of F. We experimentally obtained information on how $\mathrm{Ham}(\widetilde{F \circ F(x)})$ behaves. This limits the possibilities for the output difference of the second application of F, compared to what we expect from the case of single application of F, the Table 1 [3]. During performing the Viterbi algorithm if an output difference of the first application is not influenced at XOR which is processed just after F, we can use the above information. In this way, the Viterbi algorithm found us some better result that the value D_{min} is 130 for MAME reduced to 58 rounds.

As for the linear attack, we obtain the same value for the branch number, 8. We perform similar approach to the case of differential cryptanalysis and we estimate that the value L_{min} is 129 for MAME reduced to 53 rounds.

From the above theorem, we estimate that the maximum differential/linear characteristic probabilities are upper bounded by 2^{-260} and 2^{-258}, respectively. It follows that there is no effective differential/linear characteristic for MAME reduced to 58 rounds.

[3] e.g.if $\mathrm{Ham}(\tilde{x}) = 3$ and $\mathrm{Ham}(\widetilde{F(x)})$=5), then $\mathrm{Ham}(\widetilde{F \circ F(x)})$=3 is not possible)

Table 1. Branch table for differential attacks

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	$16 = Ham(\mathcal{L}(\tilde{x}))$
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	1
2	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
3	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
4	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
5	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
6	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
7	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
8	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
9	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
10	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
11	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
12	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
13	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
14	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
15	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$16{=}Ham(\tilde{x})$	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

4.2 A Dedicated Differential Attack

We give an alternative description of the Feistel structure for ease of analysis. Denote the four 64-bit words of the internal state at round r as y_0^r, y_1^r, y_2^r and y_3^r, then the round function is given as follows:

$$\begin{aligned} y_0^r &= y_3^{r-1} \oplus F(y_2^{r-1} \oplus K^r); \\ y_1^r &= y_0^{r-1}; \\ y_2^r &= y_1^{r-1}; \\ y_3^r &= y_2^{r-1}; \end{aligned}$$

Suppose that $F(\Delta_0) = \Delta_1$, $F(\Delta_1) = \Delta_2$, $F(\Delta_2) = \Delta_3$ and $F(\Delta_3) = \Delta_0$ with probability p_0, p_1, p_2 and p_3, respectively. We obtain the 15-round difference propagation as shown in Table 2.

The probability for the 15-round differential path is $p_0^3 \times p_1^2 \times p_2^2 \times p_3$. The probability for the next 15-round differential is $p_1^3 \times p_2^2 \times p_3^2 \times p_0$. For 60 rounds, the differential probability is $p_0^8 \times p_1^8 \times p_2^8 \times p_3^8$.

We search for the differences with at most 7 active S-boxes for each difference. There are $2^{41.5}$ such differences. Searching through all these differences, but there is no differential relations $F(\Delta_0) = \Delta_1$, $F(\Delta_1) = \Delta_2$, $F(\Delta_2) = \Delta_3$ and $F(\Delta_3) = \Delta_0$. It shows clearly that there is no differential path with small number of active S-boxes.

Then we increase the number of active S-boxes to search for the differential paths. Let the number of active Sboxes in Δ_0 and Δ_2 be both 3. We allow the number of active S-boxes in Δ_1 and Δ_3 to be as large as 15. We searched all

Table 2. Difference propagation for 15 rounds

round	y_0	y_1	y_2	y_3
$r+0$	0	0	0	$\triangle_0$
$r+1$	$\triangle_0$	0	0	0
$r+2$	0	$\triangle_0$	0	0
$r+3$	0	0	$\triangle_0$	0
$r+4$	$\triangle_1$	0	0	$\triangle_0$
$r+5$	$\triangle_0$	$\triangle_1$	0	0
$r+6$	0	$\triangle_0$	$\triangle_1$	0
$r+7$	$\triangle_2$	0	$\triangle_0$	$\triangle_1$
$r+8$	0	$\triangle_2$	0	$\triangle_0$
$r+9$	$\triangle_0$	0	$\triangle_2$	0
$r+10$	$\triangle_3$	$\triangle_0$	0	$\triangle_2$
$r+11$	$\triangle_2$	$\triangle_3$	$\triangle_0$	0
$r+12$	$\triangle_1$	$\triangle_2$	$\triangle_3$	$\triangle_0$
$r+13$	0	$\triangle_1$	$\triangle_2$	$\triangle_3$
$r+14$	0	0	$\triangle_1$	$\triangle_2$
$r+15$	0	0	0	$\triangle_1$

