Collisions and Near-Collisions for Reduced-Round Tiger

John Kelsey[1], Stefan Lucks[2]

[1] NIST, USA, john.kelsey@nist.gov
[2] University of Mannheim, Germany,
http://th.informatik.uni-mannheim.de/people/lucks/

Abstract. We describe a collision-finding attack on 16 rounds of the Tiger hash function requiring the time for about 2^{44} compression function invocations. This extends to a collision-finding attack on 17 rounds of the Tiger hash function in time of about 2^{49} compression function invocations. Another attack generates circular near-collisions, for 20 rounds of Tiger with work less than that of 2^{49} compression function invocations. Since Tiger has only 24 rounds, these attacks may raise some questions about the security of Tiger. In developing these attacks, we adapt the ideas of message modification attacks and neutral bits, developed in the analysis of MD4 family hashes, to a completely different hash function design.

Keywords: Tiger, hash function, collisions, attack

1 Introduction

In the past two years, a *flood* of cryptanalytic results [5–9, 2, 3] has washed away most of the practical hash functions used so far. Design-wise, all these hash functions (including MD5, RIPEMD, SHA0, and SHA1) descend from MD4. This has led to a growing interest into alternative hash function designs, which had been mostly overlooked by cryptanalysts so far. One such alternative construction is Tiger, designed by Anderson and Biham in 1996 [1]. Like the MD4-descendants, Tiger iterates an internal *compression function* for hashing arbitrarily long[1] messages. Tiger's compression function, however, is very different from the compression functions of the MD4 family.

Because the compression functions are so different internally, the attacks against the MD4 family would appear unlikely to be directly useful in attacking Tiger. Our analysis bears this out to some extent–the message modification techniques we use differ in important ways from those in [5–8]. However, we use message modification against Tiger for the same broad purpose as it is used in [7, 8]–to control the differences in the first few rounds by the choice of message values, despite having the message differences forced on us by our analysis of

[1] Tiger appears to restrict messages to 2^{64} bits maximum, based on the size of the message length field.

the message schedule. Further, the use of neutral parts of the message in [2] is directly applicable to our approach in attacking Tiger. In some sense, this is a hopeful sign; it implies that we may hope to take the attack techniques developed against the MD4 family, and apply them, in suitably altered form, to hash functions built on entirely different lines.

Below, we describe a collision-finding attack on Tiger reduced to 16 rounds. As the full Tiger operates on 24 rounds, this attack gets through two thirds of Tiger, with work equivalent to 2^{44} compression function invocations. Tiger produces a 192-bit hash, so a collision should ideally take 2^{96} such invocations.

We describe how to extend this attack to 17 rounds of Tiger, increasing the work to no more than 2^{49} compression function invocations.

Also, we describe an attack to choose two input chaining values with a small (namely, six bit) Hamming distance, which generates a near-colliding compression function outputs with the same Hamming distance – following, in fact, the same differential pattern as the input. In this sence, we describe our near-collisions as *circular*. (One could also describe them as pseudo-near-collisions.) This third attack gets through more than 80 % of Tiger (20 / 24 rounds), with work equivalent to less than 2^{49} compression function invocations. An ideal 192-bit hash should need approximately

$$\sqrt{2^{192} \Big/ \binom{192}{6}} \approx 2^{80} \quad \text{compression function invocations}$$

for near-collisions with six bits of Hamming distance, instead of $< 2^{49}$.

The remainder of this paper is organized as follows. Section 2 provides a description of Tiger, in sufficient detail to follow our attacks. Section 3 provides an overview over the collision attack and describes some of the details. Sections 4 and 5 deal with the core of the attack: the message modification technique. Section 6 introduces techniques to extend our attack to more than 16 rounds. These are demonstrated by a collision attack against 17 rounds of Tiger, and by a circular near-collision attack against 20 rounds. Section 7 briefly discusses the security of Tiger, and outlines some lessons learned from the attack.

2 High-Level Description of Tiger

Tiger's compression function is based on applying an internal "block cipher like" function, which takes a 192-bit "plaintext" and a 512-bit key to compute a 192-bit "ciphertext". The "block cipher like" function is applied according to the Davies-Meyer construction: a 512-bit message block is used as a key to encrypt the 192-bit chaining value, and then the input chaining value is fed forward to make the whole function non-invertible. In the remainder of this section, we will describe Tiger in sufficient detail to follow the course of our attack. Note that if, for any given input chaining value, we can generate two different messages yielding the same output chaining value, then we have found a collision for Tiger.

Tiger was designed with 64-bit architectures in mind. Accordingly, we will denote a 64-bit unsigned integer as a "word". We will represent a word as a hexadecimal number. Tiger uses arithmetic operations (addition, subtraction and multiplication by small constants), bit-wise XOR, NOT, logical shift operations and S-Box applications. The arithmetic operations over words are modulo 2^{64}. The chaining value is represented internally as three 64-bit words, the message block as eight 64-bit words.

