Preimages for Reduced-Round Tiger — Source link

Sebastiaan Indesteege, Bart Preneel

Institutions: Katholieke Universiteit Leuven

Published on: 01 Jan 2008

Topics: Preimage attack, Collision attack, Security of cryptographic hash functions, MDC-2 and Cryptographic hash function

Preimages for Reduced-Round Tiger[⋆,⋆⋆]

Sebastiaan Indesteege[⋆⋆⋆] and Bart Preneel

Katholieke Universiteit Leuven, Dept. ESAT/SCD-COSIC,
Kasteelpark Arenberg 10, B-3001 Heverlee, Belgium
`{sebastiaan.indesteege, bart.preneel}@esat.kuleuven.be`

Abstract. The cryptanalysis of the cryptographic hash function Tiger has, until now, focussed on finding collisions. In this paper we describe a preimage attack on the compression function of Tiger-12, i.e., Tiger reduced to 12 rounds out of 24, with a complexity of $2^{63.5}$ compression function evaluations. We show how this can be used to construct second preimages with complexity $2^{63.5}$ and first preimages with complexity $2^{64.5}$ for Tiger-12. These attacks can also be extended to Tiger-13 at the expense of an additional factor of 2^{64} in complexity.

Key words: Tiger, hash functions, preimages

1 Introduction

A cryptographic hash function is expected to possess three properties: collision resistance, preimage resistance and second preimage resistance. While other properties exist, the above three are the most well known.

Collision resistance: It is difficult to find two distinct messages $m \neq m'$ that hash to the same result, i.e., $h(m) = h(m')$.

Preimage resistance: When given a hash result y (for which it holds that $\exists x : h(x) = y$), it is difficult to find a message m which hashes to y, i.e., $h(m) = y$.

Second preimage resistance: When given a message m, it is difficult to find a message $m' \neq m$ that hashes to the same result as the given message, i.e., $h(m) = h(m')$.

There are generic attacks that apply to any hash function and whose time complexity only depends on the size of the hash result. Collisions for a hash function with an n-bit result can be found in time $2^{n/2}$ using a birthday attack [6], and preimages can be found by brute force in time 2^n. Weaker attacks may aim

[⋆] The final publication is available at www.springerlink.com.

[⋆⋆] This work was supported in part by the Concerted Research Action (GOA) Ambiorics 2005/11 of the Flemish Government, by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy), and by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT.

[⋆⋆⋆] F.W.O. Research Assistant, Fund for Scientific Research – Flanders (Belgium).

at finding pseudo-collisions, where slight differences in the hash results are allowed, or pseudo-near-collisions, where differences may also appear in the initial chaining values.

All attacks on the cryptographic hash function Tiger [1] have so far been collision attacks. Kelsey and Lucks [3] showed a collision attack on Tiger reduced to 16 rounds with a complexity of 2^{44} compression function evaluations. Mendel et al. [4] extended this to a collision attack on 19 rounds of Tiger with a complexity of 2^{62} compression function evaluations. In both papers some weaker attacks (e.g. pseudo-collisions) for a larger number of rounds were also shown. These results were further improved by Mendel et al. [5] towards a pseudo-near-collision for the full hash function and a pseudo-collision for 23 rounds of Tiger.

We focus on finding preimages for reduced variants of Tiger instead. More specifically, we describe a method to find first and second preimages for 12 and 13 rounds reduced Tiger. This method is conceptually similar to Dobbertin's preimage attack on reduced MD4 [2]. Our attack finds first and second preimages for Tiger-12 with a complexity of $2^{64.5}$ and $2^{63.5}$ compression function evaluations, respectively. It can be extended to Tiger-13, where the complexities become $2^{128.5}$ and $2^{127.5}$, respectively. As Tiger has a digest size of 192 bits, the theoretical complexity for finding first or second preimages is 2^{192} compression function evaluations. To the best of our knowledge, this is the first result concerning preimages for reduced round Tiger.

