25-9-2006

On the Internal Structure of ALPHA-MAC

J. Huang
University of Wollongong

Jennifer Seberry
University of Wollongong, jennie@uow.edu.au

Willy Susilo
University of Wollongong, wsusilo@uow.edu.au

On the Internal Structure of ALPHA-MAC

Abstract

ALPHA-MAC is a MAC function which uses the building blocks of AES. This paper studies the internal structure of this new design. First, we provide a method to find second preimages based on the assumption that a key or an intermediate value is known. The proposed searching algorithm exploits the algebraic properties of the underlying block cipher and needs to solve eight groups of linear functions to find a second preimage. Second, we show that our idea can also be used to find internal collisions under the same assumption. We do not make any claims that those findings in any way endanger the security of this MAC function. Our contribution is showing how algebraic properties of AES can be used for analysis of this MAC function.

Disciplines

Physical Sciences and Mathematics

Publication Details

This paper was originally published as Huang, J., Seberry, J., and Susilo, W., On the Internal Structure of ALPHA-MAC, Vietcrypt'06 International Conference on Cryptology, 2006, 271-285.

On the Internal Structure of Alpha-MAC

Jianyong Huang, Jennifer Seberry and Willy Susilo

University of Wollongong, Wollongong NSW 2522, Australia
{jyh33, jennie, wsusilo}@uow.edu.au

Abstract. Alpha-MAC is a MAC function which uses the building blocks of AES. This paper studies the internal structure of this new design. First, we provide a method to find second preimages based on the assumption that a key or an intermediate value is known. The proposed searching algorithm exploits the algebraic properties of the underlying block cipher and needs to solve eight groups of linear functions to find a second preimage. Second, we show that our idea can also be used to find internal collisions under the same assumption. We do not make any claims that those findings in any way endanger the security of this MAC function. Our contribution is showing how algebraic properties of AES can be used for analysis of this MAC function.

1 Introduction

Hash functions play an important role in many areas of cryptography. The building of hash functions has received extensive work over the years, for example, the design of MD4 [17], MD5 [18], SHA-0 [3] and SHA-1 [2]. On the other hand, the cryptanalysis of hash functions has been carried out by many researchers, for instance, recent attacks on MD4, MD5, SHA-0 and SHA-1 [6, 7, 10, 14, 19–22].

Message Authentication Codes (MACs) are keyed hash functions that provide message integrity by appending a cryptographic checksum to a message which is verifiable only by the intended recipient of the message. Message authentication is one of the most important ways of ensuring the integrity of information, and it has been used in many practical applications. MAC functions take a secret key and a message as input and generate a short digest as output. Many research groups have presented various approaches to construct MAC functions, for example, MAA [13], CBC-MAC [15], UMAC [9], MDx-MAC [16] and HMAC [4, 5].

The Alred [11] construction is a new MAC design approach presented at FSE 2005. Alpha-MAC [11] is a specific instance of the Alred construction with AES [1] as the underlying block cipher. The reason why AES was chosen as the underlying block cipher of the Alpha-MAC is because AES is efficient in hardware and software and it has withstood intense public scrutiny since its publication as Rijndael [12].

In this paper, we study the internal structure of the Alpha-MAC by employing the algebraic properties of AES and the structural features of the Alpha-MAC. First, we present a method to find second preimages of the Alpha-MAC

by solving eight groups of linear functions, based on the assumption that an authentication key or an intermediate value of this MAC is known. Each of these eight groups of linear functions contains two equations. We divide the second-preimage search algorithm into two steps: the Backwards-aNd-Forwards (BNF) search and the Backwards-aNd-Backwards (BNB) search. The BNF search provides an idea for extending 32-bit collisions to 128-bit collisions[1] by solving four groups of linear functions. Given a key (or an intermediate value) and one four-block message, the BNB search can generate another four-block message such that these two messages produce 32-bit collisions, which are a prerequisite for the BNF search. To do the BNB search, we need to solve another four groups of linear functions. By combining the BNB search with the BNF search, we can find second preimages of Alpha-MAC. Second, we show that the second-preimage finding method can also be used to generate internal collisions. The proposed collision search method can find two five-block messages such that they produce 128-bit collisions under a selected key (or a selected intermediate value).

This paper is organized as follows: Section 2 provides a description of the Alpha-MAC, and Section 3 presents the second-preimage search algorithm. Section 4 shows how to generate internal collisions and finally, Section 5 concludes this paper. Appendix A includes our experimental results.

2 A Brief Description of Alpha-MAC

Alpha-MAC [11] is a MAC function which uses the building blocks of AES. Similarly to AES, the Alpha-MAC supports keys of 128, 192 and 256 bits. The word length is 32 bits, and the injection layout places the 4 bytes of each message word $[m_0, m_1, m_2, m_3]$ into a 4×4 array. The format of the injection layout is shown as follows:

$$\begin{bmatrix} m_0 & 0 & m_1 & 0 \\ 0 & 0 & 0 & 0 \\ m_2 & 0 & m_3 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Like AES, the Alpha-MAC round function contains SubBytes (SB), ShiftRows (SR), MixColumns (MC) and AddRoundKey (ARK) , and the output of each injection layout acts as the corresponding 128-bit round key. The message padding method appends a single 1 followed by the minimum number of 0 bits such that the length of the result is a multiple of 32. In the initialization, the state is set to all zeros and AES is applied to the state. For every message word, the chaining method carries out an iteration, and each iteration maps the bits of the message word to an injection input. After that, a sequence of AES round functions are applied to the state, with the round keys replaced by the injection input. In the final transformation, AES is applied to the state. The MAC tag is the first l_m bits of the resulting final state. The length of l_m may have any value less than or equal to 128. The Alpha-MAC function is depicted in Figure 1.

[1] Here and in the rest of this paper "collisions" stands for "internal collisions"

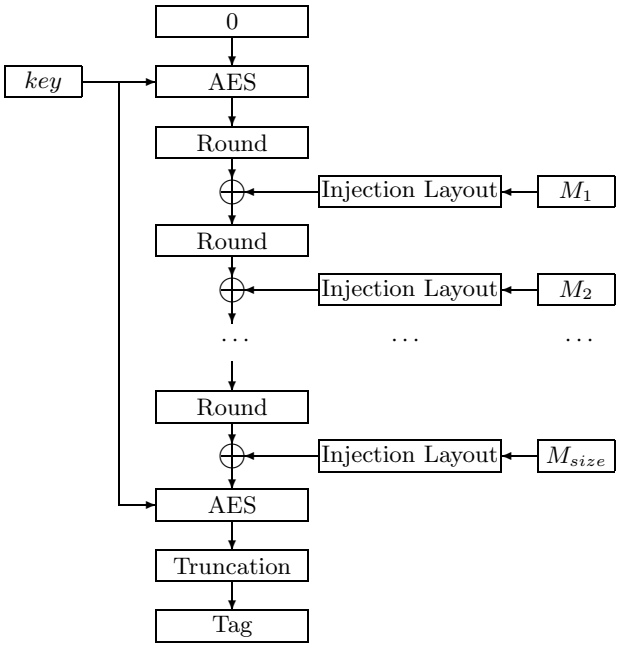

Fig. 1. ALPHA-MAC construction

3 The Second-Preimage Search Algorithm

The proposed second-preimage search algorithm aims to find a five-block second-preimage $\tilde{M}$ for a selected five-block message M, under a selected key (or a selected intermediate value). The assumption of this search is that we know two values: a selected key (or a selected intermediate value) and a selected five-block message M. The result of the search is that M and $\tilde{M}$ generate the same 128-bit value after five rounds of ALPHA-MAC iterations, under the selected key (or the selected intermediate value).