Thus, three words A, B, C describing the input chaining value and eight message words $X_0, \ldots, X_7$ are fed into the compression function, which generates three words A', B', C' describing the output chaining value. The compression function's final output A'', B'', C'' is generated by the feedforward function

$$\begin{aligned} A'' &:= A \oplus A', \\ B'' &:= B - B', \text{ and} \\ C'' &:= C + C'. \end{aligned}$$

2.1 The Tiger Round Function

In the terminology of [4], Tiger's block cipher like function is a "target-heavy unbalanced Feistel cipher". The block is broken into three words, labeled A, B, and C. Each round, a message word X is XORed into C:

$$C := C \oplus X.$$

Then A and B are modified:

$$\begin{aligned} A &:= A - \mathbf{even}(C), \\ B &:= B + \mathbf{odd}(C), \\ B &:= B \times (\text{const}), \end{aligned}$$

with a round-dependent constant (const) $\in \{5, 7, 9\}$. The results are then shifted around, so that A,B,C becomes B,C,A. See Figure 1.

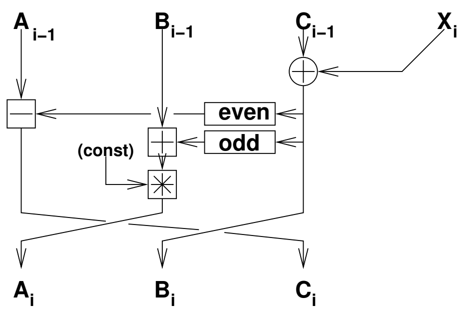

Fig. 1. The round function of Tiger

For the definition of **even** and **odd**, consider the word C being split into eight bytes $C[0]$, ..., $C[7]$, with the most significant byte $C[0]$. The functions

even and **odd** employ four S-Boxes $T_1, \ldots, T_4 : \{0,1\}^8 \to \{0,1\}^{64}$ as follows:

$$\mathbf{even}(C) := T_1(C[0]) \oplus T_2(C[2]) \oplus T_3(C[4]) \oplus T_4(C[6]) \quad \text{and}$$
$$\mathbf{odd}(C) := T_1(C[7]) \oplus T_2(C[5]) \oplus T_3(C[3]) \oplus T_4(C[1]).$$

The "even bytes" and the "odd bytes" of a word W are defined as

$$W[\mathbf{even}] = (W[0], W[2], W[4], W[6]) \in (\{0,1\}^8)^4 \quad \text{and}$$
$$W[\mathbf{odd}] = (W[7], W[5], W[3], W[1]) \in (\{0,1\}^8)^4.$$

The round function spreads changes around very quickly – a one-bit difference introduced into C in the first round will change about half the bits of the block by the end of the third round. Tiger seems to be much better at this than the members of the MD4 family.

It is easy to produce local collisions for the Tiger round function, using some pattern $(\alpha, \beta, 0, \alpha')$. Here, α is an input difference to the even bytes of the S-boxes, β is an XOR difference which is expected to cancel out the result of that difference on the even function, and α' is α multiplied by (const), being expected to cancel out the original introduced change of α. However, local collisions of this form are surprisingly hard to use in attacks on more than eight rounds of Tiger – the key schedule seems to be quite effective at destroying such patterns.

2.2 The Key Schedule

Tiger consists of 24 rounds. Each round uses one message word X_i as its round key. The first eight round keys $X_0, \ldots, X_7$ are identical to the 512-bit cipher key (or rather, to the 512-bit message block). The remaining 16 round keys are generated by applying the key schedule function:

$$(X_8, \ldots, X_{15}) := \text{KeySchedule}(X_0, \ldots, X_7)$$
$$(X_{16}, \ldots, X_{23}) := \text{KeySchedule}(X_8, \ldots, X_{15})$$

The key schedule uses logical shifts on words, denoted by $\ll$ and $\gg$, e.g.,

- $\texttt{1111 5555 9999 FFFF} \ll 5 = \texttt{222A AAB3 333F FFE0}$, and
- $\texttt{222A AAB3 333F FFE0} \gg 9 = \texttt{0011 1555 5999 9FFF}$.