The structure of the paper is as follows. In Sect. 2, the Tiger hash function is described, along with the notation that will be used throughout the paper. Section 3 describes a preimage attack on three rounds of Tiger. The three round preimage attack is then used as a building block to construct preimages for the compression function of Tiger-12 and Tiger-13 in Sect. 4. Then, in Sect. 5 it is shown how first and second preimages for these reduced variants of the Tiger hash function can be constructed. Finally, Sect. 6 presents our conclusions.

2 Description of Tiger

Tiger [1] is an iterative cryptographic hash function, designed by Anderson and Biham in 1996. It has an output size of 192 bits. Truncated variants with a digest size of 160 and 128 bits were also defined. It was designed for 64-bit architectures and hence all words are 64 bits wide and arithmetic is performed modulo 2^{64}. Tiger uses the little-endian byte ordering.

First, the message to be hashed is padded by appending a single "1"-bit and as many "0"-bits as necessary to make the message length 64 bits less than the next multiple of 512 bits. Then the message length (in bits) is appended as a 64-bit unsigned integer. After this procedure, the padded message consists of an integer number of 512-bit blocks. Then, Tiger's compression function is applied iteratively to each 512-bit block of the padded message.

Tiger's compression function operates on a 192-bit chaining value and a 512-bit message block. The message block is split into eight 64-bit words X_i. The 192-bit chaining value is split into three 64-bit words which are used as the

Table 1. Notations

$X + Y$	Addition of X and Y modulo 2^{64}
$X - Y$	Subtraction of X and Y modulo 2^{64}
$X \times Y$	Multiplication of X and Y modulo 2^{64}
$X \oplus Y$	Bit-wise exclusive or of X and Y
$\overline{X}$	Bit-wise complement of X
$X \ll n$	Logical left bit shift of X by n positions
$X \gg n$	Logical right bit shift of X by n positions
$X\|\|Y$	The concatenation of X and Y
X_i	The i-th expanded message word
Y_i	The i-th intermediate value of the key schedule algorithm
A_i, B_i, C_i	State variables at the output of round i, $0 \le i < 24$
K_i	The round constant used in round i, $0 \le i < 24$
K_i^{-1}	Multiplicative inverse of K_i modulo 2^{64}
$T_1,\ldots,T_4$	The four 8-to-64-bit S-boxes used in Tiger

initial state variables A_{-1}, B_{-1} and C_{-1}. The compression function consists of three passes of 8 rounds of a state update transformation (24 rounds in total), each using one X_i to update the three state variables A_i, B_i and C_i. Table 1 summarises the notations used in this paper.

The i-th round of Tiger ($0 \le i < 24$) is depicted in Fig. 1. Equivalently, the state update transformation can be described by the following equations:

$$\begin{aligned} A_i &= K_i \times (B_{i-1} + \text{odd}\,(C_{i-1} \oplus X_i)) \quad , \\ B_i &= C_{i-1} \oplus X_i \quad , \\ C_i &= A_{i-1} - \text{even}\,(C_{i-1} \oplus X_i) \quad . \end{aligned} \tag{1}$$

In every round, a round constant K_i is used. These constants are given by:

$$K_i = \begin{cases} 5 & \text{if } 0 \le i < 8 \;\; , \\ 7 & \text{if } 8 \le i < 16 \;\; , \\ 9 & \text{if } 16 \le i < 24 \;\; . \end{cases} \tag{2}$$

The non-linear functions odd($\cdot$) and even($\cdot$) are defined as follows.

$$\begin{aligned} \text{odd}(c_7\|\ldots\|c_0) &= T_4[c_1] \oplus T_3[c_3] \oplus T_2[c_5] \oplus T_1[c_7] \;\; , \\ \text{even}(c_7\|\ldots\|c_0) &= T_1[c_0] \oplus T_2[c_2] \oplus T_3[c_4] \oplus T_4[c_6] \;\; . \end{aligned} \tag{3}$$

Here, c_i denotes the i-th byte of a 64-bit word, using the little-endian byte ordering, i.e., c_0 is the least significant byte.[1] Both functions use four 8-to-64-bit S-boxes, T_1 through T_4. Note that both functions only use four out of eight input bytes, and thus map 32 bits to 64 bits. They are called odd($\cdot$) and even($\cdot$) because they operate on the odd, respectively even bytes of the input word.

