Preimage Attacks on Reduced Tiger and SHA-2

Takanori Isobe and Kyoji Shibutani

Sony Corporation
1-7-1 Konan, Minato-ku, Tokyo 108-0075, Japan
{Takanori.Isobe,Kyoji.Shibutani}@jp.sony.com

Abstract. This paper shows new preimage attacks on reduced Tiger and SHA-2. Indesteege and Preneel presented a preimage attack on Tiger reduced to 13 rounds (out of 24) with a complexity of $2^{128.5}$. Our new preimage attack finds a one-block preimage of Tiger reduced to 16 rounds with a complexity of 2^{161}. The proposed attack is based on meet-in-the-middle attacks. It seems difficult to find "independent words" of Tiger at first glance, since its key schedule function is much more complicated than that of MD4 or MD5. However, we developed techniques to find independent words efficiently by controlling its internal variables. Surprisingly, the similar techniques can be applied to SHA-2 including both SHA-256 and SHA-512. We present a one-block preimage attack on SHA-256 and SHA-512 reduced to 24 (out of 64 and 80) steps with a complexity of 2^{240} and 2^{480}, respectively. To the best of our knowledge, our attack is the best known preimage attack on reduced-round Tiger and our preimage attack on reduced-step SHA-512 is the first result. Furthermore, our preimage attacks can also be extended to second preimage attacks directly, because our attacks can obtain random preimages from an arbitrary IV and an arbitrary target.

Keywords: hash function, preimage attack, second preimage attack, meet-in-the-middle, Tiger, SHA-256, SHA-512.

1 Introduction

Cryptographic hash functions play an important role in the modern cryptology. Many cryptographic protocols require a secure hash function which holds several security properties such as classical ones: collision resistance, preimage resistance and second preimage resistance. However, a lot of hash functions have been broken by collision attacks including the attacks on MD4 [3], MD5 [11] and SHA-1 [12]. These hash functions are considered to be broken in theory, but in practice many applications still use these hash functions because they do not require collision resistance. However, (second) preimage attacks are critical for many applications including integrity checks and encrypted password systems. Thus analyzing the security of the hash function with respect to (second) preimage resistance is important, even if the hash function is already broken by a collision attack. However, the preimage resistance of hash functions has not been studied well.

O. Dunkelman (Ed.): FSE 2009, LNCS 5665, pp. 139–155, 2009.

Table 1. Summary of our results

Target	Attack (first or second preimage)	Attacked steps (rounds)	Complexity
Tiger (full 24 rounds)	first [4]	13	$2^{128.5}$
	first **(this paper)**	16	2^{161}
	second [4]	13	$2^{127.5}$
	second **(this paper)**	16	2^{160}
SHA-256 (full 64 steps)	first [10]	36	2^{249}
	first **(this paper)**	24	2^{240}
	second **(this paper)**	24	2^{240}
SHA-512 (full 80 steps)	first **(this paper)**	24	2^{480}
	second **(this paper)**	24	2^{480}

Tiger is a dedicated hash function producing a 192-bit hash value designed by Anderson and Biham in 1996 [2]. As a cryptanalysis of Tiger, at FSE 2006, Kelsey and Lucks proposed a collision attack on 17-round Tiger with a complexity of 2^{49} [5], where full-version Tiger has 24 rounds. They also proposed a pseudo-near collision attack on 20-round Tiger with a complexity of 2^{48}. This attack was improved by Mendel et al. at INDOCRYPT 2006 [8]. They proposed a collision attack on 19-round Tiger with a complexity of 2^{62}, and a pseudo-near collision attack on 22-round Tiger with a complexity of 2^{44}. Later, they proposed a pseudo-near-collision attack of full-round (24-round) Tiger with a complexity of 2^{44}, and a pseudo-collision (free-start-collision) attack on 23-round Tiger [9]. The above results are collision attacks and there is few evaluations of preimage resistance of Tiger. Indesteege and Preneel presented preimage attacks on reduced-round Tiger [4]. Their attack found a preimage of Tiger reduced to 13 rounds with a complexity of $2^{128.5}$.

In this paper, we introduce a preimage attack on reduced-round Tiger. The proposed attack is based on meet-in-the-middle attacks [1]. In this attack, we need to find independent words ("neutral words") in the first place. However, the techniques used for finding independent words of MD4 or MD5 cannot be applied to Tiger directly, since its key schedule function is much more complicated than that of MD4 or MD5. To overcome this problem, we developed new techniques to find independent words of Tiger efficiently by adjusting the internal variables. As a result, the proposed attack finds a preimage of Tiger reduced to 16 (out of 24) rounds with a complexity of about 2^{161}. Surprisingly, our new approach can be applied to SHA-2 including both SHA-256 and SHA-512. We present a preimage attack on SHA-256 and SHA-512 reduced to 24 (out of 64 and 80) steps with a complexity of about 2^{240} and 2^{480}, respectively. As far as we know, our attack is the best known preimage attack on reduced-round Tiger and our preimage attack on reduced-step SHA-512 is the first result. Furthermore, we show that our preimage attacks can also be extended to second preimage attacks directly and all of our attacks can obtain one-block preimages, because our preimage attacks can obtain random preimages from an arbitrary IV and an arbitrary target. These results are summarized in Table 1.

This paper is organized as follows. Brief descriptions of Tiger, SHA-2 and the meet-in-the-middle approach are given in Section 2. A preimage attack on reduced-round Tiger and its extensions are shown in Section 3. In Section 4, we present a preimage attack on reduced-step SHA-2. Finally, we present conclusions in Section 5.