Further, it uses the bit-wise NOT function, e.g. for $X = \texttt{EEEE AAAA 6666 0000}$, the negation of X is $\overline{X} = \texttt{1111 5555 9999 FFFF}$. The key schedule modifies its input $(x_0, \ldots, x_7)$ in two passes:

first pass	second pass
1. $x_0 := x_0 - (x_7 \oplus \text{Const}_1)$	9. $x_0 := x_0 + x_7$
2. $x_1 := x_1 \oplus x_0$	10. $x_1 := x_1 - (x_0 \oplus (\overline{x_7} \ll 19))$
3. $x_2 := x_2 + x_1$	11. $x_2 := x_2 \oplus x_1$
4. $x_3 := x_3 - (x_2 \oplus (\overline{x_1} \ll 19))$	12. $x_3 := x_3 + x_2$
5. $x_4 := x_4 \oplus x_3$	13. $x_4 := x_4 - (x_3 \oplus \overline{x_2} \gg 23))$
6. $x_5 := x_5 + x_4$	14. $x_5 := x_5 \oplus x_4$
7. $x_6 := x_6 - (x_5 \oplus (\overline{x_4} \gg 23))$	15. $x_6 := x_6 + x_5$
8. $x_7 := x_7 \oplus x_6$	16. $x_7 := x_7 - (x_6 \oplus \text{Const}_2)$

The final values $(x_0, \ldots, x_7)$ are used as the key schedule output. The constants are $\mathrm{Const}_1 = \mathrm{A5A5} \ldots \mathrm{A5A5}$ and $\mathrm{Const}_2 = 0123 \ldots \mathrm{CDEF}$.

3 The Attack

We propose a differential attack on Tiger in three parts. Throughout the attack, we are switching between XOR-differences and additive differences. In general, switching between differences holds with some nonzero probability; for example, an additive difference of 1 can be represented as an XOR difference of 1, with probability 1/2 of being correct.

3.1 Conventions

Transforming one type of difference into another is typically probabilistic, but for some values, it has probability one.

- If $X - Y = 2^i$, then $\Pr[X \oplus Y = 2^i] = 1/2$. The exception is $i = 63$, where $\Pr[X \oplus Y = 2^i] = 1$.
- Let $I := 2^{63}$. Switching between the additive difference I and the XOR-difference I succeeds with probability 1. In other words, when dealing with a difference I, we need not care what type of difference this actually is. Our attack will make extensive use of this simple fact.
- Note that a difference I in a word W remains the same, even if W is multiplied by some odd constant (const), as done in the Tiger compression function.

We start counting rounds by 0, and we write X_i for the message word input of the i-th round, and A_i, B_i, C_i for the output of round i – which just happens to be the input chaining values for round $i+1$. Accordingly, the chaining value input for the round 0 (the first round) is A_{-1}, B_{-1}, C_{-1}.

The differences in message words are most usefully seen as XOR-differences, since the message word (or the "round key") X_i is XORed into the state. Additive differences are what we need to know when dealing with the two target words in the round (the two words that get altered), because the arithmetic differences are all mod 2^{64}. For the S-box inputs in the message modification step, XOR differences are most useful.

We will use the following notation for the differences which occur in some word W:

- $\Delta^{+}(W) = W - W^{*} \bmod 2^{64}$ for additive differences and
- $\Delta^{\oplus}(W) = W \oplus W'$ for word-wise differences.

3.2 Outline of the Attack

The attack can be broken into three pieces:

1. Differential characteristic $(I, I, I, I, 0, 0, 0, 0) \rightarrow (I, I, 0, 0, 0, 0, 0, 0)$ in the key schedule.
2. Differential characteristic $(I, I, 0) \rightarrow (0, 0, 0)$ in rounds 6-9 of the round function. (Because the message words in rounds 10-15 are unchanged, this leads to a collision after 16 rounds.)
3. Message modification to force the difference in the round function after round 6 to $(I, I, 0)$.

3.3 Key Schedule Differences

Consider a difference of the form $(I, I, I, I, 0, 0, 0, 0)$ in the message words. The first pass of the key schedule turns this into an intermediate difference pattern $(I, 0, I, 0, 0, 0, 0, 0)$. The second pass turns this into $(I, I, 0, 0, 0, 0, 0, 0)$. This is the differential pattern we will use for our attack; it holds with probability one, and covers the expanded message words used for rounds 0-15.

The colliding messages will differ only in the high order bits of their first four words. The expanded message words will differ for rounds 8-9, only in their high order bits. Expanded message words 10-15 will have no differences. This means that if the states of the compression functions processing the two messages are equal after round 9, they will remain equal until the end of round 15, yielding a 16-round collision.

3.4 Round Function Differences

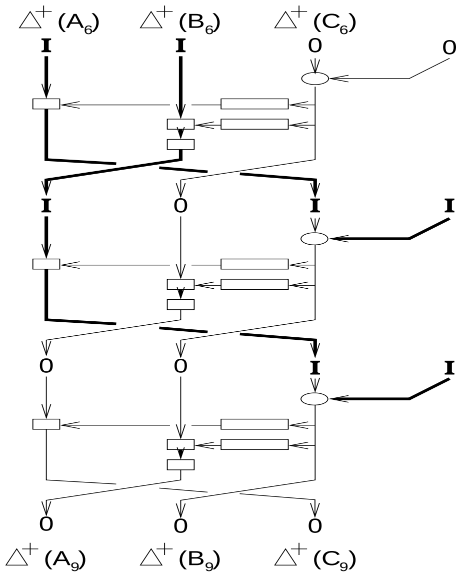

Fig. 2. Probability one characteristic from round 6-9.