The first eight message words X_i, $0 \le i < 8$, are taken directly from the message block. The message words $X_8,\ldots,X_{15}$ are derived from $X_0,\ldots,X_7$ using an

[1] Note that there was a misinterpretation of the byte order in [3,4]. The attacks described there can however be modified to overcome this problem. [5]

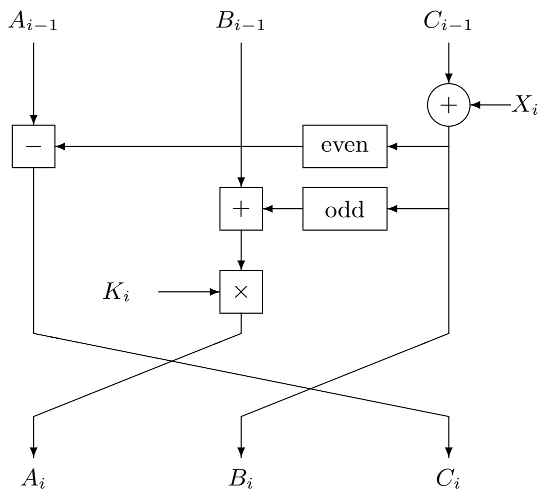

Fig. 1. The state update transformation of Tiger

algorithm which the designers of Tiger refer to as the key schedule algorithm [1]. Then, using the same algorithm, $X_{16},\ldots,X_{23}$ are determined from $X_8,\ldots,X_{15}$. This key schedule algorithm consists of two passes, given by the following equations:

$$\begin{aligned}
Y_0 &= X_0 - (X_7 \oplus \texttt{A5}\ldots\texttt{A5}_x) \ , & X_8 &= Y_0 + Y_7 \ , \\
Y_1 &= X_1 \oplus Y_0 \ , & X_9 &= Y_1 - \big(X_8 \oplus (\overline{Y_7} \ll 19)\big) \ , \\
Y_2 &= X_2 + Y_1 \ , & X_{10} &= Y_2 \oplus X_9 \ , \\
Y_3 &= X_3 - \big(Y_2 \oplus (\overline{Y_1} \ll 19)\big) \ , & X_{11} &= Y_3 + X_{10} \ , \\
Y_4 &= X_4 \oplus Y_3 \ , & X_{12} &= Y_4 - \big(X_{11} \oplus (\overline{X_{10}} \gg 23)\big) \ , \\
Y_5 &= X_5 + Y_4 \ , & X_{13} &= Y_5 \oplus X_{12} \ , \\
Y_6 &= X_6 - \big(Y_5 \oplus (\overline{Y_4} \gg 23)\big) \ , & X_{14} &= Y_6 + X_{13} \ , \\
Y_7 &= X_7 \oplus Y_6 \ . & X_{15} &= Y_7 - (X_{14} \oplus \texttt{01}\ldots\texttt{EF}_x) \ .
\end{aligned} \tag{4}$$

Finally, after 24 rounds, the initial state variables are fed forward, using a combination of exclusive or, subtraction and addition.

$$\begin{aligned}
A^\star &= A_{-1} \oplus A_{23} \ , \\
B^\star &= B_{-1} - B_{23} \ , \\
C^\star &= C_{-1} + C_{23} \ .
\end{aligned} \tag{5}$$

The 192-bit output of the compression function is $A^\star||B^\star||C^\star$, i.e., the concatenation of $A^\star$, $B^\star$ and $C^\star$.

3 Preimages for Three Rounds of Tiger

In this section we describe a solution due to Mendel et al. [4] to the problem of finding preimages for three rounds of the state update transformation of Tiger. There is always exactly one solution, which can be found in constant time. Although rather straightforward, it will prove to be a useful building block in preimage attacks on a larger number of Tiger rounds.