these differences, and found that there are 14,045 differential paths. And the maximum number of the differential paths that starts from the same difference is only 6. Each set of $(\Delta_0, \Delta_1, \Delta_2, \Delta_3)$ involves at least 34 active S-boxes. The probability of a 60-round differential path is less than 2^{-700}, which shows MAME has a large security margin against this kind of attacks.

4.3 Higher Order Differential Attack

In the higher order differential attacks [16], the attacker constructs Boolean polynomial expression for the cipher to be attacked. In the encryption process, each bit of each intermediate state can be expressed as a Boolean polynomial in terms of bits of the plaintext.

The idea of the attack is that if the intermediate bits are expressed by Boolean polynomials of degree at least d, the $(d+1)$-th order differential in polynomial sense of the Boolean polynomial would be 0. Therefore if the value d is small enough, the attack would be feasible.

In the case of MAME, we found that every output bit of the S-box S can be expressed as a Boolean polynomial of degree 3 in terms of input bits. One naive approach for a higher order differential attack is to construct a Boolean polynomial of degree 256 for the 256-bit block cipher in MAME by assigning one variable for 1 bit. However, the attacker could construct a more simple expression of smaller degree by substituting 0 into certain variables, which makes various possibilities for the variables in the polynomial expression.

We performed experiments dealing with all of these possibilities in order to observe how the S-box applications increase the degree of Boolean functions as the number of rounds are increased. We confirmed that the degree of such polynomials for MAME with 21 rounds reaches to the required degree, which depends on how many variables the polynomial has. This prohibits higher order differential attacks on the full rounds of MAME.

4.4 Interpolation Attack

In the interpolation attacks [12], the attacker constructs polynomials (typically over some finite field) expressing the cipher to be attacked by using pairs of plaintext and ciphertext. The idea of the attack is that if the degree of constructed polynomial is small, required plaintexts and ciphertexts are a few in order to solve for the coefficient depending on the key in the polynomial.

In the case of MAME, the S-box S can be expressed for as a polynomial over $\mathrm{GF}(2^4)$. By applying the Lagrange interpolation technique, we found such a polynomial expression of degree 14 for S.

If we assign one variable for each 4 bits for MAME, we could construct a polynomial expression with 64 variables over $\mathrm{GF}(2^4)$. The attacker could construct a more simple expression by substituting 0 into certain variables.

We performed experiments dealing with such attacking scenario and we confirmed that the degree of such polynomials increases to more than 255. This prohibits interpolation attacks on more than 18 rounds of MAME.

4.5 Square Attack

The Square attack has been developed to evaluate the security of the byte-oriented ciphers such as Square and AES [7]. Here we analyse the block cipher in MAME by applying this technique. The attack introduces the following terms. The ith byte is *passive* if and only if values of all ith byte in the collection of texts are equal. The ith byte is *active* if and only if values of all ith byte in the collection of texts are different. The ith byte is *balanced* if and only if the sum of all ith byte is 0. The byte which is not categorized to be any of these bytes is called *unbalanced*. In the attack on reduced-round AES, staring from a collection of texts with one active byte, the attacker obtains balance bytes after several rounds, which result in constructing an distinguisher leading to a successful attack.

In the case of MAME, we make 64-bit words play the same role as bytes do in the Square attacks on reduced AES. In this way, we have 4 different word positions in each intermediate states hence we have 2^4 states for plaintext, depending on the positions where words are active or passive. We confirmed that starting from any of those states, any word becomes unbalanced after 17 rounds of MAME. Therefore we consider that the square attack is very unlikely to be feasible to the full round MAME.