We use Figure 2 to illustrate the second-preimage search. Figure 2 depicts five consecutive rounds of the ALPHA-MAC for two different five-block messages M and $\tilde{M}$. We assume that we are able to select an intermediate value[2] [3] of the Round functions in some round (e.g., in Round $y-3$), and select five consecutive

[2] The intermediate value is:

$$\begin{bmatrix} a_0 & a_4 & a_8 & a_{12} \\ a_1 & a_5 & a_9 & a_{13} \\ a_2 & a_6 & a_{10} & a_{14} \\ a_3 & a_7 & a_{11} & a_{15} \end{bmatrix}.$$

[3] In the case of a selected key, for the sake of simplicity, we assume that (M_{y-3}, M_{y-2}, M_{y-1}, M_y, M_{y+1}) are the first five blocks of the selected message. Our search algorithm works without assuming that (M_{y-3}, M_{y-2}, M_{y-1}, M_y, M_{y+1}) are the first five blocks of the selected message.

message blocks $M(M_{y-3}, M_{y-2}, M_{y-1}, M_y, M_{y+1})$. Then we can find another five-block message $\tilde{M}(\tilde{M}_{y-3}, \tilde{M}_{y-2}, \tilde{M}_{y-1}, \tilde{M}_y, \tilde{M}_{y+1})$ such that these two five-block messages collide on 128 bits in Round $y+1$ after ARK.

The second-preimage search algorithm has the following form:

Known:	1. a selected key or a selected intermediate value. 2. a selected five-block message $M(M_{y-3}, M_{y-2}, M_{y-1}, M_y, M_{y+1})$.
Find:	another five-block message $\tilde{M}(\tilde{M}_{y-3}, \tilde{M}_{y-2}, \tilde{M}_{y-1}, \tilde{M}_y, \tilde{M}_{y+1})$ such that M and $\tilde{M}$ collide on 128 bits after ARK in Round $y+1$.
Method:	solve eight groups of linear functions. These eight groups of functions are named as (1), (2), (3), (4), (5), (6), (7) and (8) in this section.

The second-preimage search algorithm consists of two steps: the Backwards-aNd-Forwards search and the Backwards-aNd-Backwards search. The BNF search can extend 32-bit collisions to 128-bit collisions, given two messages M and $\tilde{M}$ which collide on 32 bits, namely Bytes s_4, s_{12}, s_6 and s_{14}, after MC in round y (see Figure 2). Given a key (or an intermediate value) and one four-block message, the BNB search is able to find another four-block message such that these two messages collide on Bytes s_4, s_{12}, s_6 and s_{14} after MC in Round y. The BNB search generates those 32-bit collisions which are required for the BNF search. By merging the BNB search with the BNF search, we can find second preimages of the Alpha-MAC.

3.1 The Backwards-aNd-Forwards Search

The Backwards-aNd-Forwards search has the following form:

Known:	1. a selected key or a selected intermediate value. 2. two four-block messages $M(M_{y-3}, M_{y-2}, M_{y-1}, M_y)$ and $\tilde{M}(\tilde{M}_{y-3}, \tilde{M}_{y-2}, \tilde{M}_{y-1}, \tilde{M}_y)$ colliding on 32 bits (Bytes s_4, s_{12}, s_6 and s_{14}) after MC in Round y.
Extend:	32-bit collisions to 128-bit collisions in Round $y+1$.
Method:	solve four groups of linear functions. These four groups of functions are numbered as (1), (2), (3) and (4) in this subsection.

The BNF search assumes that we are able to find two messages M and $\tilde{M}$, which collide on Bytes s_4, s_{12}, s_6 and s_{14} after MC in round y. Based on the algebraic property of the MixColumns transformation and the structure of Alpha-MAC, we can extend these 32-bit collisions to 128-bit collisions within three rounds by solving four groups of linear equations.

3.1.1 Extending 32-bit Collisions to 64-bit Collisions We use the differential XOR property [8] before and after the MixColumns transformation.