More in detail, we are given A_{-1}, B_{-1}, C_{-1}, A_2, B_2 and C_2 and want to determine the three message words X_0, X_1 and X_2 such that the constraints originating from the state update transformation are satisfied. Note that, without knowing any of the message words, all the state variables in these three rounds can already be determined. Indeed, from (1) it follows that

$$\begin{aligned}
A_1 &= C_2 + \mathrm{even}\,(B_2) \ , \\
B_1 &= \left(A_2 \times K_2^{-1}\right) - \mathrm{odd}\,(B_2) \ , \\
B_0 &= \left(A_1 \times K_1^{-1}\right) - \mathrm{odd}\,(B_1) \ , \\
A_0 &= K_0 \times (B_{-1} + \mathrm{odd}\,(B_0)) \ , \\
C_0 &= A_{-1} - \mathrm{even}\,(B_0) \ , \\
C_1 &= A_0 - \mathrm{even}\,(B_1) \ .
\end{aligned} \tag{6}$$

Note that each K_i as given in (2) is coprime with 2^{64} so its multiplicative inverse modulo 2^{64} exists and can be computed easily. Knowing the state variables, it is trivial to determine X_0, X_1 and X_2.

$$\begin{aligned}
X_0 &= C_{-1} \oplus B_0 \ , \\
X_1 &= C_0 \oplus B_1 \ , \\
X_2 &= C_1 \oplus B_2 \ .
\end{aligned} \tag{7}$$

This procedure is fully deterministic and always gives exactly one solution. The time complexity of this procedure is equivalent to three rounds of Tiger.

Of course this can equally be applied to any three consecutive rounds of Tiger, as part of a larger attack. To conclude, control over three consecutive expanded message words yields complete control over the intermediate state of Tiger.

4 Preimages for the Compression Function of Tiger-12

In this section, we first describe a method to find preimages for the compression function of Tiger, reduced to 12 rounds. Then we extend this to Tiger-13, i.e., Tiger reduced to 13 rounds.

Given the algorithm from Sect. 3, one can easily find sets of expanded message words X_i which ensure that the output of the compression function of Tiger (or a round-reduced version thereof) is equal to some desired value. However, if the number of attacked rounds is greater than eight there is no guarantee that these expanded message words satisfy the constraints from the key schedule algorithm. For eight or less rounds of Tiger, the message expansion becomes trivial, as each of the first eight expanded message words is under direct control of an adversary. Hence also finding preimages for these variants of Tiger is trivial by making arbitrary choices and using the algorithm from Sect. 3 for the last three rounds.

The circular dependency can be broken by guessing some intermediate variable(s) and later verifying if the guess was correct. If the guess was wrong, the attack is simply repeated. Hence the time complexity of the attack is highly dependent on the probability that the correct guess was made. Since we assume that every value for the guessed variables is equally likely, this probability is equal to 2^{-n} where n is the total number of guessed bits.

Conceptually, this approach is very similar to the work of Dobbertin [2] on finding preimages for a reduced variant of MD4. Of course the similarity only exists on a very high level, due to the fact that MD4 and Tiger are very different hash functions.

4.1 Algorithm

In this section, a detailed description of the algorithm for finding preimages for the compression function of Tiger-12 is given. As we are given the desired input and output chaining values, the feed-forward given in (5) can easily be removed. Therefore, the state variables A_{-1}, B_{-1}, C_{-1}, A_{11}, B_{11} and C_{11} are known at the beginning of the attack.