2 Preliminaries

2.1 Description of Tiger

Tiger is an iterated hash function that compresses an arbitrary length message into a 192-bit hash value. An input message value is divided into 512-bit message blocks $(M^{(0)}, M^{(1)}, ..., M^{(t-1)})$ by the padding process as well as the MD family. The compression function of Tiger shown in Fig. 1 generates a 192-bit output chaining value $H^{(i+1)}$ from a 512-bit message block $M^{(i)}$ and a 192-bit input chaining value $H^{(i)}$ where chaining values consist of three 64-bit variables, $A_j^{(i)}$, $B_j^{(i)}$ and $C_j^{(i)}$. The initial chaining value $H^{(0)} = (A_0^{(0)}, B_0^{(0)}, C_0^{(0)})$ is as follows:

$$A_0^{(0)} = \texttt{0x0123456789ABCDEF},$$
$$B_0^{(0)} = \texttt{0xFEDCBA9876543210},$$
$$C_0^{(0)} = \texttt{0xF096A5B4C3B2E187}.$$

In the compression function, a 512-bit message block $M^{(i)}$ is divided into eight 64-bit words $(X_0, X_1, ..., X_7)$. The compression function consists of three pass functions and between each of them there is a key schedule function. Since each pass function has eight round functions, the compression function consists of 24 round functions. The pass function is used for updating chaining values, and the key schedule function is used for updating message values. After the third pass function, the following feedforward process is executed to give outputs of the compression function with input chaining values and outputs of the third pass function,

$$A'_{24} = A_0 \oplus A_{24},\ B'_{24} = B_0 - B_{24},\ C'_{24} = C_0 + C_{24},$$

where A_i, B_i and C_i denote the i-th round chaining values, respectively, and A'_{24}, B'_{24} and C'_{24} are outputs of the compression function.

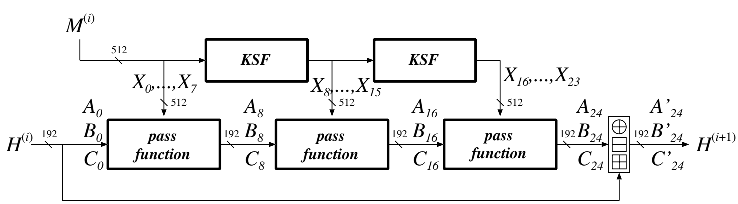

Fig. 1. Compression function f of Tiger

In each round of the pass function, chaining values A_i, B_i and C_i are updated by a message word X_i as follows:

$$B_{i+1} = C_i \oplus X_i, \tag{1}$$
$$C_{i+1} = A_i - even(B_{i+1}), \tag{2}$$
$$A_{i+1} = (B_i + odd(B_{i+1})) \times mul, \tag{3}$$

where mul is the constant value $\in \{5, 7, 9\}$ which is different in each pass function. The nonlinear functions *even* and *odd* are expressed as follows:

$$even(W) = T_1[w_0] \oplus T_2[w_2] \oplus T_3[w_4] \oplus T_4[w_6], \tag{4}$$
$$odd(W) = T_4[w_1] \oplus T_3[w_3] \oplus T_2[w_5] \oplus T_1[w_7], \tag{5}$$

where 64-bit value W is split into eight bytes $\{w_7, w_6, ..., w_0\}$ with w_7 is the most significant byte and $T_1, ..., T_4$ are the S-boxes: $\{0,1\}^8 \to \{0,1\}^{64}$. Figure 2 shows the round function of Tiger.

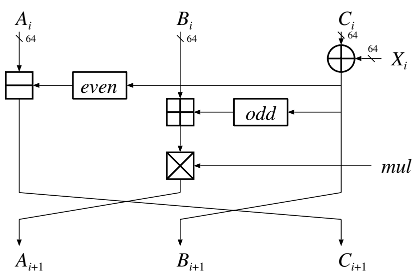

Fig. 2. Tiger round function

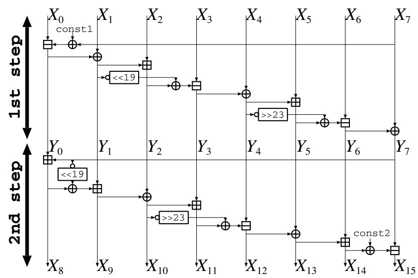

Fig. 3. Key schedule function

The key schedule function (KSF) updates message values. In the first pass function, eight message words $X_0, ..., X_7$, which are identical to input message blocks of the compression function, are used for updating chaining values. Remaining two pass functions use sixteen message words which are generated by applying KSF:

$$(X_8, ..., X_{15}) = KSF(X_0, ..., X_7), \tag{6}$$
$$(X_{16}, ..., X_{23}) = KSF(X_8, ..., X_{15}). \tag{7}$$

The function KSF which updates the inputs $X_0, ..., X_7$ in two steps, is shown in Table 2. The first step shown in the left table generates internal variables $Y_0, ..., Y_7$ from inputs $X_0, ..., X_7$, and the second step shown in the right table calculates outputs $X_8, ..., X_{15}$ from internal variables $Y_0, .., Y_7$, where `const1` is `0xA5A5A5A5A5A5A5A5` and `const2` is `0x0123456789ABCDEF`. By using the same function, $X_{16}, ..., X_{23}$ are also derived from $X_8, ..., X_{15}$. Figure 3 shows the key schedule function of Tiger.

Table 2. Algorithm of the key schedule function KSF

$Y_0 = X_0 - (X_7 \oplus \texttt{const1}),$ (8)	$X_8 = Y_0 + Y_7,$ (16)
$Y_1 = X_1 \oplus Y_0,$ (9)	$X_9 = Y_1 - (X_8 \oplus (\overline{Y_7} \ll 19)),$ (17)
$Y_2 = X_2 + Y_1,$ (10)	$X_{10} = Y_2 \oplus X_9,$ (18)
$Y_3 = X_3 - (Y_2 \oplus (\overline{Y_1} \ll 19)),$ (11)	$X_{11} = Y_3 + X_{10},$ (19)
$Y_4 = X_4 \oplus Y_3,$ (12)	$X_{12} = Y_4 - (X_{11} \oplus (\overline{X_{10}} \gg 23)),$ (20)
$Y_5 = X_5 + Y_4,$ (13)	$X_{13} = Y_5 \oplus X_{12},$ (21)
$Y_6 = X_6 - (Y_5 \oplus (\overline{Y_4} \gg 23)),$ (14)	$X_{14} = Y_6 + X_{13},$ (22)
$Y_7 = X_7 \oplus Y_6.$ (15)	$X_{15} = Y_7 - (X_{14} \oplus \texttt{const2}).$ (23)

2.2 Description of SHA-256

We only show the structure of SHA-256, since SHA-512 is structurally very similar to SHA-256 except for the number of steps, word size and rotation values. The compression function of SHA-256 consists of a message expansion function and a state update function. The message expansion function expands 512-bit message block into 64 32-bit message words $W_0, ..., W_{63}$ as follows:

$$W_i = \begin{cases} M_i & (0 \leq i < 16), \\ \sigma_1(W_{i-2}) + W_{i-7} + \sigma_0(W_{i-15}) + W_{i-16} & (16 \leq i < 64), \end{cases}$$

where the functions $\sigma_0(X)$ and $\sigma_1(X)$ are given by

$$\begin{aligned} \sigma_0(X) &= (X \ggg 7) \oplus (X \ggg 18) \oplus (X \gg 3), \\ \sigma_1(X) &= (X \ggg 17) \oplus (X \ggg 19) \oplus (X \gg 10). \end{aligned}$$

