Cryptanalysis of the MCSSHA Hash Functions

Jean-Philippe Aumasson[1] and María Naya-Plasencia[2]

[1] Nagravision SA, Switzerland
[2] INRIA projet-team SECRET, France

Abstract. This note describes attacks on the first round SHA-3 candidate MCSSHA-3, and on its subsequent versions MCSSHA-4, MCSSHA-5, and MCSSHA-6. We show a general strategy for searching second preimages for all those functions, and a dedicated preimage attack for MCSSHA-4. We deduce simple design criteria to avoid the presented attacks in MCSSHA-like designs.

Keywords: hash functions, SHA-3, cryptanalysis.

1 Introduction

The US National Institute of Standards and Technology recently started a public competition[3] to select its future cryptographic hash standard, which will be called SHA-3, and which will augment the existing SHA-2 family [1]. The so-called *SHA-3 Competition* received 64 submissions fall 2008, out of which 51 were selected by NIST as "first round candidates". NIST subsequently reduced the set of potential SHA-3's to 14 in July 2009.

MCSSHA-3 is a first round candidate in the SHA-3 Competition that did not make it to the second round. MCSSHA-4 is a modified version of MCSSHA-3 proposed to foil some shortcut attacks [2] while retaining the merits of MCSSHA-3. MCSSHA-5 is a modified version of MCSSHA-4, proposed to counteract the preimage attack reported in the present paper. Finally, MCSSHA-6 [3] is the ultimate attempt to thwart our attacks. We refer to [3–7] for complete specifications of the hash algorithms.

In this paper, we show that all four versions of MCSSHA admit shortcut second preimage attacks, and thus fail to satisfy the security requirements set by NIST for the SHA-3 Competition. Our attacks are relatively simple, for they are essentially meet-in-the-middle attacks adapted to the particular structure of each version of MCSSHA. In particular, they are comparable to the attacks presented in [8].

After a brief description of the MCSSHA hash algorithms in §2, we describe the second preimage attacks in §3, and a preimage attack specific to MCSSHA-4 in §4. Conclusions are drawn in §5.

[3]See `http://nist.gov/hash-competition`.

2 Description of the MCSSHA hash functions

This section briefly describes the hash functions MCSSHA-3, -4, -5, and -6. Performance figures for MCSSHA-4, -5, and -6 can be found on eBASH [9]. These are parametrized by a byte length N and a delay value Δ.

2.1 MCSSHA-3

MCSSHA-3 computes a digest of n bits (or N bytes, $N = n/8$) 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, so $\Delta = 3$.
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

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

The clocking mechanism involves one call to a 8×8 S-box, as only non-linear component.

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 [4].

2.2 MCSSHA-4

MCSSHA-4 differs from MCSSHA-3 by using a feedback register twice larger, by making two instead of three "blank clockings", and by initializing the register to a fixed value in the finalization step. More precisely, for $N \in \{64, 128\}$, MCSSHA-4 computes a $N/2$-byte digest by

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

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

2.3 MCSSHA-5 and MCSSHA-6

MCSSHA-5 is similar to MCSSHA-4, except that zero bytes are added between each two bytes in the finalization procedure. This makes the preimage attack described in §4 impossible.

MCSSHA-6's finalization slightly differs from MCSSHA-5's, to avoid the preimage attack in §4 without any performance penalty, unlike MCSSHA-5.

Detailed specifications of MCSSHA-5 and MCSSHA-6 can be found in [3,7].

3 Second-preimage attacks for the MCSSHA family

Below we describe our strategy for finding second-preimages for the MCSSHA functions, with register size N and delay Δ.

The key observation is that each input of a message byte allows one to "choose" a byte of the N-byte state. The number of controlled bytes in the internal state is $\lfloor N/(\Delta+1)\rfloor$. This is illustrated in Fig. 1, where the red bytes are controlled by the message block insertions, and the grey bytes correspond to the Δ "blank" rounds between two consecutive message blocks.

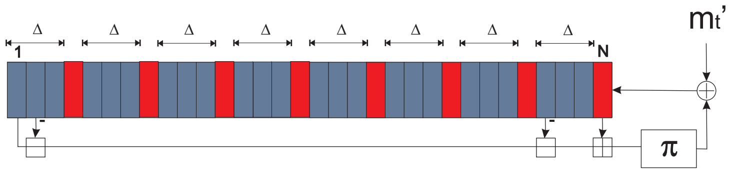

Fig. 1. Representation of controlled bytes (in red) in the N-byte register.

Since a clocking of the mechanism is invertible, we can perform a meet-in-the-middle attack, searching for a collision on the $N - \lfloor N/(\Delta+1)\rfloor \geq \Delta N/(\Delta+1)$ uncontrolled byte. Since the finalization phase is not (easily) invertible, we need to know the target value of the state before finalization, which makes preimage search impossible.