Given the key schedule differential characteristic above, we can specify a differential characteristic for the round function from the end of round 6 to the end of round 9, going from $(I, I, 0) \rightarrow (0, 0, 0)$ by canceling with the differences in rounds 8-9. The expanded message words from rounds 10-15 have no differences, and thus a collision after round 9 becomes a collision for 16 rounds of Tiger. Figure 2 shows this characteristic.

3.5 Message Modification

The main difficulty of the attack is in the message modification step. Recall that our target difference at the end of round 6 is

$$\Delta^+(A_6) = I,\ \ \Delta^+(B_6) = I, \Delta^+(C_6) = 0.$$

Independently from the choice of message words, we know $\Delta^+(C_5)$ and $\Delta^+(C_4)$. Also, since $\Delta^+(X_6) = 0$, we need $\Delta^+(C_5) = \Delta^+(B_6) = I$. Similarly, we know the relationship $\Delta^+(C_4) = I + \Delta^+(\mathbf{odd}(B_6))$.

4 Local Message Modification by Meeting in the Middle

Assume we know inputs $(A_{i-1}, B_{i-1}, C_{i-1})$ and $(A^*_{i-1}, B^*_{i-1}, C^*_{i-1})$, and some XOR differences in the message words X_i and X_{i+1}. We want to force some additive difference $\Delta^+(C_{i+1})$ to $\delta^* = C_{i+1} - C^*_{i+1}$. As depicted in Figure 3, the difference $\Delta^+(C_{i+1})$ depends on $\Delta^+(B_{i-1})$, the additive output difference of the **odd** function from round i, and the additive output difference of the **even** function from round $i + 1$.

4.1 Plain Message Modifications

First consider the **even** function, which, after computing $B_{i+1} := C_i \oplus X_{i+1}$, evaluates as

$$\mathbf{even}(B_{i+1}) := T_1(B_{i+1}[0]) \oplus T_2(B_{i+1}[2]) \oplus T_3(B_{i+1}[4]) \oplus T_4(B_{i+1}[6]).$$

For any nonzero XOR difference between words B_{i+1} and B^*_{i+1}, we expect about 2^{32} *different* additive output differences of the form $\delta_{\mathbf{even}} = \mathbf{even}(B_{i+1}) - \mathbf{even}(B^*_{i+1})$. Similarly, when we consider the **odd** function

$$\mathbf{odd}(B_i) := T_1(B_i[7]) \oplus T_2(B_i[5]) \oplus T_3(B_i[3]) \oplus T_4(B_i[1]),$$

we expect close to 2^{32} *different* additive output differences of the form $\delta_{\mathbf{odd}} = \mathbf{odd}(B_i) - \mathbf{odd}(B^*_i)$.

Thus, if the differences in $B_{i+1}[\mathbf{even}]$ and in $B_i[\mathbf{odd}]$ both are nonzero, we can apply a meet-in-the-middle (MITM) approach to force

$$(\Delta^+(B_{i-1}) + \delta_{\mathbf{odd}}) \times \text{const} - \delta_{\mathbf{even}} = \delta^*$$

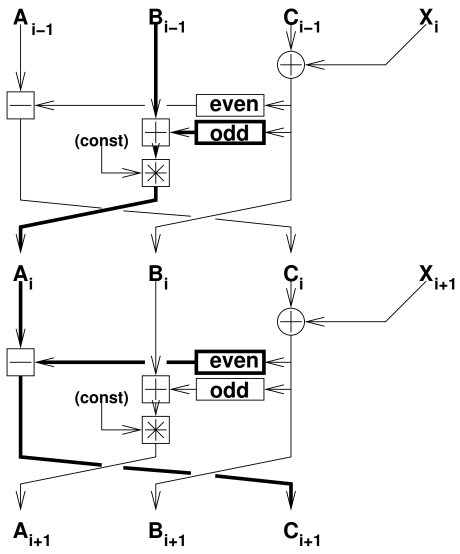

Fig. 3. The information flow from B_{i-1} to C_{i+1}.

- Store the 2^{32} candidates for $\delta_{\mathbf{odd}}$ in a table.
- For all 2^{32} candidates for $\delta_{\mathbf{even}}$, test if $\delta_{\mathbf{odd}}$ exists with

$$\delta_{\mathbf{even}} = (\Delta^{+}(B_{i-1}) + \delta_{\mathbf{odd}}) \times (\text{const}) - \delta^{*},$$

  or rather

$$\delta_{\mathbf{odd}} = (\delta_{\mathbf{even}} + \delta^{*})/(\text{const}) - \Delta^{+}(B_{i-1}) \qquad (1)$$

  (note that since (const) is odd, division by (const) mod 2^{64} is well-defined).

