

Cryptanalysis of the MCSSHA Hash Functions

Jean-Philippe Aumasson^{1,*} and María Naya-Plasencia^{2,†}

¹ FHNW, Windisch, Switzerland

² INRIA project-team SECRET, France

MCSSHA-3 is a hash function accepted as first round candidate in the NIST Hash Competition, and stands out as one of the simplest algorithms. MCSSHA-4 is a tweaked version of MCSSHA-3 proposed to foil some shortcut attacks [1] while retaining the merits of MCSSHA-3. MCSSHA-5 is a recent tweaked version of MCSSHA-4, proposed to counteract our attacks³ on MCSSHA-4. This abstract presents improved attacks on MCSSHA-3, and attacks on MCSSHA-4 and MCSSHA-5. We refer to [2–5] for complete specifications of the hash algorithms. Informal descriptions in appendix may suffice to understand our attacks.

1 Second-preimages for MCSSHA-3 and MCSSHA-5

Below we describe a second-preimage attack on MCSSHA-3 with 256-bit digest. The key observation is that each input of a message byte allows one to “choose” a byte of the 32-byte state. Since a message byte is input every four clockings, one controls $32/4 = 8$ bytes of the state after 32 clocks. Starting from a random state, with eight message bytes one can thus choose eight state bytes and gets $32 - 8 = 24$ uncontrolled bytes in the other cells.

Given an arbitrary message M , a second preimage attack then goes as follows:

1. Determine the 256-bit value of the internal state T after processing M (and before finalization).
2. Look for a collision “in the middle” on the 24 uncontrolled bytes, starting from the initial state forwards, and starting from T backwards

Standard collision search methods [6] requires negligible memory and require about $\sqrt{2^{24 \times 8}} = 2^{96}$ to find a collision, and generally $2^{3n/8}$ for n -bit digests. A collision directly gives a message that maps to T before the finalization. Since finalization is message-independent, one obtains the same digest as with the first message.

This technique also applies to MCSSHA-4 and MCSSHA-5: one then controls $1/3$ of the state, which is twice as large as in MCSSHA-3; second preimages can thus be found in $2^{2n/3}$ trials for the 256- and 512-bit versions.

Our technique does not directly extend to preimage search, because finding a state before finalization corresponding to a given image seems non-trivial.

*Supported by the Swiss National Science Foundation under project no. 113329.

†Supported in part by the French Agence Nationale de la Recherche under contract ANR-06-SETI-013-RAPIDE.

³Communicated to the author before publication.

2 Preimages for MCSSHA-4

We now present a preimage attack on MCSSHA-4, starting with a key observation. Note that our notations slightly differ from those in the specification of MCSSHA-4 (here N denotes the digest length, not the length of the register for “pre-hash computation”).

In the finalization stage, one starts from a N -byte register initialized to bytes 00, 01, 02, etc. Call this initial state R_0 . At each of the $4N$ steps, one updates the register with byte z_i , $i = 0, \dots, 4N - 1$. When the message length is a multiple of eight bits, we have

$$(z_0, \dots, z_{4N-1}) = (y_0, \dots, y_{2N-1}, y_0, \dots, y_{2N-1}) = y \| y ,$$

with y the state obtained after the message is processed. The final state of the N -byte register is returned as the digest. Now observe that:

1. One can easily find y_0, \dots, y_{N-1} such that after N steps the register comes back to R_0 . Repeating this sequence four times, one thus obtains a $4N$ -byte sequence y that maps R_0 to itself.
2. Given an arbitrary y_0, \dots, y_{N-1} , one can easily find y_N, \dots, y_{2N-1} such that after $2N$ steps the register comes back to the state R_0 . Repeating such a sequence twice, one again obtains a sequence y that maps to R_0 .