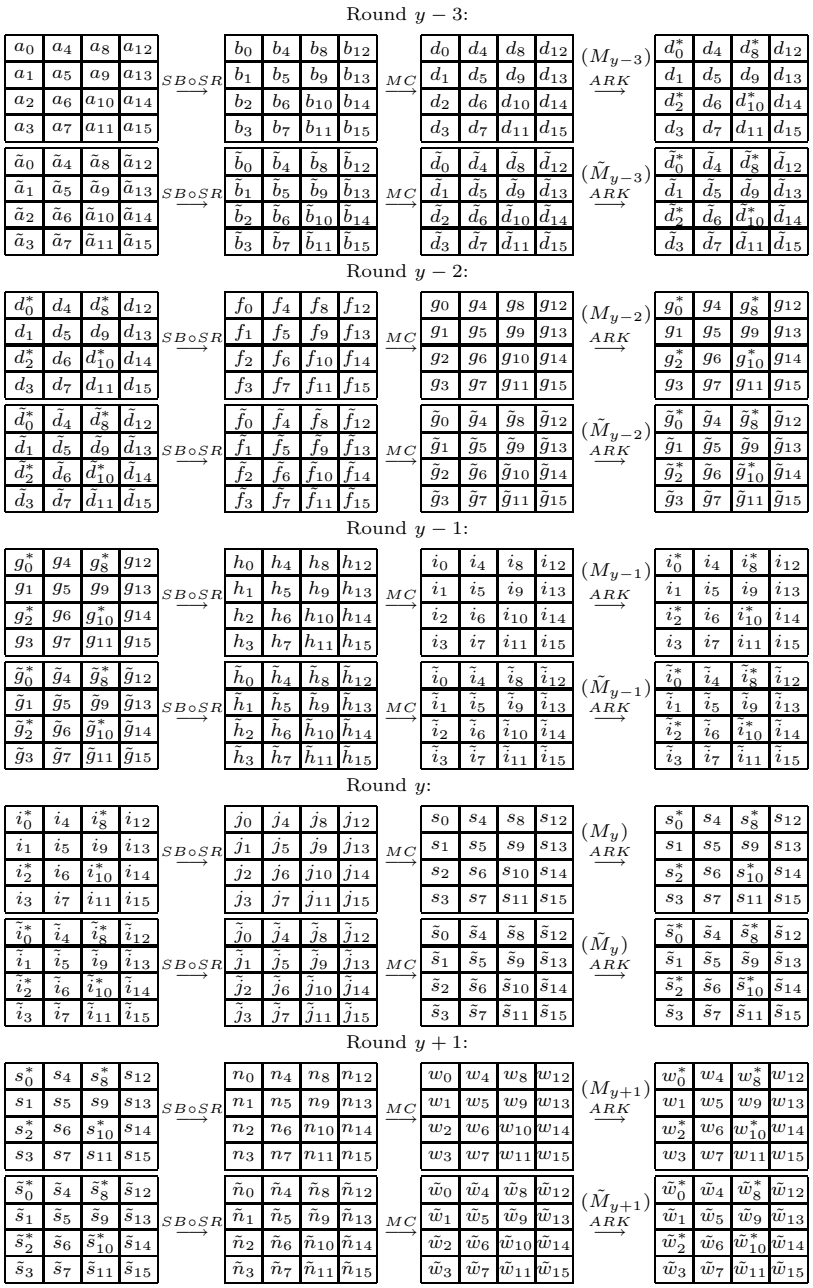

Fig. 2. The five-block collisions

In Round y before MC, by XORing those two intermediate values, we get the following result:

$$\begin{bmatrix} \tilde{j}_0 \oplus j_0 & \tilde{j}_4 \oplus j_4 & \tilde{j}_8 \oplus j_8 & \tilde{j}_{12} \oplus j_{12} \\ \tilde{j}_1 \oplus j_1 & \tilde{j}_5 \oplus j_5 & \tilde{j}_9 \oplus j_9 & \tilde{j}_{13} \oplus j_{13} \\ \tilde{j}_2 \oplus j_2 & \tilde{j}_6 \oplus j_6 & \tilde{j}_{10} \oplus j_{10} & \tilde{j}_{14} \oplus j_{14} \\ \tilde{j}_3 \oplus j_3 & \tilde{j}_7 \oplus j_7 & \tilde{j}_{11} \oplus j_{11} & \tilde{j}_{15} \oplus j_{15} \end{bmatrix} \xrightarrow{MC} \begin{bmatrix} ? & 0 & ? & 0 \\ 0 & \tilde{s}_5 \oplus s_5 & 0 & \tilde{s}_{13} \oplus s_{13} \\ ? & 0 & ? & 0 \\ 0 & \tilde{s}_7 \oplus s_7 & 0 & \tilde{s}_{15} \oplus s_{15} \end{bmatrix}.$$

Here, we use R (to replace $\tilde{j}_0 \oplus j_0$), S (to replace $\tilde{j}_8 \oplus j_8$), T (to replace $\tilde{j}_2 \oplus j_2$) and U (to replace $\tilde{j}_{10} \oplus j_{10}$) so that after the MC transformation in Round y, Bytes $\tilde{s}_1 \oplus s_1$, $\tilde{s}_3 \oplus s_3$, $\tilde{s}_9 \oplus s_9$ and $\tilde{s}_{11} \oplus s_{11}$ become zero. Now the question is "how to decide R, S, T and U". The answer is:

- There exists one and only one pair of (R, T) such that after MC, Bytes $\tilde{s}_1 \oplus s_1$ and $\tilde{s}_3 \oplus s_3$ are both zero.
- There exists one and only one pair of (S, U) such that after MC, Bytes $\tilde{s}_9 \oplus s_9$ and $\tilde{s}_{11} \oplus s_{11}$ are both zero.

According to the MC transformation, we have the following formula:

$$\begin{bmatrix} ? & 0 & ? & 0 \\ 0 & \tilde{s}_5 \oplus s_5 & 0 & \tilde{s}_{13} \oplus s_{13} \\ ? & 0 & ? & 0 \\ 0 & \tilde{s}_7 \oplus s_7 & 0 & \tilde{s}_{15} \oplus s_{15} \end{bmatrix} = \begin{bmatrix} 02\ 03\ 01\ 01 \\ 01\ 02\ 03\ 01 \\ 01\ 01\ 02\ 03 \\ 03\ 01\ 01\ 02 \end{bmatrix} \begin{bmatrix} R & \tilde{j}_4 \oplus j_4 & S & \tilde{j}_{12} \oplus j_{12} \\ \tilde{j}_1 \oplus j_1 & \tilde{j}_5 \oplus j_5 & \tilde{j}_9 \oplus j_9 & \tilde{j}_{13} \oplus j_{13} \\ T & \tilde{j}_6 \oplus j_6 & U & \tilde{j}_{14} \oplus j_{14} \\ \tilde{j}_3 \oplus j_3 & \tilde{j}_7 \oplus j_7 & \tilde{j}_{11} \oplus j_{11} & \tilde{j}_{15} \oplus j_{15} \end{bmatrix}.$$

To find out the values of (R, T) and (S, U), we need to solve the following two groups of equations.

$$\begin{cases} \begin{bmatrix} 01\ 02\ 03\ 01 \end{bmatrix} \begin{bmatrix} R \\ \tilde{j}_1 \oplus j_1 \\ T \\ \tilde{j}_3 \oplus j_3 \end{bmatrix} = 0 \\ \begin{bmatrix} 03\ 01\ 01\ 02 \end{bmatrix} \begin{bmatrix} R \\ \tilde{j}_1 \oplus j_1 \\ T \\ \tilde{j}_3 \oplus j_3 \end{bmatrix} = 0 \end{cases} \quad (1) \qquad \begin{cases} \begin{bmatrix} 01\ 02\ 03\ 01 \end{bmatrix} \begin{bmatrix} S \\ \tilde{j}_9 \oplus j_9 \\ U \\ \tilde{j}_{11} \oplus j_{11} \end{bmatrix} = 0 \\ \begin{bmatrix} 03\ 01\ 01\ 02 \end{bmatrix} \begin{bmatrix} S \\ \tilde{j}_9 \oplus j_9 \\ U \\ \tilde{j}_{11} \oplus j_{11} \end{bmatrix} = 0 \end{cases} \quad (2)$$

In the two equations in (1), there are two variables R and T, and therefore there exists one and only one pair of (R, T) to make these two equations hold simultaneously. Similarly, we can decide the values of S and U by solving the two equations in (2).