4.6 Analysis of the Iterated Hash Function Based on MAME with the MD strengthening

In order to use MAME in practice, we specify certain iterated hash function based on MAME with the MD strengthening with the 256-bit initial vector $H_0 = (H_{0,0}, H_{0,1}, \ldots, H_{0,7})$ which is given in the following:

$$H_{0,0} = \texttt{0xbc18bf6d}, H_{0,1} = \texttt{0x369c955b}, H_{0,2} = \texttt{0xbb271cbc}, H_{0,3} = \texttt{0xdd66c368},$$
$$H_{0,4} = \texttt{0x356dba5b}, H_{0,5} = \texttt{0x33c00055}, H_{0,6} = \texttt{0x50d2320b}, H_{0,7} = \texttt{0x1c617e21}.$$

We investigate the security of the hash function against the collision attacks by Wang *et al.* In the collision attacks on MAME, choosing the message input to MAME corresponds to choosing the plaintext to the underlying block cipher. For any differential characteristic the attacker finds, its differential probability is upper bounded by 2^{-256}. The attacker next tries to build a system of equations for this characteristic which is called sufficient conditions and then tries to satisfy them by controlling the chaining variable input and the message block input. However, direct control over the chaining variable input should be very difficult because the key schedule input is the output of the previous application of MAME. Therefore, all the attacker can control should be 256 bits of plaintext with which we consider it is very difficult in order to fulfill the conditions. Therefore, we consider the attacks by Wang *et al* is very unlikely to be feasible to the MAME based hash function specified here.

4.7 Regularity Analysis of Reduced MAME

The simple design of MAME enables us to develop reduced versions keeping almost the whole design principles and primitives unchanged. We used this property to launch some experiments which are not possible to do on the real size. We believe that those analysis could help us to have a better understanding of hash functions based on this kind of construction.

There are two aims in this approach. The first one is to detect possible irregularities or differential anomalies in the reduced version which may indicate a security flaw in the design approach. The second one is to parameterize the security (against differential kind of attacks for example) to some properties of the primitives and to some parameters such as number of rounds.

MAME has a 4x4 bit S-box and uses an unbalanced Feistel network. We can form 32, 64 and 128 bits block size versions without changing those structures but reducing the word size and replacing the linear transformation. Our preliminary analysis focuses on 32 bits block size but it would also be interesting to analyze other sizes and correlations among them. From now on we will call the reduced version of MAME which uses a 32 bits block size MAME-32. A detailed specification of MAME-32 is given in appendix A.

The most basic collision-finding attack we might mount on a hash function is the so-called birthday attack. In a birthday attack we choose inputs to the hash function until we find two inputs that produce the same output. If the points

are chosen independently at random, birthday attack on a hash function h with range size r requires about $r^{1/2}$ trials to find a collision. But as it is pointed out in [2] the range points computed in the attack are uniformly distributed over R if and only if h is regular, meaning every range point has the same number of pre-images under h. We will use the balance measure for a hash function introduced in [2] for our regularity analysis of MAME-32:
Let $h : D \rightarrow R$ be a hash function where the range R contains $r \geq 2$ points $R_1, ..., R_r$. For $i = 1, ..., r$ let $h^{-1}(R_i) = \{x \in D | h(x) = R_i\}$ be the pre-image of R_i under h. Let $d_i = |h^{-1}(R_i)|$ and $d = |D|$ be the cardinality of this pre-image set and the domain respectively. Balance of h is defined as $\mu(h) = log_r[\frac{d^2}{d_1^2+...+d_r^2}]$. where $log_r()$ denotes the logarithm in base r. This is a real number between 0 and 1. Balance 1 indicates that the hash function is regular and balance 0 that it is a constant function, meaning as irregular as it can be. Let $C_h(q)$ be the probability that the birthday attack on hash function h succeeds in finding a collision in q trials. Then by [2]:$C_h(q) = \binom{q}{r} \frac{1}{r^{\mu(h)}}$, i.e., a collision is expected in about $r^{\mu(h)/2}$ trials. With this equation, performance of the birthday attack can be characterized in terms of the balance of the hash function h.
As we have pointed out before, the main difference between MAME and reduced versions is the word size and the diffusion layer. Therefore we calculated μ values for three different reduced versions such as with a linear transformation consisting of shifts and XOR (as in MAME), without a linear transformation, and finally with an MDS matrix.