1. Make arbitrary choices for the message words used in the four last rounds, (i.e. X_8, X_9, X_{10} and X_{11}). The state update transformation can be used in the backwards direction to determine A_7, B_7 and C_7, as follows:
$$\begin{cases} A_{i-1} = C_i + \text{even}(B_i) \ , \\ B_{i-1} = \left(A_i \times K_i^{-1}\right) - \text{odd}(B_i) \ , \\ C_{i-1} = B_i \oplus X_i \ . \end{cases} \qquad \text{for } i = 11, \ldots, 8 \tag{8}$$
2. Guess Y_7, an intermediate value of the key schedule algorithm. This 64-bit guess is the only guess that will be made in the attack. It will be verified in the final step of the attack.
3. The message words X_8 through X_{11} are normally computed from the key schedule. These equations can easily be inverted to find the intermediate values Y_0, Y_1, Y_2 and Y_3 for which the values chosen in step 1 will appear:
$$\begin{aligned} Y_0 &= X_8 - Y_7 \ , \\ Y_1 &= X_9 + \left(X_8 \oplus \left(\overline{Y_7} \ll 19\right)\right) \ , \\ Y_2 &= X_{10} \oplus X_9 \ , \\ Y_3 &= X_{11} - X_{10} \ . \end{aligned} \tag{9}$$
This step is deterministic and always leads to a single solution. Looking further at the key schedule, the message words X_1 through X_3 can also be determined uniquely:
$$\begin{aligned} X_1 &= Y_1 \oplus Y_0 \ , \\ X_2 &= Y_2 - Y_1 \ , \\ X_3 &= Y_3 + \left(Y_2 \oplus \left(\overline{Y_1} \ll 19\right)\right) \ . \end{aligned} \tag{10}$$

4. Choose X_7 (there are 2^{64} choices) and compute X_0 using the key schedule:

$$X_0 = Y_0 + (X_7 \oplus \texttt{A5A5A5A5A5A5A5A5}_x) \tag{11}$$

5. Now, the first four expanded message words (i.e. X_0 through X_3) are known. The state update transformation can thus be used in the forward direction to calculate A_3, B_3 and C_3.

$$\begin{cases} A_i = K_i \times (B_{i-1} + \text{odd}\,(C_{i-1} \oplus X_i)) \ , \\ B_i = C_{i-1} \oplus X_i \ , \\ C_i = A_{i-1} - \text{even}\,(C_{i-1} \oplus X_i) \ . \end{cases} \qquad \text{for } i = 0, \ldots, 3 \tag{12}$$

Similarly, as X_7 is known, the state update transformation can be applied in the backwards direction to calculate A_6, B_6 and C_6.

$$\begin{aligned} A_6 &= C_7 + \text{even}\,(B_7) \ , \\ B_6 &= \left(A_7 \times K_7^{-1}\right) - \text{odd}\,(B_7) \ , \\ C_6 &= B_7 \oplus X_7 \ . \end{aligned} \tag{13}$$

6. Note that, because A_3, B_3, C_3, A_6, B_6 and C_6 are now known, the algorithm from Sect. 3 can be applied to determine the unique solution for X_4, X_5 and X_6.

$$\begin{aligned} A_5 &= C_6 + \text{even}\,(B_6) \ , \\ B_5 &= \left(A_6 \times K_6^{-1}\right) - \text{odd}\,(B_6) \ , \\ B_4 &= \left(A_5 \times K_5^{-1}\right) - \text{odd}\,(B_5) \ , \\ A_4 &= K_4 \times (B_3 + \text{odd}\,(B_4)) \ , \\ C_4 &= A_3 - \text{even}\,(B_4) \ , \\ C_5 &= A_4 - \text{even}\,(B_5) \ , \\ X_4 &= C_3 \oplus B_4 \ , \\ X_5 &= C_4 \oplus B_5 \ , \\ X_6 &= C_5 \oplus B_6 \ . \end{aligned} \tag{14}$$

7. Finally, apply the key schedule, which is given in (4), to compute the correct value for Y_7 from the message words X_0 through X_7, all of which have now been determined. Verify if the guess for Y_7 made in step 2 is correct. If it is, a preimage has been found. If not, restart from step 4 with a different choice for X_7.

The probability that the guess for Y_7 is correct is 2^{-64} so we expect to find a preimage after 2^{64} tries. Note that one attempt requires just 8 rounds of the state update transformation and 5 equations of the key schedule algorithm, which is only about 2/3 of the computations of a compression function evaluation. For simplicity, we assume that every equation of the key schedule algorithm takes an equivalent amount of work. Hence, the overall complexity of the attack is equivalent to slightly less than $2^{63.5}$ evaluations of the compression function.

