Cryptanalysis of Twister

Florian Mendel, Christian Rechberger, and Martin Schläffer

Institute for Applied Information Processing and Communications (IAIK)
Graz University of Technology, Inffeldgasse 16a, A-8010 Graz, Austria.

{florian.mendel,christian.rechberger,martin.schlaeffer}@iaik.tugraz.at

Abstract. In this paper, we present a pseudo-collision attack on the compression function of all Twister variants (224,256,384,512) with complexity of about $2^{26.5}$ compression function evaluations. Furthermore, we show how the compression function attack can be extended to construct collisions for Twister-512 with a time complexity starting from about $2^{231.4}$ and different time-memory trade-offs.

1 Description of Twister

The hash function Twister is an iterated hash function based on the Merkle-Damgård design principle. It processes message blocks of 512 bits and produces a hash value of 224, 256, 384, or 512 bits. If the message length is not a multiple of 512, an unambiguous padding method is applied. For the description of the padding method we refer to [1]. Let $m = m_1 \| m_2 \| \cdots \| m_t$ be a t-block message (after padding). The hash value $h = H(m)$ is computed as follows:

$$\begin{aligned} H_0 &= IV \\ H_i &= f(H_{i-1}, M_i) \quad \text{for } 0 < i \leq t \\ H_{t+1} &= f(H_t, C) = h \ , \end{aligned}$$

where IV is a predefined initial value and C is the value of the checksum. It is computed from the intermediate values of the internal state after each Mini-Round. Note that while for Twister-224/256 the checksum is optional it is mandatory for Twister-384/512. The compression function of Twister basically consists of 3 Maxi-Rounds. Each Maxi-Rounds consist of 3 or 4 Mini-Rounds (depending on the output size of Twister) and is followed by a feed-forward XOR-operation.

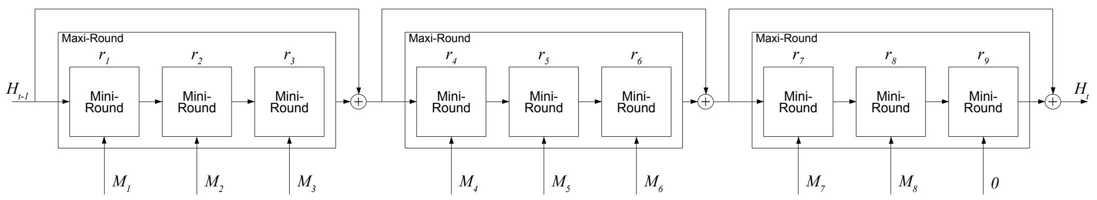

Fig. 1. The compression function of Twister-224/256.

The Mini-Round of Twister is very similar to the Advanced Encryption Standard (AES) [4]. It updates an 8×8 state S of 64 bytes as follows:

MessageInjection A 8-byte message block M is inserted (via XOR) into the last row of the 8×8 state S.

AddTwistCounter A 8-byte block counter is xored to the second column of the sate S.

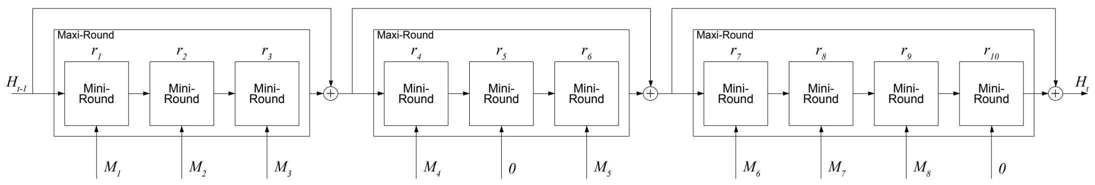

Fig. 2. The compression function of Twister-384/512.

SubBytes is identical to the SubBytes operation of AES. It applies an S-Box to each byte of the state independently

ShiftRows is a cyclic left shift similar to the ShiftRows operation of AES. It rotates row j by $(j-1) \pmod 8$ bytes to the left.

MixColumns is similar to the MixColumns operation of AES. It applies a 8×8-MDS matrix A to each column of the state S.