The state update function updates eight 32-bit chaining values, $A, B, ..., G, H$ in 64 steps as follows:

$$T_1 = H_i + \Sigma_1(E_i) + Ch(E_i, F_i, G_i) + K_i + W_i, \tag{24}$$
$$T_2 = \Sigma_0(A_i) + Maj(A_i, B_i, C_i), \tag{25}$$
$$A_{i+1} = T_1 + T_2, \tag{26}$$
$$B_{i+1} = A_i, \tag{27}$$
$$C_{i+1} = B_i, \tag{28}$$
$$D_{i+1} = C_i, \tag{29}$$
$$E_{i+1} = D_i + T_1, \tag{30}$$
$$F_{i+1} = E_i, \tag{31}$$
$$G_{i+1} = F_i, \tag{32}$$
$$H_{i+1} = G_i, \tag{33}$$

where K_i is a step constant and the function Ch, Maj, Σ_0 and Σ_1 are given as follows:

$$\begin{aligned}
Ch(X,Y,Z) &= XY \oplus \overline{X}Z,\\
Maj(X,Y,Z) &= XY \oplus YZ \oplus XZ,\\
\Sigma_0(X) &= (X \ggg 2) \oplus (X \ggg 13) \oplus (X \ggg 22),\\
\Sigma_1(X) &= (X \ggg 6) \oplus (X \ggg 11) \oplus (X \ggg 25).
\end{aligned}$$

After 64 step, a feedfoward process is executed with initial state variable by using word-wise addition modulo 2^{32}.

2.3 Meet-in-the-Middle Approach for Preimage Attack

We assume that a compression function F consists of a key scheduling function (KSF) and a round/step function as shown in Fig. 4. The function F has two inputs, an n-bit chaining variable H and an m-bit message M, and outputs an n-bit chaining variable G. The function KSF expands the message M, and provides them into the round/step function.

We consider a problem that given H and G, find a message M satisfying $G = F(H,M)$. This problem corresponds to the preimage attack on the compression function with a fixed input chaining variable. In this model, a feedforward function does not affect the attack complexity, since the targets H and G are arbitrary values. If we obtain a preimage from arbitrary values of H and G, we can also compute a preimage from H and $H \oplus G$ instead of G.

In the meet-in-the-middle preimage attack, we first divide the round function into two parts: the forward process (FP) and the backward process (BP) so that each process can compute an ℓ-bit meet point S independently. We also need independent words X and Y in KSF to compute S independently. The meet point S can be determined from FP and BP independently such that $S = FP(H,X)$ and $S = BP(G,Y)$.

If there are such two processes FP and BP, and independent words X and Y, we can obtain a message M satisfying S with a complexity of $2^{\ell/2}$ F evaluations, assuming that FP and BP are random ones, and the computation cost of BP is almost same as that of inverting function of BP. Since remaining internal state value

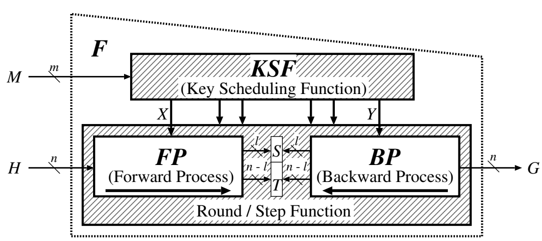

Fig. 4. Meet-in-the-middle approach

T is $(n - \ell)$ bits, the desired M can be obtained with a complexity of $2^{n-\ell/2}(= 2^{n-\ell+\ell/2})$. Therefore, if FP and BP up to the meet point S can be calculated independently, a preimage attack can succeed with a complexity of $2^{n-\ell/2}$. This type of preimage attacks on MD4 and MD5 was presented by Aoki and Sasaki [1].

In general, it is difficult to find such independent words in a complicated KSF. We developed new techniques to construct independent transforms in KSF by controlling internal variabes to obtain independent words.

3 Preimage Attack on Reduced-Round Tiger

In this section, we propose a preimage attack on 16-round Tiger with a complexity of 2^{161}. This variant shown in Fig. 5 consists of two pass functions and one key schedule function. First, we show properties of Tiger which are used for applying the meet-in-the-middle attack. Next, we show how to apply the meet-in-the-middle attack to Tiger, and then introduce the algorithm of our attack. Finally, we evaluate the required complexity and memory of our attack.

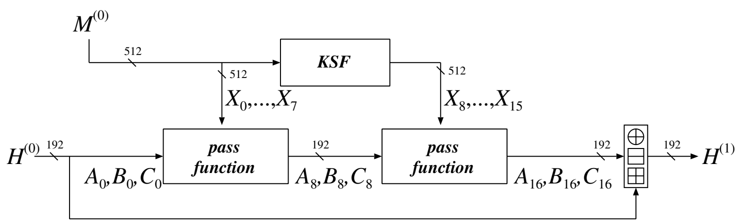

Fig. 5. Reduced-round Tiger (2-pass = 16-round)

3.1 Properties of Tiger

We show five properties of Tiger, which enable us to apply the meet-in-the-middle attack.

Property 1: *The pass function is easily invertible.*

Property 1 can be obtained from the design of the round function. From Eq. (1) to Eq. (3), A_i, B_i, and C_i can be determined from A_{i+1}, B_{i+1}, C_{i+1} and X_i. The computation cost is almost same as the cost of calculating A_{i+1}, B_{i+1} and C_{i+1} from A_i, B_i, C_i and X_i. Since the round function is invertible, we can construct the inverse pass function.

Property 2: *In the inverse pass function, the particular message words are independent of particular state value.*

The detail of the Property 2 is that once X_i, A_{i+3} B_{i+3} and C_{i+3} are fixed, then C_i, B_{i+1}, A_{i+2} and B_{i+2} can be determined from Eq. (1) to Eq. (3) independently of X_{i+1} and X_{i+2}. Thus the property 2 implies that X_{i+1} and X_{i+2} are independent of C_i in the inverse pass function.