This technique takes some 2^{32} evaluations of each of the functions **even** and **odd**, which is equivalent to about 2^{28} evaluations of the compression function – and, of course, some 2^{32} units of storage space.

We estimate that for given $\Delta^{+}(B_{i-1})$ and δ^{*}, the meet-in-the-middle approach succeeds with a probability close to 1/2. In the attack scenario, we will repeat the approach with another $\Delta^{+}(B_{i-1})$ or another target difference δ^{*}, if necessary.

Assume $X_i[\mathbf{even}]$ has been fixed and the MITM delivered $\delta_{\mathbf{even}}$ and $\delta_{\mathbf{odd}}$ satisfying Equation 1. We can now determine value for $B_{i+1}[\mathbf{even}]$ and $B_i[\mathbf{odd}]$ which will produce the desired differences, and thus will map the input difference to the output difference as required. During the MITM step, each candidate additive difference for $\delta_{\mathbf{even}}$ is produced by one or more specific values of $B_{i+1}[\mathbf{even}]$, and likewise, each value of δ_{odd} is produced by one or more specific values of $B_i[\mathbf{odd}]$.

Finally, we are able to compute 64 local message bits:

$$X_i[\mathbf{odd}] := C_{i-1}[\mathbf{odd}] \oplus B_i[\mathbf{odd}] \quad \text{and}$$
$$X_{i+1}[\mathbf{even}] := C_i[\mathbf{even}] \oplus B_{i+1}[\mathbf{even}].$$

Note that C_i has been defined by fixing $X_i[\mathbf{even}]$.

In the attacks below, we use two variations on these ideas.

4.2 Message Modification to get an XOR Difference

In step 3 of the attack below, we need a specific XOR difference in C_3. However, the meet-in-the-middle technique above takes an additive difference, not an XOR difference, as input. Our solution to this is to throw brute force computation at the problem: For a desired XOR difference of Hamming weight k, we simply go through the meet-in-the-middle search for each additive difference which could be produced by the XOR difference, until we run out of choices or find an additive difference which both matches and yields the desired XOR difference when we compute it forward.

An additive difference which can lead to a given k-bit XOR difference has about a 2^{-k} probability of doing so[2] This means we expect to need to try about 2^k additive differences which are consistent with the k-bit XOR difference before we succeed in finding a match. Since each MITM step succeeds in finding a matching additive difference about half the time, we will need to do a total of 2^{k+1} MITM steps. However, we can optimize this in a simple way, by only redoing one side of the MITM search for each new targeted additive difference. The expected work is thus bounded by 2^{28+k}.

4.3 Message Modification with Constraints

Two of the MITM steps (steps 4 and 5) in the attack below must live with constraints on the selection of message bits. The constraints come from the transition between an XOR difference in C_3 and an additive difference in B_4. Since the XOR difference has k bits active, and the additive difference is consistent with only one set of values for those bits, k bits of message word X_4 are constrained[3].

Constrained message modification is relatively simple: Instead of searching over 2^{32} possible additive differences from each side, we search over a smaller number, with the constrained bits of the message fixed to their required values. For the sake of simplicity, we assume that $k/2$ bits are constrained in the even bytes, and $k/2$ in the odd bytes. However, the probability of success is decreased in a corresponding way; with only 2^{28} choices from one side, and 2^{32} from the other, we expect about a 2^{-4} probability of a match. Thus, we expect to have to repeat an MITM search with 4 constrained bits about 16 times.

[2] A k-bit XOR difference has either 2^k or 2^{k-1} additive differences consistent with it. For a flipped bit in position j, this represents the choice of whether to add or subtract 2^j. A flipped high order bit always matches both $+2^{63}$ and -2^{63} in mod 2^{64} arithmetic.

[3] For example, an XOR difference of 1 is consistent with an additive difference of either -1 or +1. If the low bit in C_3 is 0, the low bit in C_3^* will be 1, and reaching an additive difference of -1 will require fixing the low bit of X_4 to 1.

5 The Global Message Modification Scenario

0. Do a one-time precomputation to find a additive difference L with a low Hamming weight corresponding XOR difference which we can cancel out by our choice of the even bytes of X_6. (Note that the specific value of X_6 is not determined yet; we are simply ensuring that this additive difference will permit a choice of X_6 that will cancel the resulting difference out.) This costs 2^{27} Tiger-16 hash function equivalents, and we expect it to yield an additive difference which is consistent with an 8-bit XOR difference[4]

1. Choose X_0 and $X_1[\mathbf{even}]$ to ensure that C_0 and C_1 have useful (that is, nonzero in both the even and odd bytes) differences. Note that at the end of this step, we know $\Delta^{\oplus}(C_1)$ and $\Delta^{\oplus}(C_2)$. We use these in the next step. The work here is negligible.

2. Choose $X_1[\mathbf{odd}]$ and $X_2[\mathbf{even}]$ to ensure that C_2 has a useful difference. Note that at the end of this step, we know $\Delta^{\oplus}(C_2)$. We use this XOR difference in the next step. The work here is negligible.