Table 3. μ values for MAME-32 with different diffusion layers

Rounds	with no diffusion	similar diffusion	with mds matrix
8	0.962917	0.972480	0.972037
16	0.937925	0.968750	0.968750
32	0.937983	0.968750	0.968751
64	0.937815	0.968750	0.968750
96	0.938165	0.968750	0.968751

MAME-32 without a diffusion layer has lower balance. Note that for the 32 bits block size word size is 4 bits and F function inputs and outputs 8 bits. Half of the bits input to the S-boxes. Unlike the real size any weak diffusion would have greater impact on the following rounds. As we can observe from Table 3, regularity with a diffusion layer consisting of shifts and XORs does not differ from the one with an MDS matrix and it is reasonably high. MAME-32 has a block cipher structure for the first layer and uses Matyas-Meyer-Oseas mode for the second layer. Underlying block cipher as should be is 1-1 and onto, one might think that balancedness is mainly due to the second layer, however as can be seen from Table 3, it is effected from the diffusion layer.

5 Performance

5.1 Hardware Performance

The use of logical operations in the most part of the design allows us to achieve a hardware implementation of MAME requiring 8.1 Kgates on 0.18 μm technology. In our implementation, f_K and f_R share the same circuit and processing one round takes one cycle. We also implemented SHA-256 in the same environment as we did in the case of MAME. We here present our hardware implementation comparison of MAME with SHA-256 [22]. as shown in Table 4.

Table 4. Comparison of hardware implementation of MAME with SHA-256

Algorithm	Area (KGates)	throughput (Mbps)	Max Frequency (MHz)
MAME	8.1	440	333
SHA-256	18.0	2600	333
SHA-256 [10]	10.9	22.5	50

We note that the relatively slower throughput is not a real barrier in the case of a hash function aimed at low-end devices, as they are not expected to handle large amounts of data in any case.

5.2 Software Performance

We present our software implementations of MAME and SHA-256 on some microcomputer made by Renesas for smart cards. In software implementation, we partially unroll the round functions code to increase the speed and we take the approach described in [24] to achieve a bit slice implantation where S-box is transformed into 20 logical operations. We also implemented SHA-256 in the same environment as we did in the case of MAME. We present our software implementation comparison of MAME with SHA-256 as shown in Table 5.

Table 5. Comparison of software implementation of MAME with SHA-256

Algorithm	Time (ms)	RAM (Bytes)
MAME	49.4	96
SHA-256	31.4	128

Acknowledgements

The authors would like to thank Christophe De Cannière, Orr Dunkelman, Sebastiaan Indesteege, Joseph Lano, and Souradyuti Paul for useful discussions on this work and improving the editorial quality of this paper. We are also grateful to the anonymous referees for their valuable remarks.

6 Conclusion

We presented a new compression function, MAME designed for a hardware-oriented hash function. We make it clear what the design rational we adopt and evaluate its security applying techniques from block cipher analysis and confirm that there is no weakness in MAME. Our implementation shows some sort of compactness of MAME but this leaves room for further optimizations.