Property 3: *In the round function,* C_{i+1} *is independent of odd bytes of* X_i*.*

The property 3 can be obtained from the property of the non-linear function *even*.

Property 4: *The key schedule function* KSF *is easily invertible.*

The property 4 implies that we can build the inverse key schedule function KSF^{-1}. Moreover, the computation cost of KSF^{-1} is almost the same as that of KSF.

Property 5: *In the inverse key schedule function* KSF^{-1}*, if input values are chosen appropriately, there are two independent transforms.*

The property 5 is one of the most important properties for our attack. In the next section, we show this in detail.

3.2 How to Obtain Two Independent Transforms in the KSF^{-1}

Since any input word of KSF^{-1} affects all output words of KSF^{-1}, it appears that there is no independent transform in the KSF^{-1} at first glance.

However, we analyzed the relation among the inputs and the outputs of KSF^{-1} deeply, and then found a technique to construct two independent transforms in the KSF^{-1} by choosing inputs carefully and controlling internal variables. Specifically, we can show that a change of input word X_8 only affects output words X_0, X_1, X_2 and X_3, and also modifications of X_{13}, X_{14} and X_{15} only affect X_5 and X_6 if these input words are chosen properly. We present the relation among inputs, outputs and internal variables of KSF^{-1} and then show how to build independent transforms in the KSF^{-1}.

As shown in Fig. 6, changes of inputs X_{13}, X_{14} and X_{15} only propagate internal variables Y_0, Y_1, Y_5, Y_6 and Y_7. If internal variables Y_6 and Y_7 are fixed

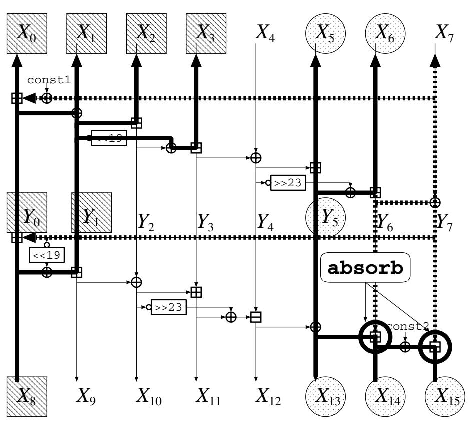

Fig. 6. Relation among inputs and outputs of KSF^{-1}

even when X_{13}, X_{14} and X_{15} are changed, it can be considered that an internal variable Y_0, Y_1 and an output X_7 are independent of changes of X_{13}, X_{14} and X_{15}. From Eq. (22) and (23), Y_6 and Y_7 can be fixed to arbitrary values by choosing X_{13}, X_{14} and X_{15} satisfying the following formulae:

$$X_{14} = Y_6 + X_{13}, \tag{34}$$

$$X_{15} = Y_7 - (X_{14} \oplus \texttt{const2}). \tag{35}$$

Therefore modifications of inputs X_{13}, X_{14} and X_{15} only propagate X_5 and X_6 by selecting these input values appropriately. In addition, a modification of X_8 only affects X_0, ..., X_3.

As a result, we obtain two independent transforms in KSF^{-1} by choosing X_{13}, X_{14} and X_{15} properly, since in this case a change of X_8 only affects X_0, ..., X_3, and changes of X_{13}, X_{14} and X_{15} only propagate X_5 and X_6.

3.3 Applying Meet-in-the-Middle Attack to Reduced-Round Tiger

We show the method for applying the meet-in-the-middle attack to Tiger by using above five properties. We define the meet point as 64-bit C_6, the process 1 as rounds 1 to 6, and the process 2 as rounds 7 to 16.

In the process 2, intermediate values A_9, B_9 and C_9 can be calculated from A_{16}, B_{16}, C_{16} and message words X_9 to X_{15}, since Tiger without the feedforward function is easily invertible. From the property 2, C_6 can be determined from A_8, B_8 and X_6. It is also observed that A_8 and B_8 are independent of X_8, because these values are calculated from A_9, B_9 and C_9. From the property 5, X_8 does not affect X_6. Therefore, C_6, the output of the process 2, can be determined from X_6, X_9 to X_{15}, A_{16}, B_{16} and C_{16}.

In the process 1, the output C_6 can be calculated from X_0 to X_5, A_0, B_0 and C_0. If some changes of the message words used in each process do not affect the message words used in the other process, C_6 can be determined independently in each process.

The message words X_0 to X_4 are independent of changes of X_6 and X_{13} to X_{15}, if X_9 to X_{12} are fixed and X_{13} to X_{15} are calculated as illustrated in the section 3.2. Although changes of X_{13}, X_{14} and X_{15} propagate X_5, from the property 3, C_6 in the process 1 is not affected by changes of odd bytes of X_5. Therefore, if even bytes of X_5 are fixed, C_6 in the process 1 can be determined independently from a change of X_5.

We show that the even bytes of X_5 can be fixed by choosing X_{11}, X_{12} and X_{13} properly. From Eq. (21), Y_5 is identical to X_{13} when X_{12} equals zero, and from Eq. (13), X_5 is identical to Y_5 when Y_4 equals zero. Thus X_5 is identical to X_{13} when both X_{12} and Y_4 are zero. Consequently, if the even bytes of X_{13} are fixed, and X_{12} and Y_4 equal zero, the even bytes of X_5 can be fixed. Y_4 can be fixed to zero by choosing X_{11} as $X_{11} \leftarrow \overline{X_{10}} \gg 23$. Therefore, if the following conditions are satisfied, C_6 in the process 1 can be independent of changes of X_{13}, X_{14} and X_{15}.

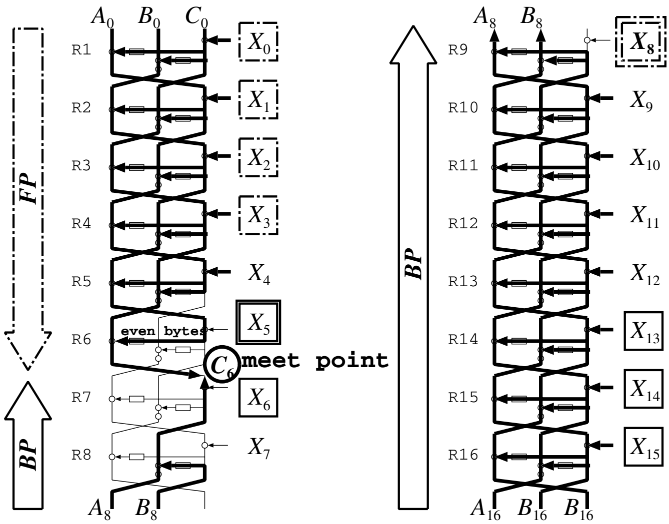

Fig. 7. Meet-in-the-middle attack on 16-round Tiger

- X_9 and X_{10} are fixed arbitrarily,
- $X_{11} = \overline{X_{10}} \gg 23,\ X_{12} = 0,$
- X_{13}, X_{14} and X_{15} are chosen properly.