Once we get the values of R, S, T and U, message block $\tilde{M}_{y-1}$ can be constructed as follows:

1. Set the values of $\tilde{j}_0^{new}$, $\tilde{j}_8^{new}$, $\tilde{j}_2^{new}$ and $\tilde{j}_{10}^{new}$ as follows: $\tilde{j}_0^{new} = j_0 \oplus R$, $\tilde{j}_8^{new} = j_8 \oplus S$, $\tilde{j}_2^{new} = j_2 \oplus T$, and $\tilde{j}_{10}^{new} = j_{10} \oplus U$. Use $\tilde{j}_0^{new}$ to replace $\tilde{j}_0$, $\tilde{j}_8^{new}$ to replace $\tilde{j}_8$, $\tilde{j}_2^{new}$ to replace $\tilde{j}_2$, and $\tilde{j}_{10}^{new}$ to replace $\tilde{j}_{10}$.
2. Perform SR^{-1} (inverse ShiftRows) and SB^{-1} (inverse SubBytes). As SR^{-1} and SB^{-1} are permutation and substitution, they do not change the properties we have found. Now we have the outputs of ARK in Round $y-1$.
3. Compute the value of $\tilde{M}_{y-1}^{new}$ as follows:

$$\tilde{M}_{y-1}^{new} = (\tilde{j}_0^{new} \oplus \tilde{i}_0)||(\tilde{j}_8^{new} \oplus \tilde{i}_8)||(\tilde{j}_{10}^{new} \oplus \tilde{i}_2)||(\tilde{j}_2^{new} \oplus \tilde{i}_{10}).$$

   Use $\tilde{M}_{y-1}^{new}$ to replace $\tilde{M}_{y-1}$.

At this stage, two messages (M_{y-3}, M_{y-2}, M_{y-1}) and ($\tilde{M}_{y-3}$, $\tilde{M}_{y-2}$, $\tilde{M}_{y-1}^{new}$) collide on 64 bits (Bytes s_4, s_{12}, s_6, s_{14}, s_1, s_9, s_3 and s_{11}) in Round y after MC.

3.1.2 Extending 64-bit Collisions to 96-bit Collisions We only need to focus on Round y and Round $y+1$ to extend 64-bit collisions to 96-bit collisions. The idea is to choose message block $\tilde{M}_y$ to cancel out the differences between Bytes (s_5, s_{13}, s_7, s_{15}) and Bytes ($\tilde{s}_5$, $\tilde{s}_{13}$, $\tilde{s}_7$, $\tilde{s}_{15}$) in Round y. The method of choosing $\tilde{M}_y$ is exactly same as the method for constructing $\tilde{M}_{y-1}$ in Section 3.1.1.

By taking the outputs of ARK in Round y, we perform the SB and SR operations, and then XOR the results after SB and SR:

$$\begin{bmatrix} n_0 & n_4 & n_8 & n_{12} \\ n_1 & n_5 & n_9 & n_{13} \\ n_2 & n_6 & n_{10} & n_{14} \\ n_3 & n_7 & n_{11} & n_{15} \end{bmatrix} \oplus \begin{bmatrix} \tilde{n}_0 & n_4 & \tilde{n}_8 & n_{12} \\ \tilde{n}_1 & n_5 & \tilde{n}_9 & n_{13} \\ \tilde{n}_2 & n_6 & \tilde{n}_{10} & n_{14} \\ \tilde{n}_3 & n_7 & \tilde{n}_{11} & n_{15} \end{bmatrix} = \begin{bmatrix} n_0 \oplus \tilde{n}_0 & 0 & n_8 \oplus \tilde{n}_8 & 0 \\ n_1 \oplus \tilde{n}_1 & 0 & n_9 \oplus \tilde{n}_9 & 0 \\ n_2 \oplus \tilde{n}_2 & 0 & n_{10} \oplus \tilde{n}_{10} & 0 \\ n_3 \oplus \tilde{n}_3 & 0 & n_{11} \oplus \tilde{n}_{11} & 0 \end{bmatrix} \xrightarrow{MC} \begin{bmatrix} ? & 0 & ? & 0 \\ 0 & 0 & 0 & 0 \\ ? & 0 & ? & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Here we use π to replace $n_0 \oplus \tilde{n}_0$, ρ to replace $n_8 \oplus \tilde{n}_8$, ϕ to replace $n_2 \oplus \tilde{n}_2$ and ω to replace $n_{10} \oplus \tilde{n}_{10}$ so that after MixColumns in Round $y+1$, Bytes $w_1 \oplus \tilde{w}_1$, $w_9 \oplus \tilde{w}_9$, $w_3 \oplus \tilde{w}_3$ and $w_{11} \oplus \tilde{w}_{11}$ are zero:

$$\begin{bmatrix} \pi & 0 & \rho & 0 \\ n_1 \oplus \tilde{n}_1 & 0 & n_9 \oplus \tilde{n}_9 & 0 \\ \phi & 0 & \omega & 0 \\ n_3 \oplus \tilde{n}_3 & 0 & n_{11} \oplus \tilde{n}_{11} & 0 \end{bmatrix} \xrightarrow{MC} \begin{bmatrix} ? & 0 & ? & 0 \\ 0 & 0 & 0 & 0 \\ ? & 0 & ? & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Now the question is "how to decide π, ρ, ϕ and ω". The answer is:

– There exists one and only one pair of (π, ϕ) such that after MC, Bytes $w_1 \oplus \tilde{w}_1$ and $w_3 \oplus \tilde{w}_3$ are both zero. The values of (π, ϕ) can be decided by solving (3).

– There exists one and only one pair of (ρ, ω) such that after MC, Bytes $w_9 \oplus \tilde{w}_9$ and $w_{11} \oplus \tilde{w}_{11}$ are both zero. By solving (4), we get the values of (ρ, ω).

$$\begin{cases} \begin{bmatrix} 01 & 02 & 03 & 01 \end{bmatrix} \begin{bmatrix} \pi \\ n_1 \oplus \tilde{n}_1 \\ \phi \\ n_3 \oplus \tilde{n}_3 \end{bmatrix} = 0 \\ \begin{bmatrix} 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} \pi \\ n_1 \oplus \tilde{n}_1 \\ \phi \\ n_3 \oplus \tilde{n}_3 \end{bmatrix} = 0 \end{cases} \quad (3) \qquad \begin{cases} \begin{bmatrix} 01 & 02 & 03 & 01 \end{bmatrix} \begin{bmatrix} \rho \\ n_9 \oplus \tilde{n}_9 \\ \omega \\ n_{11} \oplus \tilde{n}_{11} \end{bmatrix} = 0 \\ \begin{bmatrix} 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} \rho \\ n_9 \oplus \tilde{n}_9 \\ \omega \\ n_{11} \oplus \tilde{n}_{11} \end{bmatrix} = 0 \end{cases} \quad (4)$$

Once we know the values of π, ϕ, ρ and ω, message block $\tilde{M}_y$ can be chosen as follows:

1. Set the values of $\tilde{n}_0^{new}$, $\tilde{n}_8^{new}$, $\tilde{n}_2^{new}$ and $\tilde{n}_{10}^{new}$ as follows: $\tilde{n}_0^{new} = n_0 \oplus \pi$, $\tilde{n}_8^{new} = n_8 \oplus \rho$, $\tilde{n}_2^{new} = n_2 \oplus \phi$, and $\tilde{n}_{10}^{new} = n_{10} \oplus \omega$. Use $\tilde{n}_0^{new}$ to replace $\tilde{n}_0$, $\tilde{n}_8^{new}$ to replace $\tilde{n}_8$, $\tilde{n}_2^{new}$ to replace $\tilde{n}_2$, and $\tilde{n}_{10}^{new}$ to replace $\tilde{n}_{10}$.
2. Perform SR^{-1} and SB^{-1}. Since SR^{-1} and SB^{-1} are permutation and substitution, they do not affect the properties we have found. Now we have the outputs of ARK in Round y.
3. Compute the value of $\tilde{M}_y$ as follows:

$$\tilde{M}_y = (\tilde{n}_0^{new} \oplus \tilde{s}_0)||(\tilde{n}_8^{new} \oplus \tilde{s}_8)||(\tilde{n}_{10}^{new} \oplus \tilde{s}_2)||(\tilde{n}_2^{new} \oplus \tilde{s}_{10}).$$

So far, two messages (M_{y-3}, M_{y-2}, M_{y-1}, M_y) and ($\tilde{M}_{y-3}$, $\tilde{M}_{y-2}$, $\tilde{M}_{y-1}^{new}$, $\tilde{M}_y$) collide on 96 bits (i.e., Bytes w_1, w_3, w_4, w_5, w_6, w_7, w_9, w_{11}, w_{12}, w_{13}, w_{14} and w_{15}) in Round $y+1$ after MC transformation.