After the last message block and /or the checksum has been processed, the final hash value is generated from the last chaining value by an output transformation. For a detailed description of Twister we refer to [1].

2 Pseudo-collision for the compression function

In this section, we present a pseudo-collision attack on the compression function of Twister for all output sizes. The attack has a complexity of about $2^{26.5}$ compression function evaluations. In the attack we use the characteristic of Figure 3 for the first Maxi-Round (3 Mini-Rounds) of Twister. The 3 Mini-Rounds are denoted by r_1, r_2 and r_3 and the state after the Mini-Round r_i is denoted by S_i. The initial state or chaining value is denoted by S_0. In the attack we add a difference in message word M_1 (8 active bytes) to the state S_0, which results in a full active state S_1 after the first Mini-Round r_1. After the MixColumns transformation of the second Mini-Round r_2, the differences result in 8 active bytes of the last row of state S_2, which can be canceled by the message word M_3 in the third Mini-Round r_3.

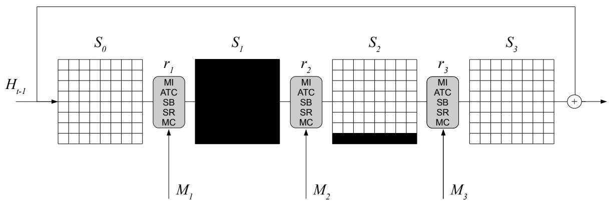

Fig. 3. Characteristic to construct a pseudo-collision in the first Maxi-Round.

The message differences and values for the state are found using a meet-in-the-middle approach and Figure 4 shows the characteristic in detail. We start with message word differences in M_1 and M_3 at states S_1' and S_2. The differences can be propagated backward and forward through the MixColumns transformation with a probability of one (Step 1). Then, we simply need to find a match for the resulting input and output differences of the SubBytes layer of round r_2 (Step 2).

Step 1. We start the attack with 8 active bytes in state S_1' and S_2 (injected by message words M_1 and M_3) and compute backward and forward to two full active states S_2'' and S_2'''. The

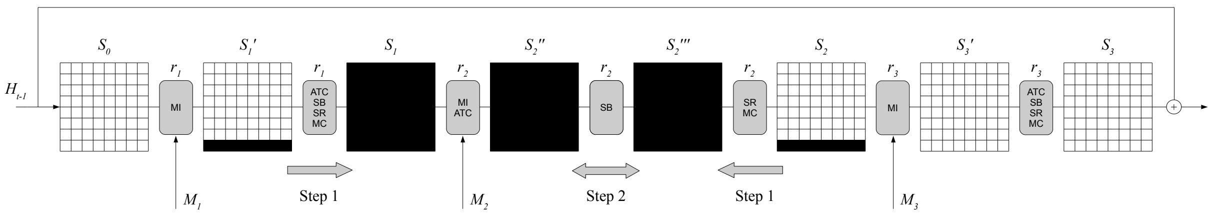

Fig. 4. We start with differences in states S_1' and S_2 injected by message words M_1 and M_3, and propagate backward and forward (Step 1) to find a match for the S-box of round r_2 (Step 2).

is happens with a probability of one due to the properties of the ShiftRows and MixColumns transformations. We repeat the computation 2^{28} times for message word M_1 and 2^{28} times for message word M_3. Hence, we get 2^{56} pairs of input/output differences for the S-boxes of round r_2.

Step 2. Next, we show how to find a match for these input/output differences of the 64 S-boxes. Note that for the S-box, the probability of a matching input/output difference pair is about one half if we can chose the (absolute) value of the S-box input freely. Hence, we expect to find a match for all 64 S-boxes with a probability of 2^{-64}. Note that we can adapt the differences of 8 S-boxes by injecting proper differences in message word M_2. This reduces the complexity of finding a matching pair for the full SubBytes layer to 2^{-56}. With the 2^{28} input and 2^{28} output differences of Step 1, we expect to find at least one match due to the birthday paradox. Note that in fact we get 2^{56} matches since we can choose from at least two possible values for each S-box match.