By choosing inputs of the inverse pass function satisfying the above conditions, we can execute the process 1 and the process 2 independently. Specifically, if only X_{13}, X_{14} and X_{15} are treated as variables in the process 2, then the process 2 can be executed independently from the process 1. Similarly, if only X_8 is treated as a variable in the process 1, then the process 1 is independent of the process 2, as long as X_8 to X_{15} satisfy the above conditions. These results are shown in Fig. 7.

3.4 (Second) Preimage Attack on 16-Round Tiger Compression Function

We present the whole algorithm of the (second) preimage attack on the compression function of Tiger reduced to 16 rounds. The attack consists of three phases: preparation, first and second phase.

The preparation phase sets $X_i (i \in \{4, 7, 9, 10, 11, 12\})$, $Y_i (i \in \{2, 3, 4, 6, 7\})$ and even bytes of X_{13} as follows:

Preparation

1: Let A'_{16}, B'_{16} and C'_{16} be given targets. Choose A_0, B_0 and C_0 arbitrarily, and set A_{16}, B_{16} and C_{16} as follows:

$$A_{16} \leftarrow A_0 \oplus A'_{16},\ B_{16} \leftarrow B_0 - B'_{16},\ C_{16} \leftarrow C'_{16} - C_0.$$

2: Choose X_9, X_{10}, Y_6, Y_7 and even bytes of X_{13} arbitrarily, set X_{12} and Y_4 to zero, and set X_7, X_{11}, Y_2, Y_3 and X_4 as follows:

$$X_7 \leftarrow Y_6 \oplus Y_7,\ X_{11} \leftarrow \overline{X_{10}} \gg 23,\ Y_2 \leftarrow X_9 \oplus X_{10},\ Y_3 \leftarrow X_{11} - X_{10},\ X_4 \leftarrow Y_3.$$

The first phase makes a table of (C_6, odd bytes of X_{13}) pairs in the process 2 as follows:

First Phase

1: Choose odd bytes of X_{13} randomly.
2: Set X_5, X_6, X_{14} and X_{15} as follows:

$$X_5 \leftarrow X_{13}, X_6 \leftarrow Y_6 + X_{13}, X_{14} \leftarrow Y_6 + X_{13}, X_{15} \leftarrow Y_7 - ((Y_6 + X_{13}) \oplus \texttt{const2}).$$

3: Compute C_6 from $A_{16}, B_{16}, C_{16}, X_6$ and X_9 to X_{15}.
4: Place a pair (C_6, odd bytes of X_{13}) into a table.
5: If all 2^{32} possibilities of odd bytes of X_{13} have been checked, terminate this phase. Otherwise, set another value, which has not been set yet, to odd bytes of X_{13} and return to the step 2.

The second phase finds the desired message values X_0 to X_{15} in the process 1 by using the table as follows:

Second Phase

1: Choose X_8 randomly.
2: Set Y_0, Y_1, X_0, X_1, X_2 and X_3 as follows:

$$\begin{aligned}
Y_0 &\leftarrow X_8 - X_7,\\
Y_1 &\leftarrow X_9 + (X_8 \oplus (\overline{Y_7} \ll 19)),\\
X_0 &\leftarrow Y_0 + (X_7 \oplus \texttt{const1}),\\
X_1 &\leftarrow Y_0 \oplus Y_1,\\
X_2 &\leftarrow Y_2 - Y_1,\\
X_3 &\leftarrow Y_3 + (Y_2 \oplus (\overline{Y_1} \ll 19)).
\end{aligned}$$

3: Compute C_6 from X_0 to X_4, even bytes of X_5, A_0, B_0 and C_0.
4: Check whether this C_6 is in the table generated in the first phase. If C_6 is in the table, the corresponding X_0 to X_7 are a preimage for the compression function of the target $A'_{16}, B'_{16}, C'_{16}$ and successfully terminates the attack. Otherwise, set another value, which has not been set yet, to X_8 and return to the step 2.

By repeating the second phase about 2^{32} times for different choices of X_8, we expect to obtain a matched C_6. The complexity of the above algorithm is $2^{32}(= 2^{32} \cdot \frac{6}{16} + 2^{32} \cdot \frac{10}{16})$ compression function evaluations, and success probability is about 2^{-128}. By executing the above algorithm 2^{128} times with different fixed values, we can obtain a preimage of the compression function. In the preparation phase, A_0, B_0, C_0, X_9, X_{10}, Y_6, Y_7 and even bytes of X_{13} can be chosen arbitrarily. In other words, this attack can use these values as free words. These free words are enough for searching 2^{128} space. Accordingly, the complexity of the preimage attack on the compression function is $2^{160}(= 2^{32} \cdot 2^{128})$. Also, this algorithm requires 2^{32} 96-bit or $2^{35.6}$ bytes memory.

3.5 One-Block (Second) Preimage Attack on 16-Round Tiger

The preimage attack on the compression function can be extended to the one-block preimage attack on 16-round Tiger hash function. For extending the attack, A_0, B_0, C_0 are fixed to the IV words, the padding word X_7 is fixed to 447 encoded in 64-bit string, and the remaining 224 bits are used as free bits in the preparation phase. Although our attack cannot deal with another padding word X_6, the attack still works when the least significant bit of X_6 equals one.

Hence, the success probability of the attack on the hash function is half of that of the attack on the compression function. The total complexity of the one-block preimage attack on 16-round Tiger hash function is 2^{161} compression function computations.

This preimage attack can also be extended to the one-block second preimage attack directly. Our second preimage attack obtains a one-block preimage with the complexity of 2^{161}. Moreover, the complexity of our second preimage attack can be reduced by using the technique given in [4]. In this case, the second preimage attack obtains the preimage which consists of at least two message blocks with a complexity of 2^{160}.