Now we present an attack that computes preimages of R_0 . For ease of exposition, we describe the attack for $N = 32$, i.e., for 256-bit digests:

1. Pick an arbitrary unique message M , perform the “pre-hash computation” stage, to reach a state $y = (y_0, \dots, y_{63})$.
2. Using the observation above, find y'_0, \dots, y'_{32} such that

$$(y_{33}, \dots, y_{63}, y'_0, \dots, y'_{32})$$

maps R_0 to itself after 64 finalization steps (out of 128 steps in total).

3. Choose the next message byte to obtain y'_0 as new value. Then, the two subsequent zeroes gives y'_1 and y'_2 with probability 2^{-16} . If the correct values are obtained, continue with the next message byte until y'_{32} . When a zero byte leads to a wrong value, go to step 2.

This algorithm terminates when all the 22 zero bytes lead to the desired byte value. This happens with probability 2^{-8} for each zero individually. When an erroneous value is obtained, one directly goes to step 2 and does not need to continue with the next zero. The expected number of trials is thus about $2^{8 \times 22} = 2^{176}$. When the state

$$(y_{33}, \dots, y_{63}, y'_0, \dots, y'_{32})$$

is reached, finalization will map R_0 to itself after 64 and 128 steps, thus returning R_0 as digest. Note that MCSSHA-4 (as MCSSHA-3) uses no specific padding rule, which simplifies our attack. The attack does not on MCSSHA-5.

The strategy is identical for 512-bit digests, leading to a complexity 2^{344} instead of 2^{176} to find one preimage of R_0 . The attack does not directly work for the 224- and 384-bit versions, because the register size does not divide 64 (resp., 128).

References

1. Jean-Philippe Aumasson and María Naya-Plasencia. Second preimages on MCSSHA-3. Public comment on the NIST Hash Competition, 2008.
2. Mikhail Maslennikov. Secure hash algorithm MCSSHA-3. Submission to NIST, 2008.
3. Mikhail Maslennikov. MCSSHA: Secure hash algorithms family. Presentation material for the First SHA-3 Conference, 2009.
4. Mikhail Maslennikov. Secure hash algorithm MCSSHA-4. http://registercsp.nets.co.kr/hash_competition.htm, 2009.
5. Mikhail Maslennikov. Secure hash algorithm MCSSHA-5. http://registercsp.nets.co.kr/hash_competition.htm, 2009.
6. Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *J. Cryptology*, 12(1):1–28, 1999.

MCSSHA-3

MCSSHA-3 computes a n -bit (or N -byte, $N = n/8$) digest by

1. Initializing a nonlinear feedback shift register (NFSR) with N byte elements.
2. Clocking the register once with input of a message byte.
3. Clocking the register three times with input of the zero byte.
4. Repeating steps 2 and 3 for each message byte (each message byte is input only once), to obtain a state $S = S_0, \dots, S_{N-1}$.
5. Clocking the NFSR $4N$ times, with as input the $4N$ -byte sequence

$$S_0, \dots, S_{N-1}, S_0, \dots, S_{N-1}, S_0, \dots, S_{N-1}, S_0, \dots, S_{N-1} .$$

MCSSHA-3 uses no message length padding, and avoids length-extension by the use of a distinct function for finalization. Details of the specification can be found in [2].

MCSSHA-4

For $N \in \{256, 512\}$, MCSSHA-4 computes a N -byte digest by

1. Initializing an NFSR with $2N$ byte elements.
2. Clocking the register once with input of a message byte.
3. Clocking the register *twice* with input of the zero byte.
4. Repeating steps 2 and 3 for each message byte to obtain a state $S = S_0, \dots, S_{2N-1}$.
5. Initializing an NFSR with N byte elements 00, 01, 02, etc.
6. Clocking this register $4N$ times, with as input the $4N$ -byte sequence

$$S_0, \dots, S_{2N-1}, S_0, \dots, S_{2N-1} .$$

MCSSHA-5

MCSSHA-5 is similar to MCSSHA-4, except that zero bytes are added between each two bytes in the finalization procedure.