Once we have fixed the values of the state S_2'' such that the difference match for SubBytes layer this also determines S_0, and the differences in the message words. Hence, we have constructed a pseudo-collision for one Maxi-Round with complexity of 2^{28}. Note that the first Maxi-Round is equal for Twister-224/256 and Twister-384/512. Hence, by constructing a pseudo-collision for the first Maxi-Round we get a pseudo-collision for the compression function of Twister-224/256 and Twister-384/512. The attack has a complexity of about $2^{28}/3 \approx 2^{26.5}$ compression function evaluations.

3 Collision Attack on Twister-512

In this section, we show how the pseudo-collision attack on Twister-512 can be extended to the hash function. We first show how to construct collisions in the compression function of Twister-512 with a complexity of 2^{223} compression function evaluations. This collision attack on the compression function is then extended to a collision attack on the hash function. The extension is possible by combining a multicollision attack and a generalized birthday attack on the checksum. The attack has a complexity of about $2^{231.4}$ evaluations of the compression function of Twister-512.

3.1 Collision Attack on the compression Function of Twister-512

For the collision attack on the compression function of Twister-512 we can use the characteristic of the previous section in the last Maxi-Round (see Figure 5). Remember that in Twister-512 the 3 message words M_6, M_7 and M_8 are injected in the last Maxi-Round. Hence, we can use the first 5 message words $M_1 - M_5$ for a birthday match on 56 state bytes with a complexity of $2^{8 \cdot 56/2} = 2^{224}$. Since the 8 bytes of the last row can always be adapted by using the freedom in the (absolute) values of the message word M_6, we only need to match 56 out of 64 bytes. It can be summarized as follows:

1. Compute 2^{224} pseudo-collisions for the last Maxi-Round of Twister-512 and save them in a list L. This has a complexity of about $3 \cdot 2^{224}$ Mini-Round computations.
   Note that we can choose from $2^{3 \cdot 64} = 2^{192}$ differences in M_6, M_7 and M_8 in the attack. Furthermore, by varying the values of M_7, we get additional 2^{64} degrees of freedom. Hence, we can construct up to 2^{256} pseudo-collisions for the last Maxi-Round.
2. Compute the input of the last Maxi-Round by going forward and check for a match in the list L. After testing about 2^{224} candidates for the input of the last Maxi-Round we expect to find a match in the list L and hence a collision for the compression function of Twister-512. Note that finishing this step of the attack has a complexity of about $2^{224} + 2 \cdot 2^{160} + 2^{96} + 2^{32} \approx 2^{224}$ Mini-Round computations.

Hence, we can find a collision for the compression function of Twister-512 for an predefined chaining value with complexity of about 2^{223} compression function evaluations ($10 \cdot 2^{223}$ Mini-Round computations) and memory requirements of 2^{224}. The memory requirements of this attack can significantly be reduced by applying a memory-less variant of the meet-in-the-middle attack introduced by Quisquater and Delescaille in [5].

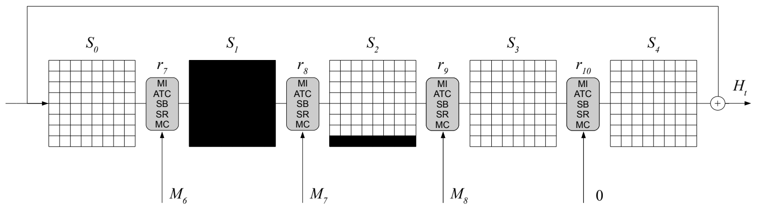

Fig. 5. The characteristic for the last Maxi-Round of Twister-512.

3.2 Collision Attack on the Hash Function Twister-512

In this section, we show how the collision attack on the compression function can be extended to the hash function. The attack has a complexity of about $2^{231.4}$ evaluations of the compression function of Twister-512. Note that the hash function defines, in addition to the common iterative structure, a checksum computed over the outputs of each Mini-Round which is then part of the final hash computation. Therefore, to construct a collision in the hash function we have to construct a collision in the iterative structure (*i.e.* chaining variables) as well as in the checksum. To do this we use multicollisions similar as in the recent collision attack on the hash function GOST [3].