4 Preimage Attack on Reduced-Round SHA-2

We apply our techniques to SHA-2 including both SHA-256 and SHA-512 in straightforward and present a preimage attack on SHA-2 reduced to 24 (out of 64 and 80, respectively) steps. We first check the properties of SHA-2, then introduce the algorithm of the preimage attack on 24-step SHA-2.

4.1 Properties of 24-Step SHA-2

We first check whether SHA-2 has similar properties of Tiger. The pass function of Tiger corresponds to the 16-step state update function of SHA-2, and the key schedule function of Tiger corresponds to the 16-step message expansion function of SHA-2. Since the state update function and the message expansion function of SHA-2 are easily invertible, the compression function of SHA-2 without the feedforward function is also invertible.

In the inverse state update function, $A_{18}, B_{18}, ..., H_{18}$ are determined from $A_{24}, B_{24}, ..., H_{24}$ and W_{18} to W_{23}, and A_{11} only depends on $A_{18}, ..., H_{18}$. Thus A_{11} is independent of W_{11} to W_{17} when $A_{18}, ..., H_{18}$ and W_{18} to W_{23} are fixed. It corresponds to the property 2 of Tiger.

Then we check whether there are independent transforms in the inverse message expansion function of SHA-2. It corresponds to the property 5 of Tiger. For

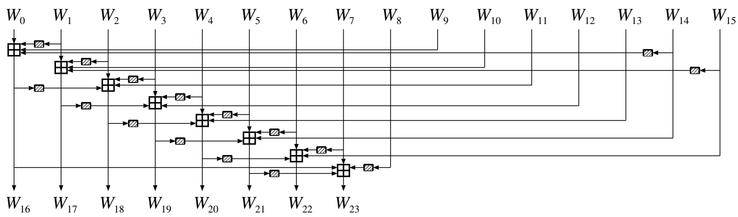

Fig. 8. Message expansion function of 24-step SHA-2

the 24-step SHA-2, 16 message words W_0 to W_{15} used in the first 16 steps are identical to input message blocks of the compression function, and 8 message words W_{16} to W_{23} used in the remaining eight steps are derived from W_0 to W_{15} by the message expansion function shown in Fig. 8. Table 3 shows the relation among message words in the message expansion function. For example, W_{16} is determined from W_{14}, W_9, W_1 and W_0. By using these relation and techniques introduced in previous sections, we can configure two independent transforms in the message expansion function of SHA-2.

We show that, in the inverse message expansion function of 24-step SHA-2, i) a change of W_{17} only affects W_0, W_1, W_3 and W_{11}, and ii) W_{19}, W_{21} and W_{23} only affect W_{12} by using the message modification techniques. In Tab. 3, asterisked values are variables of i), and underlined values are variables of ii).

First, we consider the influence of W_{23}. Though W_{23} affects W_7, W_8, W_{16} and W_{21}, this influence can be absorbed by modifying $W_{21} \rightarrow W_{19} \rightarrow W_{12}$. Consequently, we obtain a result that W_{19}, W_{21} and W_{23} only affect W_{12} by choosing these values properly, since W_{12} does not affect any other values in the inverse message expansion function.

Similarly, we consider the influence of W_{17} in the inverse message expansion function. W_{17} affects W_1, W_2, W_{10} and W_{15}. This influence can be absorbed by modifying $W_1 \rightarrow W_0$. W_{17} is also used for generating W_{19}. In order to cancel this influence, $W_3 \rightarrow W_{11}$ are also modified. As a result, we obtain a result that W_{17} only affects W_0, W_1, W_3 and W_{11} by choosing these values appropriately.

Table 3. Relation among message values W_{16} to W_{23}

computed value	values for computing
W_{16}	W_{14}, W_9, W_1*, W_0*
$W_{17}*$	$W_{15}, W_{10}, W_2, W_1*$
W_{18}	$W_{16}, W_{11}*, W_3*, W_2$
$\underline{W_{19}}$	$W_{17}*, \underline{W_{12}}, W_4, W_3*$
W_{20}	W_{18}, W_{13}, W_5, W_4
$\underline{W_{21}}$	$\underline{W_{19}}, W_{14}, W_6, W_5$
W_{22}	W_{20}, W_{15}, W_7, W_6
$\underline{W_{23}}$	$\underline{W_{21}}, W_{16}, W_8, W_7$

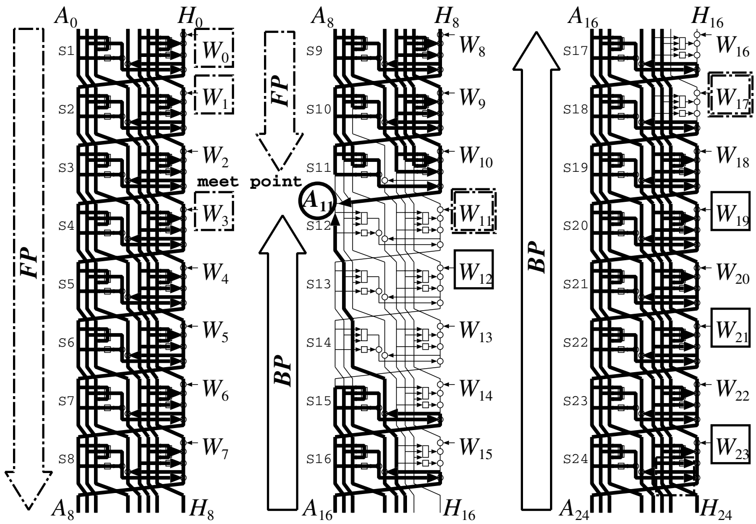

Fig. 9. Meet-in-the-middle attack on 24-step SHA-2

4.2 (Second) Preimage Attack on 24-Step SHA-256 Compression Function

As shown in Fig. 9, we define the meet point as 32-bit A_{11}, the process 1 as steps 1 to 11, and the process 2 as steps 12 to 24. In the process 1, A_{11} can be derived from $A_0, ..., H_0$ and W_0 to W_{10}. Similarly, in the process 2, A_{11} can be determined from $A_{24}, ..., H_{24}$ and W_{18} to W_{23}. Since the process 1 and process 2 are independent of each other for A_{11} by using the above properties of SHA-2, we apply the meet-in-the-middle attack to SHA-2 as follows:

Preparation

1: Let $A'_{24}, ..., H'_{24}$ be given targets. Choose $A_0, ..., H_0$ arbitrarily, and compute $A_{24}, ..., H_{24}$ by the feedforward function.