References

1. M.Bellare, R.Canetti, H.Krawczyk, "Keying Hash Functions for Message Authentication," Advances in Cryptology - CRYPTO 96, LNCS1109, (1996),1-15.
2. M.Bellare, T.Kohno, "Hash Function Balance and Its Impact on Birthday Attacks," Advances in Cryptology- Eurocrypt 2004, Springer-Verlag, LNCS 3027, (2004).
3. J. Black, P. Rogaway, and T. Shrimpton, "Black-box analysis of the block cipher-based hash-function constructions from PGV," Advances in Cryptology - CRYPTO 2002, Springer-Verlag, LNCS 2442, (2002), 320-335.
4. A. Biryukov, D. Wagner, "Advanced slide attacks," in *Proceedings of Eurocrypt 2000*, LNCS 1807, B. Preneel, Ed., Springer-Verlag, pp. 589–606, 2000.
5. E. Biham, A. Shamir, *Differential Cryptanalysis of the Data Encryption Standard*, Springer-Verlag, 1993.
6. F. Chabaud and S. Vaudenay, "Links between Differential and Linear Cryptanalysis," in *Proceedings of Eurocrypt '94*, LNCS 950, Springer-Verlag, pp. 356–365, 1995.
7. J. Daemen, L.R. Knudsen and V. Rijmen, "The block cipher Square," Fast Software Encryption, LNCS 1267, E. Biham, Ed., Springer-Verlag, 1997, pp. 149-165. Also available as http://www.esat.kuleuven.ac.be/rijmen/square/fse.ps.gz.
8. I. Damgård, "A design principle for hash functions," in *Proceedings of Crypto'89*, LNCS 435, G. Brassard, Ed., Springer-Verlag, pp. 416–427, 1990.
9. B. R. Gladman,
`http://fp.gladman.plus.com/cryptography_technology/`
10. M. Feldhofer, C. Rechberger, "A Case Against Currently Used Hash Functions in RFID Protocols", proceedings of On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops, LNCS 4227, pp. 372–381, Springer-Verlag, 2006.
11. J. J. Hoch, A. Shamir, "Breaking the ICE – Finding Multicollisions in Iterated Concatenated and Expanded (ICE) Hash Functions", Proceedings of Fast Software Encryption 2006, , LNCS 4047, pp. 179–194, 2006.
12. T. Jakobsen, L. R. Knudsen, "The interpolation attack on block ciphers," In Fast Software Encryption, Israel, LNCS 1267, pp. 28–40, Springer-Verlag, 1997
13. Antoine Joux, "Multicollisions in Iterated Hash Functions, Advances in Cryptology," Proceedings of CRYPTO 2004, LNCS 3152, pp. 306–316, Springer-Verlag, 2004.

14. J. Kelsey, B. Schneier, “Second Preimages on n-Bit Hash Functions for Much Less than 2n Work,” In Advances in Cryptology-Eurocrypt’2005, volume 3494 of Lecture Notes in Computer Science, pages 474–490, Springer-Verlag, 2005.
15. J. Kelsey and T. Kohno, “Herding hash functions and the Nostradamus attack,”. In Advances in Cryptology- EUROCRYPT 2006, volume 4004 of LNCS, pages 183–200, S. Vaudenay, Ed., Springer-Verlag, 2006.
16. L.R. Knudsen, “Truncated and Higher Order Differentials,” Proceedings of the Second Internation al Workshop on Fast Software Encryption, Leuven, Belgium, 1995, LNCS 1008, Springer, pp.196-211.
17. C.Kocher, J.Jaffe, B.Jun, “Differential Power Analysis,” Advances in Cryptology-CRYPTO 99, LNCS1666, (1999), 388-397.
18. K.Lemke, K.Schramm, C.Paar, “DPA on n-Bit Sized Boolean and Arithmetic Operations and Its Application to IDEA, RC6, and the HMAC Construction,” Cryptographic Hardware and Embedded Systems (CHES 2004), LNCS3156, (2004), 205-219.
19. A. J. Menezes, P. C. van Oorshot, and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.
20. M. Matsui, “Linear Cryptanalysis Method for DES cipher,” in Proceedings of EUROCRYPT’93, LNCS 765, pp.386-397, 1994.
21. T.S.Messerges, E.A.Dabbish, R.H.Sloan, “Investigations of Power Analysis Attacks on Smartcards,” USENIX Workshop on Smartcard Technology (1999).
22. National Institute of Standards and Technology, FIPS-180-2: “Secure Hash Standard (SHS),” August 2002.
23. K. Okeya, “Side Channel Attacks against HMACs based on Block-Cipher based Hash Functions,” *ACISP 2006 Conference, Proceedings*, pp. 317–329, 2006.
24. D. A. Osvik, “Speeding up Serpent,” *The 3rd AES Conference, Proceedings*, pp. 317–329, 2000.
25. B. Preneel, R. Govaerts, and J. Vandewalle, “Hash functions based on block ciphers: A synthetic approach,” Advanced in Cryptology, CRYPTO 93, Springer-Verlag, LNCS 773, (1994), 368-378.
26. R. Rivest, “The MD5 message-digest algorithm,” Request for Comments (RFC) 1321, Internet Activities Board, Internet Privacy Task Force,April 1992.
27. X. Wang, Y. L. Yin, H. Yu, “Finding collisions in the full SHA1,” in *Proceedings of CRYPTO 2005*, LNCS 3621, V. Shoup, Ed., Springer-Verlag, pp. 17–36, 2005.
28. X. Wang, X. Lai, D. Feng, H. Chen, and X. Yu, “Cryptanalysis of the Hash Functions MD4 and RIPEMD, in Proceedings of Eurocrypt 2005, LNCS 3494, R. Cramer, Ed., Springer-Verlag, pp. 1?18, 2005.
29. Y. Zheng, T. Matsumoto, H. Imai, “On the Construction of Block Ciphers Provably Secure and Not Relying on Any Unproved Hypotheses”, Advances in Cryptology, proceedings of CRYPTO 1989, LNCS 435, pp. 461–480, Springer-Verlag, 1989.