3.1.3 Extending 96-bit Collisions to 128-bit Collisions This step is straightforward as we can select message M_{y+1} arbitrarily, and construct message $\tilde{M}_{y+1}$ to cancel the differences between Bytes w_0, w_8, w_2 and w_{10}. The construction is provided as follows:

$$\tilde{M}_{y+1} = ((w_0 \oplus \tilde{w}_0)||(w_8 \oplus \tilde{w}_8)||(w_2 \oplus \tilde{w}_2)||(w_{10} \oplus \tilde{w}_{10})) \oplus M_{y+1}.$$

3.2 The Backwards-aNd-Backwards Search

The Backwards-aNd-Backwards search has the following form:

Known:	1. a selected key or a selected intermediate value. 2. one selected four-block message $M(M_{y-3},\ M_{y-2},\ M_{y-1},\ M_y)$.
Find:	another four-block message $\tilde{M}(\tilde{M}_{y-3},\ \tilde{M}_{y-2},\ \tilde{M}_{y-1},\ \tilde{M}_y)$ such that these two messages collide on 32 bits (Bytes s_4, s_{12}, s_6 and s_{14}) after MC in Round y.
Method:	solve four groups of linear functions. These four groups of functions are named as (5), (6), (7) and (8) in this subsection.

We propose a method to find 32-bit collisions on Bytes s_4, s_{12}, s_6 and s_{14} (see Figure 2) by solving four groups of linear functions. This search assumes that for a selected key (or a selected intermediate value) and a selected four-block message $(M_{y-3},\ M_{y-2},\ M_{y-1},\ M_y)$, we can generate another four-block message $(\tilde{M}_{y-3},\ \tilde{M}_{y-2},\ \tilde{M}_{y-1},\ \tilde{M}_y)$ such that these two messages collide on Bytes s_4, s_{12}, s_6 and s_{14} after MC in Round y. The method used by the BNB search is similar to the idea employed by the BNF search, but works in only one direction (i.e., only backwards).

3.2.1 Deciding Four Values ($\tilde{j}_5$, $\tilde{j}_7$, $\tilde{j}_{13}$ and $\tilde{j}_{15}$) In the beginning, we choose $(\tilde{M}_{y-3},\ \tilde{M}_{y-2},\ \tilde{M}_{y-1},\ \tilde{M}_y)$ randomly. Assume that the input and the output of MC in Round y are listed as follows:

$$\begin{bmatrix} \tilde{j}_0 & \tilde{j}_4 & \tilde{j}_8 & \tilde{j}_{12} \\ \tilde{j}_1 & \tilde{j}_5^{old} & \tilde{j}_9 & \tilde{j}_{13}^{old} \\ \tilde{j}_2 & \tilde{j}_6 & \tilde{j}_{10} & \tilde{j}_{14} \\ \tilde{j}_3 & \tilde{j}_7^{old} & \tilde{j}_{11} & \tilde{j}_{15}^{old} \end{bmatrix} \xrightarrow{MC} \begin{bmatrix} \tilde{s}_0 & \tilde{s}_4 & \tilde{s}_8 & \tilde{s}_{12} \\ \tilde{s}_1 & \tilde{s}_5 & \tilde{s}_9 & \tilde{s}_{13} \\ \tilde{s}_2 & \tilde{s}_6 & \tilde{s}_{10} & \tilde{s}_{14} \\ \tilde{s}_3 & \tilde{s}_7 & \tilde{s}_{11} & \tilde{s}_{15} \end{bmatrix}.$$

Now we do not use the values of $\tilde{j}_5^{old}$, $\tilde{j}_7^{old}$, $\tilde{j}_{13}^{old}$ or $\tilde{j}_{15}^{old}$. Instead, we use $\tilde{j}_5$ (to replace $\tilde{j}_5^{old}$), $\tilde{j}_7$ (to replace $\tilde{j}_7^{old}$), $\tilde{j}_{13}$ (to replace $\tilde{j}_{13}^{old}$), and $\tilde{j}_{15}$ (to replace $\tilde{j}_{15}^{old}$) such that we get values s_4, s_{12}, s_6, and s_{14} on Bytes $\tilde{s}_4$, $\tilde{s}_{12}$, $\tilde{s}_6$, and $\tilde{s}_{14}$, respectively (illustrated as follows):

$$\begin{bmatrix} \tilde{j}_0 & \tilde{j}_4 & \tilde{j}_8 & \tilde{j}_{12} \\ \tilde{j}_1 & \tilde{j}_5 & \tilde{j}_9 & \tilde{j}_{13} \\ \tilde{j}_2 & \tilde{j}_6 & \tilde{j}_{10} & \tilde{j}_{14} \\ \tilde{j}_3 & \tilde{j}_7 & \tilde{j}_{11} & \tilde{j}_{15} \end{bmatrix} \xrightarrow{MC} \begin{bmatrix} \tilde{s}_0 & s_4 & \tilde{s}_8 & s_{12} \\ \tilde{s}_1 & \tilde{s}_5 & \tilde{s}_9 & \tilde{s}_{13} \\ \tilde{s}_2 & s_6 & \tilde{s}_{10} & s_{14} \\ \tilde{s}_3 & \tilde{s}_7 & \tilde{s}_{11} & \tilde{s}_{15} \end{bmatrix}.$$

Now the question is "how can we make this happen". Our answer is to solve two groups of linear functions. For the values of s_4 and s_6, we have two linear equations in (5) with only two unknown variables ($\tilde{j}_5$ and $\tilde{j}_7$). Therefore, we can solve (5) to obtain the values of $\tilde{j}_5$ and $\tilde{j}_7$.