2: Choose 32-bit value CON and $W_i (i \in \{2, 4, 5, 6, 7, 8, 9, 10, 13, 14, 15, 16, 18\})$ arbitrarily, and then calculate W_{20} and W_{22}.

First Phase

1: Choose W_{23} randomly.

2: Determine W_{21}, W_{19} and W_{12} as follows[1]:

$$W_{21} \leftarrow \sigma_1^{-1}(W_{23} - W_{16} - \sigma_0(W_8) - W_7),$$
$$W_{19} \leftarrow \sigma_1^{-1}(W_{21} - W_{14} - \sigma_0(W_6) - W_5),$$
$$W_{12} \leftarrow W_{19} - \texttt{CON}.$$

[1] The method how to calculate σ_1^{-1} is illustrated in the appendix.

3: Compute A_{11} from $A_{24}, ..., H_{24}$ and W_{18} to W_{23}.
4: Place a pair (A_{11}, W_{23}) into a table.
5: If 2^{16} pairs of (A_{11}, W_{23}) have been listed in the table, terminate this algorithm. Otherwise, set another value, which has not been set yet, to W_{23} and return to the step 2.

Second Phase

1: Choose W_{17} randomly.
2: Determine W_0, W_1, W_3 and W_{11} as follows:

$$\begin{aligned} W_1 &\leftarrow W_{17} - \sigma_1(W_{15}) - W_{10} - \sigma_0(W_2), \\ W_0 &\leftarrow W_{16} - \sigma_1(W_{14}) - W_9 - \sigma_0(W_1), \\ W_3 &\leftarrow \texttt{CON} - \sigma_1(W_{17}) - \sigma_0(W_4), \\ W_{11} &\leftarrow W_{18} - \sigma_1(W_{16}) - \sigma_0(W_3) - W_2. \end{aligned}$$

3: Compute A_{11} from $A_0, ..., H_0$ and W_0 to W_{10}.
4: Check whether this A_{11} is in the table generated in the first phase. If A_{11} is in the table, the corresponding W_0 to W_{23} is a preimage of the compression function of the target $A'_{24}, ..., H'_{24}$ and successfully terminates the attack. Otherwise, set another value, which has not been set yet, to W_{17} and return to the step 2.

By repeating the second phase about 2^{16} times for different W_{17}, we expect to obtain a matched A_{11}. The complexity of the preimage attack on the compression function is $2^{240} (= 2^{256-32/2})$ compression function evaluations. The required memory is 2^{16} 64-bit or 2^{19} bytes. In this attack, the words $A_0, ..., H_0$, CON and $W_i (i \in \{2, 4, 5, 6, 7, 8, 9, 10, 13, 14, 15, 16, 18\})$ can be used as free words. The total free words are 22 words or 704 bits.

4.3 One-Block (Second) Preimage Attack on 24-Step SHA-2 Hash Function

The preimage attack on the compression function can be extended to the (second) preimage attack on the hash function directly, since our preimage attack can obtain random preimages from an arbitrary IV and an arbitrary target, and can deal with the padding words W_{14} and W_{15}. Thus the complexities of the preimage attack and the second preimage attack on 24-step SHA-256 are 2^{240}. Furthermore, this attack can also be extended to the (second) preimage attack on 24-step SHA-512. The complexities of the (second) preimage attack on 24-step SHA-512 are $2^{480} (= 2^{512-64/2})$.

5 Conclusion

In this paper, we have shown preimage attacks on reduced-round Tiger, reduced-step SHA-256 and reduced-step SHA-512. The proposed attacks are based on meet-in-the-middle attack. We developed new techniques to find "independent words" of the compression functions. In the attack on reduced-round Tiger, we found the "independent transforms" in the message schedule function by adjusting the internal variables, then we presented there are independent words in the compression function of Tiger. In the attack on reduced-round SHA-2, we found the "independent transforms" in the message expansion function by modifying the messages, then we showed that there are independent words in the compression function of SHA-2.

Our preimage attack can find a preimage of 16-step Tiger, 24-step SHA-256 and 24-step SHA-512 with a complexity of 2^{161}, 2^{240} and 2^{480}, respectively. These preimage attacks can be extended to second preimage attacks with the almost same complexities. Moreover, our (second) preimage attacks can find a one-block preimage, since it can obtain random preimages from an arbitrary IV an arbitrary target, and can also deal with the padding words.

Acknowledgments. The authors thank to the anonymous referees for their valuable comments.

References

1. Aoki, K., Sasaki, Y.: Preimage attacks on one-block MD4, 63-step MD5 and more. In: Avanzi, R., Keliher, L., Sica, F. (eds.) SAC 2008. LNCS, pp. 82–98. Springer, Heidelberg (2008)
2. Anderson, R., Biham, E.: Tiger: A fast new hash function. In: Gollmann, D. (ed.) FSE 1996. LNCS, vol. 1039, pp. 89–97. Springer, Heidelberg (1996)
3. Dobbertin, H.: Cryptanalysis of MD4. In: Gollmann, D. (ed.) FSE 1996. LNCS, vol. 1039, pp. 53–69. Springer, Heidelberg (1996)
4. Indesteege, S., Preneel, B.: Preimages for reduced-round Tiger. In: Lucks, S., Sadeghi, A.-R., Wolf, C. (eds.) WEWoRC 2007. LNCS, vol. 4945, pp. 90–99. Springer, Heidelberg (2008)
5. Kelsey, J., Lucks, S.: Collisions and near-collisions for reduced-round Tiger. In: Robshaw, M.J.B. (ed.) FSE 2006. LNCS, vol. 4047, pp. 111–125. Springer, Heidelberg (2006)
6. Leurent, G.: MD4 is not one-way. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 412–428. Springer, Heidelberg (2008)
7. Mendel, F., Pramstaller, N., Rechberger, C.: A (second) preimage attack on the GOST hash function. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 224–234. Springer, Heidelberg (2008)
8. Mendel, F., Preneel, B., Rijmen, V., Yoshida, H., Watanabe, D.: Update on Tiger. In: Barua, R., Lange, T. (eds.) INDOCRYPT 2006. LNCS, vol. 4329, pp. 63–79. Springer, Heidelberg (2006)
9. Mendel, F., Rijmen, V.: Cryptanalysis of the Tiger hash function. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 536–550. Springer, Heidelberg (2007)
10. Sasaki, Y., Aoki, K.: Preimage Attacks on MD, HAVAL, SHA, and Others. In: Rump session at CRYPTO 2008 (2008)
11. Wang, X., Yu, H.: How to break MD5 and other hash functions. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 19–35. Springer, Heidelberg (2005)
12. Wang, X., Yin, Y.L., Yu, H.: Finding collisions in the full SHA-1. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 17–36. Springer, Heidelberg (2005)