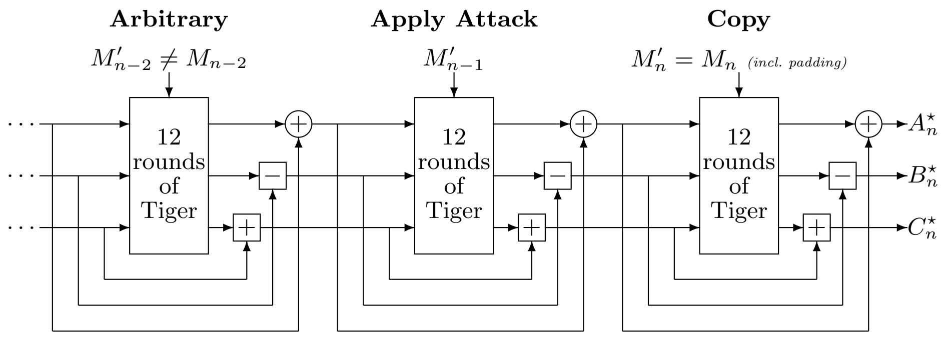

Fig. 2. Constructing second preimages for Tiger-12

4.2 Extension to Tiger-13

The attack can be extended to 13 rounds, by additionally guessing the value of X_{12} before the attack and verifying if the guess was correct afterwards. This again happens with a probability of 2^{-64}, yielding a total complexity of $2^{127.5}$. While it is theoretically possible to make an extension towards 14 rounds of Tiger, this hardly has an advantage over a simple exhaustive search.

5 First and Second Preimages for Tiger-12

The technique that has been developed in the previous section will now be applied to construct first and second preimages for Tiger-12. An extension of this construction to Tiger-13 is also possible.

5.1 Second Preimages for Tiger-12

Figure 2 shows how second preimages for Tiger-12 can be constructed, for (padded) messages with at least two message blocks and no padding bits in the first message block. This is equivalent to the requirement that the given message is at least 512 bits long.

In order to circumvent any issues that arise from the padding (which includes the message length) we choose the length of the preimage to be equal to that of the given message. We can hence reuse the last message block from the given message. All message blocks from the beginning up to the second to last message block can be chosen arbitrarily. This leaves us with exactly one message block, the central block in Fig. 2. Because the chaining values are known before and after this block, the attack from Sect. 4 can be applied. Of course a trivial generalisation where more than one message block is copied from the given message exists. In this case, the attack is applied to an earlier message block instead.

This procedure to find second preimages adds negligible overhead to the attack as described in Sect. 4. Hence, the time complexity remains at $2^{63.5}$ evaluations of the Tiger-12 compression function.

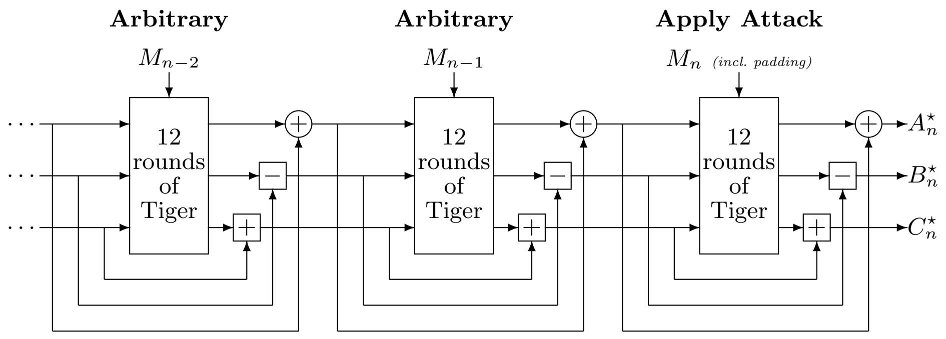

Fig. 3. Constructing first preimages for Tiger-12

5.2 First Preimages for Tiger-12

Finding first preimages is a bit more involved due to the fact that there is no given message which can be used to easily circumvent issues originating from the padding. To construct first preimages for Tiger-12, we proceed as follows.