A multicollision is a set of messages of equal length that all lead to the same hash value. As shown in [2], constructing a 2^t collision, *i.e.* 2^t messages consisting of t message blocks which all lead to the same hash value, can be done with a complexity of about $t \cdot 2^x$ for any iterated hash function, where 2^x is the cost of constructing a collision in the compression function. As shown in the previous section, collisions for the compression function of Twister-512 can be constructed with a complexity of 2^{223}. Hence, we can construct a $2 \cdot 2^{256}$ collision with a complexity of about $257 \cdot 2^{223} \approx 2^{231}$ evaluations of the compression function of Twister-512. With this method we get 2^{257} values for the checksum C that all lead to the same chaining value H_{257}.

To construct a collision in the checksum of Twister-512 we have to find 2 distinct messages consisting of 258 message blocks (257 message blocks for the multicollision and 1 message block for the padding) which produce the same value in the checksum. By applying a birthday attack we can find these 2 messages with a complexity of about 2^{257} checksum computations and memory requirements of 2^{256}. Due to the high complexity and memory requirements of the birthday attack, one could see this part as the bottleneck of the attack. However, the runtime and memory requirements can significantly be reduced by applying a generalized birthday attack introduced by Wagner in [6]. Wagner shows that if ℓ is a power of two then the memory requirements and the running time for the generalized birthday problem is given by $2^{n/(1+\lg \ell)}$ and $\ell \cdot 2^{n/(1+\lg \ell)}$, respectively. Note that in the standard birthday attack we have $\ell = 2^1$.

Table 1 shows the complexity and memory requirements for the collision attack depending on the choice of ℓ. The first row shows the complexity for a standard birthday attack ($\ell = 2$) with a checksum computation complexity above the birthday bound. For larger values of ℓ the complexity for the checksum computation decreases while the total complexity increases only slightly. Hence, in the case of $\ell = 4$ we can find a collision for the hash function Twister-512 with a complexity of about $2^{231.4}$ compression function calls.

Table 1. Complexities in base 2 logarithm for some values of ℓ.

ℓ	memory requirements	checksum computations	compression function computations
2	256	257	231
4	170,7	172,7	231,4
8	128	131	232
16	102,4	106,4	232,7
32	85,3	90,3	233,4
64	73,1	79,1	234,2
128	64	71	235
2^{15}	32	47	242
2^{23}	21,3	44,3	249,4
2^{31}	16	47	257

4 Conclusion

This paper shows two things: Although Twister is heavily based on a Merkle-Damgaard style iteration (as many other hash function like SHA-2), the corresponding reduction proof that reduces the collision resistance of the hash function to the collision resistance of the compression function is not applicable anymore. We show practical (in time and memory) attacks that invalidate such an assumption about the compression function.

Secondly, we give a theoretical collision short-cut attack on the hash function Twister-512. Although the practicality of the proposed attack might be debatable, it nevertheless exhibits non-random properties that are not present in SHA-512.

References

1. Ewan Fleischmann, Christian Forler, and Michael Gorski. The Twister Hash Function Family. Submission to NIST, 2008.
2. Antoine Joux. Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. In Matthew K. Franklin, editor, *CRYPTO*, volume 3152 of *LNCS*, pages 306–316. Springer, 2004.
3. Florian Mendel, Norbert Pramstaller, Christian Rechberger, Marcin Kontak, and Janusz Szmidt. Cryptanalysis of the GOST Hash Function. In David Wagner, editor, *CRYPTO*, volume 5157 of *LNCS*, pages 162–178. Springer, 2008.

4. National Institute of Standards and Technology. FIPS PUB 197, Advanced Encryption Standard (AES). Federal Information Processing Standards Publication 197, U.S. Department of Commerce, November 2001.
5. Jean-Jacques Quisquater and Jean-Paul Delescaille. How Easy is Collision Search. New Results and Applications to DES. In Gilles Brassard, editor, *CRYPTO*, volume 435 of *LNCS*, pages 408–413. Springer, 1989.
6. David Wagner. A Generalized Birthday Problem. In Moti Yung, editor, *CRYPTO*, volume 2442 of *LNCS*, pages 288–303. Springer, 2002.