Since we search for a collision on $8(N - \lfloor N/(\Delta+1)\rfloor)$ bits, approximately $2^{4(N-\lfloor N/(\Delta+1)\rfloor)}$ forwards (resp. backwards) computations are required in average to find a collision. Standard collision search methods [10] require negligible memory. 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.

Table 3 gives the complexities of our attack for each MCSSHA version. Note that the attack can be used to find collisions for MCSSHA-3 faster than with a generic birthday search.

4 Preimage attack for MCSSHA-4

We present a preimage attack on MCSSHA-4, starting with a key observation: In the finalization stage, one starts from a $N/2$-byte register initialized to bytes $\texttt{00}, \texttt{01}, \texttt{02}$, etc. Call this initial state R_0. At each of the $2N$ steps, one updates the register with byte z_i, $i = 0, \ldots, 2N-1$. When the message length is a multiple of eight bits, we have

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

with y the state obtained after the message is processed. The final state of the $N/2$-byte register is returned as the digest. Now observe that:

Table 1. Time complexity of the attack applied to different versions of MCSSHA.

Version		N	Δ	Trials
MCSSHA-3	256	32	3	2^{96}
	512	64	3	2^{192}
MCSSHA-4	256	64	2	2^{172}
	512	128	2	2^{344}
MCSSHA-5	256	64	3	2^{192}
	512	128	3	2^{384}
MCSSHA-6	256	64	3	2^{192}
	512	128	3	2^{384}

1. One can easily find $y_0, \ldots, y_{N/2-1}$ such that after $N/2$ steps the register comes back to R_0. Repeating this sequence four times, one thus obtains a $2N$-byte sequence y that maps R_0 to itself.
2. Given an arbitrary $y_0, \ldots, y_{N/2-1}$, one can easily find $y_N, \ldots, y_{N-1}$ such that after N 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 $N = 64$, i.e., for 256-bit digests:

1. Pick an arbitrary unique message M, perform the "pre-hash computation" stage, to reach a state $y = (y_0, \ldots, y_{63})$.
2. Using the observation above, find $y'_0, \ldots, y'_{32}$ such that
$$(y_{33}, \ldots, y_{63}, y'_0, \ldots, y'_{32})$$
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
$$(y_{33}, \ldots, y_{63}, y'_0, \ldots, y'_{32})$$
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 work on MCSSHA-5.

The strategy is identical for 512-bit digests, leading to a complexity 2^{344} instead of 2^{512} 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).

5 Conclusion

We showed that the hash functions MCSSHA-3, MCSSHA-4, MCSSHA-5, and MCSSHA-6, submitted to the SHA-3 Competition, do not satisfy the security criteria set by NIST. However, none of our attacks is practical. Indeed, the lower complexity of our shortcut attacks is as high as 2^{84}, when considering the 224-bit version of MCSSHA-3.

For a MCSSHA-like architecture, a design criteria to avoid our second preimage attacks is to use a register size whose "uncontrolled" length (that is, the full size of the register times the fraction of zero bytes included, as in step 3 in §§2.1) at least twice as large as the digest size; that is, we need $4\,(N - \lfloor N/(\Delta + 1)\rfloor) \geq \ell$, where ℓ is the digest bit length.

Acknowledgments

This work was realized while the first author was with FHNW, Windisch Switzerland, and supported by the Swiss National Science Foundation under project no. 113329. The second author was supported in part by the French Agence Nationale de la Recherche under contract ANR-06-SETI-013-RAPIDE.

References

1. NIST: FIPS 180-2 secure hash standard (2002)
2. Aumasson, J.P., Naya-Plasencia, M.: Second preimages on MCSSHA-3. Public comment on the NIST Hash Competition (2008)
3. Maslennikov, M.: Secure hash algorithm MCSSHA-6. `http://registercsp.nets.co.kr/hash_competition.htm` (June 2009)
4. Maslennikov, M.: Secure hash algorithm MCSSHA-3. Submission to NIST (2008)
5. Maslennikov, M.: Secure hash algorithm MCSSHA-4. `http://registercsp.nets.co.kr/hash_competition.htm` (December 2008)
6. Maslennikov, M.: MCSSHA: Secure hash algorithms family. Presentation material for the First SHA-3 Conference (2009)
7. Maslennikov, M.: Secure hash algorithm MCSSHA-5. `http://registercsp.nets.co.kr/hash_competition.htm` (June 2009)
8. Khovratovich, D., Nikolic, I., Weinmann, R.P.: Meet-in-the-middle attacks on sha-3 candidates. In Dunkelman, O., ed.: FSE. Volume 5665 of LNCS., Springer (2009) 228–245
9. Bernstein, D.J., (editors), T.L.: eBACS: ECRYPT Benchmarking of Cryptographic Systems. http://bench.cr.yp.to Accessed 23 December 2009.
10. van Oorschot, P.C., Wiener, M.J.: Parallel collision search with cryptanalytic applications. J. Cryptology **12**(1) (1999) 1–28