A Specifications for MAME-32

A detailed specification of MAME-32 is given as follows:

1. MDS matrix over $GF(2)[x]/x4 + x + 1$:

$$\begin{pmatrix} x+1 & x^2 \\ x^2+x & x^2+x+1 \end{pmatrix}$$

2. Initial value = 0x1b5b8cbd.
3. The constants are the same as 4 LSBs of the MAME-32.
4. The linear transformation layer:

$$b_L = b_L \oplus (b_H <<< 3); b_H = b_H \oplus (b_L <<< 2)$$

B Round constants

For the constant generation function, the following polynomial $g(x)$ over GF(2) defining the Linear feedback shift register (LFSR) f_L is given:

$$\begin{aligned} g(x) = {} & x^{63} + x^{62} + x^{58} + x^{55} + x^{54} + x^{52} + x^{50} + x^{49} + x^{46} + x^{43} \\ & + x^{40} + x^{38} + x^{37} + x^{35} + x^{34} + x^{30} + x^{28} + x^{26} + x^{24} \\ & + x^{23} + x^{22} + x^{18} + x^{17} + x^{12} + x^{11} + x^{10} + x^{7} + x^{3} + x^{2} + 1 \end{aligned}$$

Round constants $C^{(0)}, \ldots, C^{(95)}$, are shown as follows:

Table 6. Round constants

$C^{(0)} =$ 0x51151113,	$C^{(1)} =$ 0x3b4f5a2f,	$C^{(2)} =$ 0x2b0e343a,	$C^{(3)} =$ 0x46b151a6,
$C^{(4)} =$ 0xac38d0e9,	$C^{(5)} =$ 0xde130ff4,	$C^{(6)} =$ 0x1b6f7abf,	$C^{(7)} =$ 0xbc9a76bc,
$C^{(8)} =$ 0xc631d3e6,	$C^{(9)} =$ 0xf269daf1,	$C^{(10)} =$ 0xdc1106f5,	$C^{(11)} =$ 0xa6fd1bb3,
$C^{(12)} =$ 0x1f1e6ba2,	$C^{(13)} =$ 0x307857d6,	$C^{(14)} =$ 0x7c79ae88,	$C^{(15)} =$ 0xc1e15f59,
$C^{(16)} =$ 0x3530f34d,	$C^{(17)} =$ 0x68df0d12,	$C^{(18)} =$ 0x7f4ff42f,	$C^{(19)} =$ 0x67aa7d25,
$C^{(20)} =$ 0x9265a0cb,	$C^{(21)} =$ 0xf1f384e2,	$C^{(22)} =$ 0xe21aba37,	$C^{(23)} =$ 0x03185ae5,
$C^{(24)} =$ 0xe73098aa,	$C^{(25)} =$ 0xa7ed528f,	$C^{(26)} =$ 0x58142bc4,	$C^{(27)} =$ 0x34397327,
$C^{(28)} =$ 0xa486e67c,	$C^{(29)} =$ 0x7b69f586,	$C^{(30)} =$ 0x921b99f1,	$C^{(31)} =$ 0x29719f74,
$C^{(32)} =$ 0xe3e25ede,	$C^{(33)} =$ 0xa5c67dd1,	$C^{(34)} =$ 0x4b5f3214,	$C^{(35)} =$ 0x3c95ce5f,
$C^{(36)} =$ 0xe9aa813c,	$C^{(37)} =$ 0x59db0067,	$C^{(38)} =$ 0x627c4d9d,	$C^{(39)} =$ 0x083671eb,
$C^{(40)} =$ 0xe6ab4602,	$C^{(41)} =$ 0x8b55feb7,	$C^{(42)} =$ 0x5e7b5164,	$C^{(43)} =$ 0x86dbc3c7,
$C^{(44)} =$ 0xbd3b0cfc,	$C^{(45)} =$ 0xb0e33606,	$C^{(46)} =$ 0xf4ec33f0,	$C^{(47)} =$ 0xc38cd819,
$C^{(48)} =$ 0x176686ad,	$C^{(49)} =$ 0x61691012,	$C^{(50)} =$ 0xf61623af,	$C^{(51)} =$ 0x41720925,
$C^{(52)} =$ 0xb702fecb,	$C^{(53)} =$ 0x6a9254e2,	$C^{(54)} =$ 0x7787c237,	$C^{(55)} =$ 0x6e9f1ae5,
$C^{(56)} =$ 0xb14578ab,	$C^{(57)} =$ 0xd5261be2,	$C^{(58)} =$ 0x6e99dbb7,	$C^{(59)} =$ 0x904e26e5,
$C^{(60)} =$ 0xd53d1eaa,	$C^{(61)} =$ 0xeab4a28f,	$C^{(62)} =$ 0x902233c5,	$C^{(63)} =$ 0xc588fa4a,
$C^{(64)} =$ 0xeb04f60f,	$C^{(65)} =$ 0xd2f5a045,	$C^{(66)} =$ 0xc349a84b,	$C^{(67)} =$ 0x248cf163,
$C^{(68)} =$ 0x627cd15a,	$C^{(69)} =$ 0x39bffc97,	$C^{(70)} =$ 0x4d250c04,	$C^{(71)} =$ 0x4d73cb47,
$C^{(72)} =$ 0xf042797d,	$C^{(73)} =$ 0x5a955d6b,	$C^{(74)} =$ 0xae539583,	$C^{(75)} =$ 0x050f05da,
$C^{(76)} =$ 0x12c26f16,	$C^{(77)} =$ 0x143c1768,	$C^{(78)} =$ 0x4b09bc58,	$C^{(79)} =$ 0x50f05da1,
$C^{(80)} =$ 0xe8f0b80d,	$C^{(81)} =$ 0x2c9b06f3,	$C^{(82)} =$ 0xcc989042,	$C^{(83)} =$ 0x19e022d7,
$C^{(84)} =$ 0xf6b40864,	$C^{(85)} =$ 0xcc0cb247,	$C^{(86)} =$ 0x1e0668fd,	$C^{(87)} =$ 0x5f68b96a,
$C^{(88)} =$ 0xd3959aef,	$C^{(89)} =$ 0xb974acc5,	$C^{(90)} =$ 0x210c1bca,	$C^{(91)} =$ 0x4e5e8a0e,
$C^{(92)} =$ 0x84306f29,	$C^{(93)} =$ 0xfdac6154,	$C^{(94)} =$ 0xbb4d85bf,	$C^{(95)} =$ 0x3267cc3c.