3. Do a message modification step to get XOR difference $\Delta^{\oplus}$ in C_3 which is consistent with the additive difference L. This is described above. The expected work here is about 2^{36} Tiger-16 hash equivalents, and we determine $X_2[\mathbf{odd}], X_3[\mathbf{even}]$.

4. Do a constrained meet in the middle step, choosing $X_3[\mathbf{odd}], X_4[\mathbf{even}]$ to get $\Delta C_4 = I$. We expect there to be four constrained bits, meaning that we expect to have to try this 16 times before we get a match. Each failed attempt requires that we go back to step 2. We thus expect to spend about $2^4 2^{36} = 2^{40}$ Tiger-16 equivalents completing this step of the attack.

5. Do a constrained meet in the middle step, choosing $X_4[\mathbf{odd}], X_5[\mathbf{even}]$ to force $\Delta C_5 = I$. As before, we expect this to be constrained by four bits, and thus to need to be repeated 16 times. Each failure requires that we go back to step 2. This step thus is expected to be completed after doing about $2^4 2^{40} = 2^{44}$ Tiger-16 hash equivalents of work.

6. Given the value of C_5, we use the results of step 0's search to determine the value for the even bytes of X_6. This is negligible work, and never fails.

The result of this is an additive difference in the output of of $I, 0, I$ in the output of round 7. With probability one, this cancels out with the key schedule characteristic, leading to a 16-round collision.

5.1 Neutral Bits

The above attack has specified message words $X_{0,1,2,3,4}$ and the even bytes of message words $X_{5,6}$. This leaves an enormous number of bits of the message

[4] We expect this because in a set of 2^{32} random 64-bit integers, we expect about one with a Hamming weight of 8, since 64-bit integers with Hamming weight 8 make up about 2^{-32} of all 64-bit integers. In this case, a 9-bit XOR difference where one of the active bits is the high-order bit gives identical results in the remainder of the attack.

which can be varied without interfering with the 16-round collision. After having found the collision, we may freely determine the values for the odd bytes of $X_{5,6}$ and all of X_7. The above attack thus finds 2^{128} 16-round collisions for Tiger.

For example, consider varying the bytes of $X_5^{[}$**odd**]. This alters the output of the **odd** function in round 5, and thus the value of A_6. However, since there is no difference active in the odd bytes of B_5, changing the input to the odd function in round 5 adds the same change to A_6 and A_6^*. This leaves the additive difference in A_6 unchanged, which means that the same difference in the **even** function in round 6 will cancel it out. Similarly, a change to the odd bytes of X_5 changes the value of B_5, but doesn't change the additive difference $B_5^* - B_5$, as it adds the same amount to both. The same kind of analysis applies to all the neutral bits.

5.2 Free Bits

The attack also imposes almost no constraints on $X_{0,1}$ or the even bytes of X_2. We need control of about 12 of those bits during the attack. A natural thing to do is to choose $X_{0,1}$ freely at the beginning of the attack in any way that is convenient, and then use the even bytes of X_2 to provide multiple trials for the message modification steps.

6 Going beyond 16 Rounds

In this section, we will apply the 16-round collision finding technique from above as some subroutine, to attack more rounds of Tiger. Make the following two assumptions

1. The round keys $X_8, \ldots, X_{15}$ observe the characteristic $(I, I, I, I, 0, 0, 0, 0)$.
2. The input difference $(\Delta^+(A_7), \Delta^+(B_7), \Delta^+(C_7))$ to round eight is $(0, 0, 0)$.

If both assumptions hold, we can apply the 16-round technique from above to compute an "intermediate-message" $(X_8, \ldots, X_{15})$, such that we get a collision after round 23. As the key schedule is invertible, the "real message" $(X_0, \ldots, X_7)$ can easily be computed by by running the key schedule backwards.

6.1 A Round Key Differential

Set $I^* := I \gg 23 = 2^{40}$ and $I^{**} := I'^* \gg 23 = 2^{17}$. Assume that the eight message words observe the differential characteristic $(I, I, 0, 0, I + I^*, I + I^*, I^* + I^{**}, 0)$. With the probability $\approx 1/16$, we expect the first pass through the message schedule turns this characteristic into $(I, 0, 0, 0, I + I^*, 0, 0, 0)$. If this happens, then we expect the second pass to produce our target characteristic $(I, I, I, I, 0, 0, 0, 0)$ with probability $1/2$.

To summarise, we expect the differential characteristic

$$(I, I, 0, 0, I + I^*, I + I^*, I^* + I^{**}, 0) \;\longrightarrow\; (I, I, I, I, 0, 0, 0, 0) \tag{2}$$

to hold with a probability of about 1/32. We have verified this experimentically. (Actually, our results seem to indicate a slightly better probability of approximately 1/28. But for simplicity, we use $1/32 = 2^{-5}$ for our analysis.)