First we choose the message length such that only a single bit of padding will be placed in X_6 of the last message block. This is equivalent to choosing a message length of $k \cdot 512 + 447$ bits, where k is a positive integer. Next, as shown in Fig. 3, all message blocks besides the last one are chosen arbitrarily and the attack is applied to this last block.

By choosing the message length in this way, X_7 of the last message block contains the message length as a 64-bit integer, which is fixed. Hence we can no longer choose X_7 freely during step 4 of the attack. By using the freedom in the choice of Y_7 in step 2 instead, the attack still works. Because step 3 is now also repeated, a larger part of the key schedule has to be redone on every attempt. The complexity figure of $2^{63.5}$ compression function evaluations can however be maintained because even with the larger part of the key schedule, the work of a single attempt does not exceed 70% — a fraction $2^{-0.5}$ — of a compression function evaluation. But additionally, we have to verify if the last bit of X_6 is a "1", as dictated by the padding rule. This happens with probability 2^{-1}, resulting in an overall complexity of $2^{64.5}$ compression function evaluations.

Note that the first preimages constructed in this way do not contain an integer number of bytes, which may not be acceptable. This problem can be solved by choosing the message length equal to $k \cdot 512 + 440$ bits instead. The only difference is that X_6 of the last message block now contains an entire byte of padding. The probability that this byte turns out to be correct after executing the attack is only 2^{-8}, and hence the overall complexity increases to $2^{71.5}$ compression function evaluations.

5.3 Extension to Tiger-13

Both attacks can be extended to Tiger-13, as explained in Sect. 4.2. The complexities become $2^{127.5}$ for second preimages, $2^{128.5}$ for first preimages and $2^{135.5}$ for first preimages of an integer number of bytes. A similar extension to Tiger-14 could be made, but as previously explained it does not give any advantage over an exhaustive search.

6 Conclusion

In this paper we have shown preimage attacks on reduced variants of the Tiger hash function. A method to find preimages for the compression function of Tiger-12 and Tiger-13 with a complexity of $2^{63.5}$ and $2^{127.5}$, respectively, was described. It was shown how to construct first and second preimages for these variants of Tiger based on this method. To the best of our knowledge, this is the first result with respect to preimages of the Tiger hash function.

Acknowledgements

We would like to thank Florian Mendel, Christian Rechberger, Hirotaka Yoshida for interesting discussions, and the anonymous reviewers for their helpful comments.

References

1. Ross Anderson and Eli Biham, *Tiger: A Fast New Hash Function*, Proceedings of Fast Software Encryption '96, Lecture Notes in Computer Science vol. 1039, pp. 89–97, Springer-Verlag, 1996.
2. Hans Dobbertin, *The First Two Rounds of MD4 are Not One-Way*, Proceedings of Fast Software Encryption '98, Lecture Notes in Computer Science vol. 1372, pp. 284–292, Springer-Verlag, 1998.
3. John Kelsey and Stefan Lucks, *Collisions and Near-Collisions for Reduced-Round Tiger*, Proceedings of Fast Software Encryption 2006, Lecture Notes in Computer Science vol. 4047, pp. 111–125, Springer-Verlag, 2006.
4. Florian Mendel, Bart Preneel, Vincent Rijmen, Hirotaka Yoshida and Dai Watanabe, *Update on Tiger*, Progress in Cryptology, Proceedings of INDOCRYPT 2006, Proceedings, Lecture Notes in Computer Science vol. 4329, pp. 63–79, Springer-Verlag, 2006.
5. Florian Mendel and Vincent Rijmen, *Cryptanalysis of the Tiger Hash Function*, Advances in Cryptology, Proceedings of ASIACRYPT 2007, Lecture Notes in Computer Science vol. 4833, pp. 536–550, Springer-Verlag, 2007.
6. Bart Preneel, *Cryptographic primitives for information authentication – state of the art*, In State of the Art and Evolution of Computer Security and Industrial Cryptography, 1997, Lecture Notes in Computer Science vol. 1528, pp. 50–105, Springer-Verlag, 1998.