Appendix A

Here, we show how to calculate the inverse function σ_1^{-1}. Let $(x_{31}, ..., x_0)$ and $(y_{31}, ..., y_0)$ be outputs and inputs of σ_1^{-1} respectively, where $x_i, y_i \in \{0, 1\}$, and x_{31} and y_{31} are the most significant bit. The inverse function σ_1^{-1} is calculated as follows:

$$(x_{31}, x_{30}, ..., x_0)^t = M_{\sigma_1^{-1}} \cdot (y_{31}, y_{30}, ..., y_0)^t,$$

where

$$M_{\sigma_1^{-1}} = \begin{pmatrix}
1&0&0&1&0&1&1&1&0&0&1&0&0&0&1&0&1&1&1&0&1&0&1&0&0&0&0&1&1&0&0&0\\
0&1&0&0&1&0&1&1&1&0&0&1&0&0&0&1&0&1&1&1&0&1&0&1&0&0&0&0&1&1&0&0\\
0&0&1&0&0&1&0&1&1&1&0&0&1&0&0&0&1&0&1&1&1&0&1&0&1&0&0&0&0&1&1&0\\
1&0&0&0&0&1&0&1&1&1&0&0&0&1&1&0&1&0&1&1&0&1&1&1&0&1&0&1&1&0&1&1\\
0&1&1&1&0&0&0&1&0&1&0&1&1&1&0&0&0&0&0&0&1&0&1&1&0&0&1&1&1&0&0&1\\
1&0&0&1&1&1&0&0&0&0&1&1&0&0&1&1&1&0&1&1&1&1&1&1&0&0&0&1&0&0&0&0\\
0&1&0&0&1&1&1&0&0&0&0&1&1&0&0&1&1&1&0&1&1&1&1&1&1&0&0&0&1&0&0&0\\
0&0&1&0&0&1&1&1&0&0&0&0&1&1&0&0&1&1&1&0&1&1&1&1&1&1&0&0&0&1&0&0\\
1&0&0&0&0&1&0&0&1&0&1&0&0&1&0&0&1&0&0&1&1&1&0&1&1&1&1&1&1&0&1&0\\
0&1&0&0&0&0&1&0&0&1&0&1&0&0&1&0&0&1&0&0&1&1&1&0&1&1&1&1&1&1&0&1\\
0&0&0&1&0&0&1&0&1&0&0&1&0&1&1&0&0&1&1&1&0&1&1&1&1&1&1&0&1&0&1&0\\
0&0&0&0&1&0&0&1&0&1&0&0&1&0&1&1&0&0&1&1&1&0&1&1&1&1&1&1&0&1&0&1\\
0&0&1&1&0&1&1&1&0&0&0&1&1&0&1&0&1&1&0&0&1&1&0&1&0&1&1&0&1&1&1&0\\
1&0&0&0&1&1&0&0&1&0&1&0&1&1&1&1&1&0&0&0&1&1&0&0&1&0&1&0&1&1&1&1\\
0&1&1&1&0&1&0&1&1&1&1&0&1&0&0&0&1&0&0&1&0&1&1&0&1&1&0&0&0&0&1&1\\
1&0&0&1&1&1&1&0&0&1&1&0&1&0&0&1&1&1&1&1&0&0&0&1&1&1&1&0&1&1&0&1\\
0&1&1&1&1&1&0&0&1&0&0&0&1&0&1&1&1&0&1&0&1&0&0&0&0&1&1&0&0&0&1&0\\
1&0&1&0&1&0&0&1&0&1&1&0&0&1&1&1&0&0&1&1&1&1&1&0&0&0&1&0&1&0&0&1\\
1&1&1&1&0&0&0&0&0&0&1&0&1&1&1&0&0&0&1&0&0&1&0&1&1&0&0&1&1&0&0&0\\
1&1&1&0&1&1&1&1&0&0&1&1&0&1&0&1&1&1&1&1&1&0&0&0&1&1&0&1&0&1&0&0\\
1&1&1&0&0&0&0&0&1&0&1&1&1&0&0&0&0&0&0&1&0&1&1&0&0&1&1&1&0&0&1&0\\
1&1&1&0&0&1&1&1&0&1&1&1&1&1&1&0&1&1&1&0&0&0&0&1&0&0&1&0&0&0&0&1\\
1&1&0&1&0&1&1&1&0&0&1&0&0&0&1&0&1&1&0&0&1&0&1&0&0&0&0&1&1&1&0&0\\
0&1&1&0&1&0&1&1&1&0&0&1&0&0&0&1&0&1&1&0&0&1&0&1&0&0&0&0&1&1&1&0\\
1&0&1&0&0&0&1&0&1&1&1&0&1&0&1&0&0&1&0&1&1&0&0&0&1&0&0&1&1&1&1&1\\
1&1&1&1&0&1&0&1&1&1&1&0&1&0&0&0&1&0&0&1&0&1&1&0&1&1&0&0&0&0&1&1\\
1&1&0&1&1&1&1&0&0&1&1&0&1&0&0&1&1&1&1&1&0&0&0&1&1&1&1&0&1&1&0&1\\
0&1&0&1&1&1&0&0&1&0&0&0&1&0&1&1&1&0&1&0&1&0&0&0&0&1&1&0&0&0&1&0\\
0&0&1&0&1&1&1&0&0&1&0&0&0&1&0&1&1&1&0&1&0&1&0&0&0&0&1&1&0&0&0&1\\
1&0&1&1&0&0&1&1&1&0&1&1&1&1&1&1&0&1&0&1&0&0&0&0&1&0&0&1&0&1&0&0\\
1&1&0&0&1&1&1&0&1&1&1&1&1&1&0&1&0&1&0&0&0&0&1&0&0&1&0&1&0&0&1&0\\
0&1&1&0&0&1&1&1&0&1&1&1&1&1&1&0&1&0&1&0&0&0&0&1&0&0&1&0&1&0&0&1
\end{pmatrix}.$$