$$\begin{cases} \begin{bmatrix} 02 & 03 & 01 & 01 \end{bmatrix} \begin{bmatrix} \tilde{j}_4 \\ \tilde{j}_5 \\ \tilde{j}_6 \\ \tilde{j}_7 \end{bmatrix} = s_4 \\ \begin{bmatrix} 01 & 01 & 02 & 03 \end{bmatrix} \begin{bmatrix} \tilde{j}_4 \\ \tilde{j}_5 \\ \tilde{j}_6 \\ \tilde{j}_7 \end{bmatrix} = s_6 \end{cases} \quad (5) \qquad \begin{cases} \begin{bmatrix} 02 & 03 & 01 & 01 \end{bmatrix} \begin{bmatrix} \tilde{j}_{12} \\ \tilde{j}_{13} \\ \tilde{j}_{14} \\ \tilde{j}_{15} \end{bmatrix} = s_{12} \\ \begin{bmatrix} 01 & 01 & 02 & 03 \end{bmatrix} \begin{bmatrix} \tilde{j}_{12} \\ \tilde{j}_{13} \\ \tilde{j}_{14} \\ \tilde{j}_{15} \end{bmatrix} = s_{14} \end{cases} \quad (6)$$

Similarly, for the values of s_{12} and s_{14}, we have two linear functions in (6) with two unknown variables ($\tilde{j}_{13}$ and $\tilde{j}_{15}$). We can solve (6) to decide the values of $\tilde{j}_{13}$ and $\tilde{j}_{15}$. After getting four values ($\tilde{j}_5$, $\tilde{j}_7$, $\tilde{j}_{13}$, and $\tilde{j}_{15}$) decided, we perform the SR^{-1} and SB^{-1} transformations. As SR^{-1} is permutation and SB^{-1} is substitution, $\tilde{j}_5$, $\tilde{j}_7$, $\tilde{j}_{13}$, and $\tilde{j}_{15}$ are first relocated then substituted by another four values $\tilde{i}_9$, $\tilde{i}_3$, $\tilde{i}_1$, and $\tilde{i}_{11}$, respectively. As the message injection layout does not change the values of $\tilde{i}_9$, $\tilde{i}_3$, $\tilde{i}_1$, and $\tilde{i}_{11}$, these four values are not changed after we do ARK. So, we get four known values ($\tilde{i}_9$, $\tilde{i}_3$, $\tilde{i}_1$, and $\tilde{i}_{11}$) after MC in Round $y-1$. Our next target is to modify message block $\tilde{M}_{y-2}$ so that we get those four values $\tilde{i}_9$, $\tilde{i}_3$, $\tilde{i}_1$, and $\tilde{i}_{11}$ after MC in Round $y-1$.

3.2.2 Modifying Message Block $\tilde{M}_{y-2}$ Suppose by using the original message block $\tilde{M}_{y-2}$, we have the following states in Round $y-1$:

$$\begin{bmatrix} \tilde{g}_0^{*old} & \tilde{g}_4 & \tilde{g}_8^{*old} & \tilde{g}_{12} \\ \tilde{g}_1 & \tilde{g}_5 & \tilde{g}_9 & \tilde{g}_{13} \\ \tilde{g}_2^{*old} & \tilde{g}_6 & \tilde{g}_{10}^{*old} & \tilde{g}_{14} \\ \tilde{g}_3 & \tilde{g}_7 & \tilde{g}_{11} & \tilde{g}_{15} \end{bmatrix} \xrightarrow{SB \circ SR} \begin{bmatrix} \tilde{h}_0^{old} & \tilde{h}_4 & \tilde{h}_8^{old} & \tilde{h}_{12} \\ \tilde{h}_1 & \tilde{h}_5 & \tilde{h}_9 & \tilde{h}_{13} \\ \tilde{h}_2^{old} & \tilde{h}_6 & \tilde{h}_{10}^{old} & \tilde{h}_{14} \\ \tilde{h}_3 & \tilde{h}_7 & \tilde{h}_{11} & \tilde{h}_{15} \end{bmatrix} \xrightarrow{MC} \begin{bmatrix} ? & \tilde{i}_4 & ? & \tilde{i}_{12} \\ ? & \tilde{i}_5 & ? & \tilde{i}_{13} \\ ? & \tilde{i}_6 & ? & \tilde{i}_{14} \\ ? & \tilde{i}_7 & ? & \tilde{i}_{15} \end{bmatrix}.$$

Now we replace values ($\tilde{h}_0^{old}$, $\tilde{h}_2^{old}$, $\tilde{h}_8^{old}$, $\tilde{h}_{10}^{old}$) with ($\tilde{h}_0$, $\tilde{h}_2$, $\tilde{h}_8$, $\tilde{h}_{10}$) and then we get those four values ($\tilde{i}_9$, $\tilde{i}_3$, $\tilde{i}_1$, and $\tilde{i}_{11}$) located as follows:

$$\begin{bmatrix} \tilde{g}_0^{*} & \tilde{g}_4 & \tilde{g}_8^{*} & \tilde{g}_{12} \\ \tilde{g}_1 & \tilde{g}_5 & \tilde{g}_9 & \tilde{g}_{13} \\ \tilde{g}_2^{*} & \tilde{g}_6 & \tilde{g}_{10}^{*} & \tilde{g}_{14} \\ \tilde{g}_3 & \tilde{g}_7 & \tilde{g}_{11} & \tilde{g}_{15} \end{bmatrix} \xrightarrow{SB \circ SR} \begin{bmatrix} \tilde{h}_0 & \tilde{h}_4 & \tilde{h}_8 & \tilde{h}_{12} \\ \tilde{h}_1 & \tilde{h}_5 & \tilde{h}_9 & \tilde{h}_{13} \\ \tilde{h}_2 & \tilde{h}_6 & \tilde{h}_{10} & \tilde{h}_{14} \\ \tilde{h}_3 & \tilde{h}_7 & \tilde{h}_{11} & \tilde{h}_{15} \end{bmatrix} \xrightarrow{MC} \begin{bmatrix} ? & \tilde{i}_4 & ? & \tilde{i}_{12} \\ \tilde{i}_1 & \tilde{i}_5 & \tilde{i}_9 & \tilde{i}_{13} \\ ? & \tilde{i}_6 & ? & \tilde{i}_{14} \\ \tilde{i}_3 & \tilde{i}_7 & \tilde{i}_{11} & \tilde{i}_{15} \end{bmatrix}.$$

Based on the property of MC transformation, we can form the following two groups of linear functions:

$$\begin{cases} \begin{bmatrix} 01 & 02 & 03 & 01 \end{bmatrix} \begin{bmatrix} \tilde{h}_0 \\ \tilde{h}_1 \\ \tilde{h}_2 \\ \tilde{h}_3 \end{bmatrix} = \tilde{i}_1 \\ \begin{bmatrix} 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} \tilde{h}_0 \\ \tilde{h}_1 \\ \tilde{h}_2 \\ \tilde{h}_3 \end{bmatrix} = \tilde{i}_3 \end{cases} \quad (7) \qquad \begin{cases} \begin{bmatrix} 01 & 02 & 03 & 01 \end{bmatrix} \begin{bmatrix} \tilde{h}_8 \\ \tilde{h}_9 \\ \tilde{h}_{10} \\ \tilde{h}_{11} \end{bmatrix} = \tilde{i}_9 \\ \begin{bmatrix} 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} \tilde{h}_8 \\ \tilde{h}_9 \\ \tilde{h}_{10} \\ \tilde{h}_{11} \end{bmatrix} = \tilde{i}_{11} \end{cases} \quad (8)$$