6.2 Attacking 17 rounds of Tiger

Now we describe an attack on 17 rounds of Tiger, namely rounds 7 to 23:

1. Given the initial value (A_6, B_6, C_6), choose the message word X_7 and apply one round of Tiger to get (A_7, B_7, C_7).
2. Apply the 16-round attack to get a message $(X_8, \ldots, X_{14}, X_{15})$ colliding with $(X_8, \ldots, X_{14}, X_{15}) + (I, I, I, I, 0, 0, 0, 0)$.
   Recall that $(X_8, \ldots, X_{14}, X_{15})$ contains 128 neutral bits, including all the 64 bits of X_{15}. I.e., any choice of X_{15} will produce a collision.
3. Now run the key schedule backwards to get $(X_0, \ldots, X_7)$. As X_7 has already been chosen, we have to observe a little twist here: Given $X_7, \ldots, X_{14}$, but ignoring X_{15}, we compute the remaining seven message words $X_0, \ldots, X_6$ as explained below.
4. Now we check our differential characteristic (2). If it holds, we have found a 17-round collision for rounds 7 to 23 and are done.
   Else, we go back to the first step.

On the average, the characteristic (2) holds at least one in $32 = 2^5$ times. Thus, the attack on 17 rounds of Tiger takes the time of about

$$2^5 * 2^{44} = 2^{49}$$

compression function invocations.

What about step 3 of the attack, i.e., running the key schedule backwards? Given X_7 and $X_8, \ldots, X_{14}$, we have to compute $X_0, \ldots, X_6$. We write $Y_0, \ldots, Y_7$ for the output of the first key schedule pass when computing

$$(X_8, \ldots, X_{14}, X_{15}^*) = \text{KeySchedule}(X_0, \ldots, X_7).$$

(Note that we actually know $X_8, \ldots, X_{14}$, while X_{15}^* is unknown.)

Inverting steps 11–15 of the second pass is straightforward:

$$\begin{aligned}
Y_6 &:= X_{14} - X_{13} \\
Y_5 &:= X_{13} \oplus X_{12} \\
Y_4 &:= X_{12} + (X_{11} \oplus (\overline{X_{10}} \gg 23)) \\
Y_3 &:= X_{11} - X_{10} \\
Y_2 &:= X_{10} \oplus X_9
\end{aligned}$$

We get the value Y_7 by inverting step 8 of the first pass:

$$Y_7 := X_7 \oplus Y_6$$

Finally, we can invert those steps of second pass which depend on Y_7:

$$Y_6 := X_9 + (X_8 \oplus (\overline{Y_7} \ll 19))$$
$$Y_0 := X_8 - Y_7$$

Inverting steps 1 to 7 of the first pass is quite similar.

6.3 Circular Near-Collisions for 20 rounds of Tiger

Now we go even further, to 20 rounds of Tiger, at the cost of dealing with a weaker attack model. Instead of a collision-attack, we provide circular near-collisions with small Hamming weight (Hamming weight 6). We attack rounds 4 to 23 (i.e., all but the first four rounds). Hence, we denote the input chaining values by A_3, B_3, C_3. The attack works as follows:

1. Arbitrarily choose the chaining values A_7, B_7, C_7 for round 8.
2. Employ the 16-round attack, to find message words $X_8, \ldots, X_{15}$ such that the output after round 23 collides.
3. Run the key schedule backwards, to compute the "real" message words $X_0, \ldots, X_7$.
   If the characteristic (2) does not hold, go back to step 2.
4. Run the rounds 7, 6, 5, and 4 backwards to compute the initial values A_3, B_3, C_3. The differences in the message words induce the same differences in the initial values, namely
   $$\Delta^{\oplus}(A_3) = I + I^* = \Delta^{\oplus}(B_3) \text{ and } \Delta^{\oplus}(C_3) = I^* + I^{**}.$$
5. The feedforward destroys the collision, of course. But with very high probability, it leaves us with a low Hamming weight near-collision. With probability 2^{-3} the feedforward output follows the same differential pattern than the input chaining values:
   $$\Delta^{\oplus}(A_{23}) = I + I^* = \Delta^{\oplus}(B_{23}) \text{ and } \Delta^{\oplus}(C_{23}) = I^* + I^{**}.$$
   If it doesn't follow this pattern, then randomly vary the neutral bits in $X_{13,14,15}$ until it does hold. We expect to need to try about $2^3 = 8$ sets of neutral bits for this.

Similarly to Section 6.2, we expect to iterate the 16-round attack no more than 2^5 times, on the average. In total, we expect a running time of about 2^{49} Tiger-20 equivalents. Varying the neutral bits in the last step adds negligible cost. Thus, the Tiger-20 near-collision attack costs less work than iterating Tiger-20 2^{40} times.

7 Conclusions and Open Questions

In this paper, we have developed collision attacks:

- an attack against 16 rounds of Tiger, requiring work equivalent to about 2^{44} compression function computations and
- an attack gainst 17 rounds of Tiger, being no more than 32 times slower – work equivalent to about 2^{44} compression function computations.

These attacks heavily use message modification techniques.

We have further exploited this technique for a near-collision attack (with adversarially chosen input chaining values) against the last 20 rounds of Tiger, also for about 2^{49} compression function computations worth of work. These near-collisions are circular, i.e., the input and the output chaining values have identical differences (with a Hamming weight of 6).

7.1 The Security of Tiger

All of our results are based on message modification techniques, which mean that we choose both the XOR differences in the message, and also specific values for most or all of the message bits. This constrains the attack in many ways. For example, we can see no way to adapt our current techniques to collisions against an application using 16-round Tiger in the HMAC construction–our lack of knowledge of the chaining values would make our approach impossible.

Second-preimage attacks on a single compression function computation also appear to be very difficult using our techniques. Both the difference between the colliding message blocks and the specific values of the messages are constrained by our attack; it appears to be very difficult to "work backward" from a specified message block with some hash output to a colliding message block. Second preimages are trivial to find for up to 8 rounds, and appear possible to find for up to 11 rounds using local collisions, but we have not investigated this line of attack in much detail yet.

We are more concerned with the possibility of extending the collision attack to more rounds. As Tiger has only 24 rounds, attaking 16–20 rounds is threatening. A relatively small improvement might make the attack techniques applicable to the full hash function. We definitely do not believe that the attack techniques presented here have been fully exploited in the current attack.

We point out that pseudo-collisions and near-collisions can be more than just certificational weaknesses. Some of the attacks against ciphers from the MD4 family employ pseudo- and near-collisions in attack scenarios with more than one message block, to find plain collisions for the hash function itself (see, e.g., [6]).

7.2 What We've Learned About Tiger

We draw two broad lessons from the analysis so far. First, we believe that Tiger has too few rounds. Message modification techniques allow us to almost completely control what happens in the first third of the hash function at present, allowing us to place differences in the remaining rounds almost without constraint. Second, the use of large S-boxes and mixing between addition and XOR operations is an excellent strategy for building a block cipher, but it works very differently inside a hash function. Large S-boxes tend to have a large set of equally good differentials, but which differential will pass the next round depends on the value of the internal state of the hash function; the attacker facing a block cipher with such large S-boxes must guess which differential to try; the attacker facing a hash function can often choose those values to make his differential work, or at least look inside the state of the hash function to determine the best differential path to try.

7.3 Applicability of the Tools of the MD4 Family Attacks

We have also seen some overlap in the tools used to attack the MD4 family, and our results on reduced-round Tiger. Broadly, we analyze the message expansion for the hash function, and form a differential characteristic which, if entered after round 7, will lead to a collision in the full hash function. We then use message modification to force the hash states processing a pair of messages with our desired difference onto this differential characteristic after round 7. This is quite similar to the techniques used in [7] and [8], though without (yet) the use of advanced message modification techniques. Similarly, a variation on the neutral bit techniques of [2] are used to make our 20-round pseudo-near-collision attack more efficient. While the details of using these attack tools are different for Tiger, the high level similarities in approach suggest that we may be learning generally useful attack techniques against hash functions from the recent results on the MD4 family of hash functions.

8 Acknowledgements

The authors wish to thank Eli Biham, Lily Chen, Orr Dunkelman, Morris Dworkin, Matt Fanto, and the anonymous referees for useful comments and discussions.

References

1. Anderson, R., Biham, E., Tiger: A Fast New Hash Function, Fast Software Encryption, FSE'96, LNCS 1039, 1996.
2. E. Biham, R. Chen. Near-Collisions of SHA-0. Crypto 04, LNCS 3152.
3. E. Biham, R. Chen, A. Joux, P. Carribault, C. Lemuet, W. Jalby. Collisions of SHA-0 and reduced SHA-1. Eurocrypt 2005, LNCS 3494, 36–57.
4. B. Schneier, J. Kelsey. Unbalanced Feistel Networks and Block Cipher Design. FSE 1996, LNCS, 121–144.
5. X. Wang, X. Lai, D. Feng, H. Cheng, X. Yu. Cryptanalyisis of the hash functions MD4 and RIPEMD. Eurocrypt 2005, LNCS 3494, 1–18.
6. X. Wang, H. Yu. How to break MD5 and other hash functions. Eurocrypt 2005, LNCS 3494, 19–35.

7. X. Wang, H. Yu, Y. L. Yin. Efficient collision search attacks on SHA0. Crypto 2005.
8. X. Wang, Y. L. Yin, H. Yu. Finding collisions in the full SHA1. Crypto 2005.
9. X. Wang, A. Yao, F. Yao. New Collision Search for SHA-1. Presentation at rump session of Crypto 2005 (communicated by A. Shamir).