We know the values of $\tilde{h}_1$, $\tilde{h}_3$, $\tilde{h}_9$ and $\tilde{h}_{11}$ from the original message block $\tilde{M}_{y-2}$. We can get the values of ($\tilde{h}_0$, $\tilde{h}_2$) by solving (7), and get the values of ($\tilde{h}_8$, $\tilde{h}_{10}$) by solving (8). After finding the values of ($\tilde{h}_0$, $\tilde{h}_2$, $\tilde{h}_8$, $\tilde{h}_{10}$), we perform SR^{-1} and SB^{-1}, and obtain the corresponding four values ($\tilde{g}_0^*$, $\tilde{g}_2^*$, $\tilde{g}_8^*$, $\tilde{g}_{10}^*$). Once we know the values of ($\tilde{g}_0^*$, $\tilde{g}_2^*$, $\tilde{g}_8^*$, $\tilde{g}_{10}^*$), we replace $\tilde{M}_{y-2}$ with $\tilde{M}_{y-2}^{new}$. $\tilde{M}_{y-2}^{new}$ is constructed as follows (note that $\tilde{g}_0, \tilde{g}_8, \tilde{g}_2$ and $\tilde{g}_{10}$ are known from the message block $\tilde{M}_{y-3}$ in Round $y-3$):

$$\tilde{M}_{y-2}^{new} = (\tilde{g}_0^* \oplus \tilde{g}_0)||(\tilde{g}_8^* \oplus \tilde{g}_8)||(\tilde{g}_2^* \oplus \tilde{g}_2)||(\tilde{g}_{10}^* \oplus \tilde{g}_{10}).$$

3.3 Combining the BNB Search with the BNF Search

The second-preimage search algorithm combines the BNB search with the BNF search. To search for a second preimage of the Alpha-MAC, we perform the following steps:

1. Select a key or an intermediate value.
2. Select a five-block message $M(M_{y-3}, M_{y-2}, M_{y-1}, M_y, M_{y+1})$.
3. Generate the second preimage $\tilde{M}(\tilde{M}_{y-3}, \tilde{M}_{y-2}, \tilde{M}_{y-1}, \tilde{M}_y, \tilde{M}_{y+1})$ randomly. We need to guarantee that $\tilde{M}_{y-3}$ is not equal to M_{y-3}.
4. Perform the BNB search to generate 32-bit collisions. The BNB search is done by modifying message block $\tilde{M}_{y-2}$.
5. Use the BNF search to extend those 32-bit collisions to 128-bit collisions. The BNF search is carried out by modifying the values of $\tilde{M}_{y-1}, \tilde{M}_y$, and $\tilde{M}_{y+1}$. Message $\tilde{M}(\tilde{M}_{y-3}, \tilde{M}_{y-2}, \tilde{M}_{y-1}, \tilde{M}_y, \tilde{M}_{y+1})$ is a second preimage of message $M(M_{y-3}, M_{y-2}, M_{y-1}, M_y, M_{y+1})$ under the selected key (or the selected intermediate value).

The routine of finding second preimages is shown in Table 1, and Figure 3 depicts this finding. The name of the BNB search comes from the fact that searching for $\tilde{M}_{y-2}$ is carried out by moving backwards and then backwards, and the name of the BNF search comes from the fact that searching for $\tilde{M}_{y-1}$, $\tilde{M}_y$ and $\tilde{M}_{y+1}$ is performed by moving backwards and then forwards (see Table

1). A personal computer takes about 1 second to find a second preimage of the Alpha-MAC. In Appendix A, we provide a second preimage of a selected key and a selected five-block message.

Table 1. Second-preimage search = BNB search + BNF search

Search	R	Round $y-2$	Di	Round $y-1$	Di	Round y
BNB	1				$\Leftarrow$	$\tilde{s}_4 \rightharpoonup s_4$, $\tilde{s}_{12} \rightharpoonup s_{12}$, $\tilde{s}_6 \rightharpoonup s_6$, $\tilde{s}_{14} \rightharpoonup s_{14}$
	2		$\Leftarrow$	$\tilde{h}_0^{old} \rightharpoonup \tilde{h}_0$, $\tilde{h}_2^{old} \rightharpoonup \tilde{h}_2$, $\tilde{h}_8^{old} \rightharpoonup \tilde{h}_8$, $\tilde{h}_{10}^{old} \rightharpoonup \tilde{h}_{10}$		
	3	$\tilde{M}_{y-2} \rightharpoonup \tilde{M}_{y-2}^{new}$				
		Round $y-1$	Di	Round y	Di	Round $y+1$
BNF	4	modify $\tilde{M}_{y-1}$	$\Leftarrow$	collisions on s_4, s_{12}, s_6 and s_{14}		
	5		$\Rightarrow$	collisions on s_4, s_{12}, s_6, s_{14}, s_1, s_9, s_3 and s_{11}		
	6			modify $\tilde{M}_y$	$\Rightarrow$	96-bit collisions
	7					modify $\tilde{M}_{y+1} \rightarrow$ 128-bit collisions

Di - Direction

R - Routine

4 The Collision Search Algorithm

Known:	a selected key or a selected intermediate value.
Find:	two five-block messages M and $\tilde{M}$ such that they collide under the selected key or the intermediate value.
Method:	employ the second-preimage search.

In the second-preimage search, we choose the first five-block message arbitrarily, and once it is decided, we do not modify it. All we need to do is modify the second five-block message so that 128-bit collisions happen. Therefore, the second-preimage search can also be used to find two colliding five-block messages under a selected key (or a selected intermediate value).

5 Conclusions

In this paper, we have presented our analysis on the internal structure of Alpha-MAC. We proposed a method to find second preimages of the Alpha-MAC by combining the Backwards-aNd-Forwards search and the Backwards-aNd-Backwards search, based on the assumption that a key or an intermediate value is

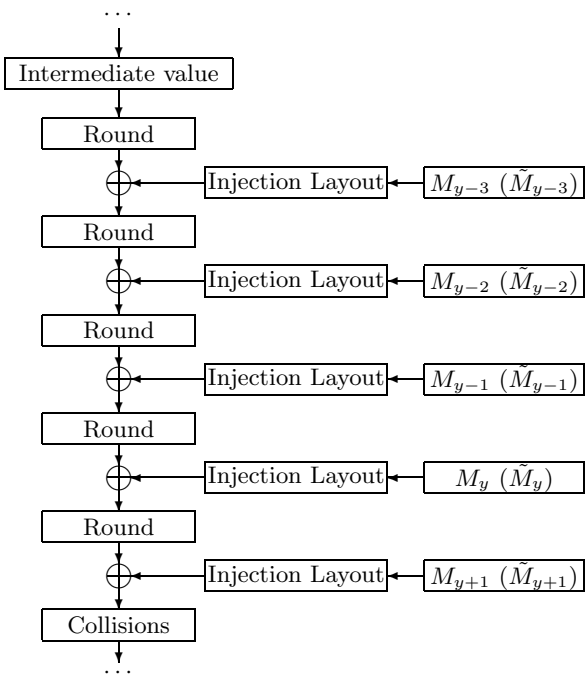

Fig. 3. The second-preimage search

known. Our method employs the algebraic properties of AES and the structural features of the ALPHA-MAC. To find a second preimage of the ALPHA-MAC, our idea needs to solve eight groups of linear functions. We also showed that the second-preimage finding method can be used to generate internal collisions.

References

1. National Institute of Standards and Technology, U.S. Department of Commerce. *Advanced Encryption Standard (AES)*. Federal Information Processing Standard 197, 2001.
2. National Institute of Standards and Technology, U.S. Department of Commerce. *Secure hash standard*. Federal Information Processing Standard, FIPS-180-1, 1995.
3. National Institute of Standards and Technology, U.S. Department of Commerce. *Secure hash standard*. Federal Information Processing Standard, FIPS-180, 1993.
4. National Institute of Standards and Technology, U.S. Department of Commerce. *The Keyed-Hash Message Authentication Code (HMAC)*. Federal Information Processing Standard 198, 2002.
5. Mihir Bellare, Ran Canetti, and Hugo Krawczyk. *Keying Hash Functions for Message Authentication*. Advances in Cryptology - CRYPTO'96, 16th Annual International Cryptology Conference, Lecture Notes in Computer Science 1109, pp. 1-15, Springer-Verlag, 1996.
6. Eli Biham and Rafi Chen. *Near-Collisions of SHA-0*. Advances in Cryptology - CRYPTO 2004, 24th Annual International Cryptology Conference, Lecture Notes in Computer Science 3152, pp. 290-305, Springer-Verlag, 2004.

7. Eli Biham, Rafi Chen, Antoine Joux, Patrick Carribault, Christophe Lemuet, and William Jalby. *Collisions of SHA-0 and Reduced SHA-1.* Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lecture Notes in Computer Science 3494, pp. 36-57, Springer-Verlag, 2005.
8. Eli Biham and Adi Shamir. *Differential Cryptanalysis of the Data Encryption Standard*. Springer-Verlag, 1993.
9. John Black, Shai Halevi, Hugo Krawczyk, Ted Krovetz, and Phillip Rogaway. *UMAC: Fast and Secure Message Authentication*. Advances in Cryptology - CRYPTO'99, 19th Annual International Cryptology Conference, Lecture Notes in Computer Science 1666, pp. 216-233, Springer-Verlag, 1999.
10. Florent Chabaud and Antoine Joux. *Differential Collisions in SHA-0*. Advances in Cryptology - CRYPTO'98, 18th Annual International Cryptology Conference, Lecture Notes in Computer Science 1462, pp. 56-71, Springer-Verlag, 1998.
11. Joan Daemen and Vincent Rijmen. *A New MAC Construction ALRED and a Specific Instance ALPHA-MAC*. Fast Software Encryption: 12th International Workshop, FSE 2005, Lecture Notes in Computer Science 3557, pp. 1-17, Springer-Verlag, 2005.
12. Joan Daemen and Vincent Rijmen. *AES Proposal: Rijndael*, AES Round 1 Technical Evaluation CD-1: Documentation, National Institute of Standards and Technology, 1998.
13. Donald Davies. *A Message Authenticator Algorithm Suitable for A Mainframe Computer*. Advances in Cryptology, Proceedings of CRYPTO'84, Lecture Notes in Computer Science 196, pp. 393-400, Springer-Verlag, 1985.
14. Hans Dobbertin. *Cryptanalysis of MD4*. Fast Software Encryption: Third International Workshop, FSE 1996, Lecture Notes in Computer Science 1039, pp. 53-69, Springer-Verlag, 1996.
15. International Organization for Standardization. *ISO/IEC 9797-1, Information technology – Security techniques – Message Authentication Codes (MACs) – Part 1: Mechanisms using a block cipher* . 1999.
16. Bart Preneel and Paul C. van Oorschot. *MDx-MAC and Building Fast MACs from Hash Functions*. Advances in Cryptology - CRYPTO'95, 15th Annual International Cryptology Conference, Lecture Notes in Computer Science 963, pp. 1-14, Springer-Verlag, 1995.
17. Ronald Rivest. The MD4 message-digest algorithm, Request for Comments (RFC) 1320, Internet Activities Board, Internet Privacy Task Force, 1992.
18. Ronald Rivest. The MD5 message-digest algorithm, Request for Comments (RFC 1320), Internet Activities Board, Internet Privacy Task Force, 1992.
19. Xiaoyun Wang, Xuejia Lai, Dengguo Feng, Hui Chen, and Xiuyuan Yu. *Cryptanalysis of the Hash Functions MD4 and RIPEMD*. Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lecture Notes in Computer Science 3494, pp. 1-18, Springer-Verlag, 2005.
20. Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. *Finding Collisions in the Full SHA-1*. Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Lecture Notes in Computer Science 3621, pp. 17-36, Springer-Verlag, 2005.
21. Xiaoyun Wang and Hongbo Yu. *How to Break MD5 and Other Hash Functions*. Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lecture Notes in Computer Science 3494, pp. 19-35, Springer-Verlag, 2005.

22. Xiaoyun Wang, Hongbo Yu, and Yiqun Lisa Yin. *Efficient Collision Search Attacks on SHA-0.* Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Lecture Notes in Computer Science 3621, pp. 1-16, Springer-Verlag, 2005.

A A Found Second Preimage

For a selected key K (see Table 3) and a selected five-block message M (see Table 2), a second preimage found by our algorithm is $\tilde{M}$ (shown in Table 2). The 128-bit colliding value is listed in Table 4. Note that these two messages are listed after injection layout.

Table 2. Two five-block messages

M (the selected message)																				
M_{y-3}				M_{y-2}				M_{y-1}				M_y				M_{y+1}			
c4	0	8c	0	e6	0	2a	0	77	0	fd	0	ef	0	a1	0	81	0	9f	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
94	0	f3	0	95	0	04	0	4c	0	37	0	68	0	09	0	25	0	2c	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\tilde{M}$ (the found second preimage)																				
$\tilde{M}_{y-3}$				$\tilde{M}_{y-2}$				$\tilde{M}_{y-1}$				$\tilde{M}_y$				$\tilde{M}_{y+1}$			
1d	0	43	0	22	0	04	0	e4	0	83	0	2f	0	e5	0	69	0	06	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1c	0	0d	0	2f	0	30	0	2f	0	9b	0	d4	0	30	0	f4	0	3a	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 3. The selected key K

83	55	2d	81
88	2c	05	67
c1	63	be	c2
2a	a2	52	a4

Table 4. The 128-bit collisions

7d	69	88	d7
02	cb	1f	af
b9	d8	7b	5e
0e	10	79	21