Huge 2ndpreimages and collisions of khichidi-1

Prasanth Kumar Thandra and S.A.V. Satya Murty
Indira Gandhi Centre for Atomic Research,
Kalpakkam, Tamilnadu 603102.

prasanth@igcar.gov.in, satya@igcar.gov.in,

(draft)

Abstract

Khichidi-1[1] is a contestant of Sha-3[2]. A collision attack on khichidi is proposed in [3]. In this paper we exploited a weakness in the design of the algorithm. This allowed us to propose two kind of attacks 1) 2ndpreimage attack[4], 2) Collision attack[4]. Our attacks are applicable to all the versions 224, 256, 384 and 512 and it is potentially strong.

Key words: khichidi-1, Hash function, 2ndpreimage resistance, Collision resistance,.

Introduction: Khichidi-1 [1] is a contestant of Sha-3 program. The algorithm uses a sequential combination of shuffling of bits, T-function, and LFSR in its compression function, C. For further details of the algorithm, please see [1]. collision attacks are proposed on khichidi-1 in [3].Our observation showed that the algorithm has potential flaws in the design. In this paper, we proved that a khichidi-1 is vulnerable to 2ndpreimage attacks as well as collision attacks. With our attack, both 2ndpreimage and collisions can be found for almost all messages of practical purpose. Our attack require a pre-computed table, tablexC, with ($2*2^{32}$) words of 32-bit length. Once the table is computed, finding a 2ndpreimage for a given message requires only calling of Round1 for the given message and its 2ndpreimage. Finding a collision also requires the same. Hence, the complexity of attacks requires the call of compression function for (N-1) times for finding a first collision/2ndpreimage, where N is the total number of blocks for compression in both message and its 2ndpreimage/collision. Our attack allows constructing huge number of collisions with same hash value, similarly 2ndpreimages.

In khichidi, the compression function C used recursively to compress padded messages of lengths multiple of 448, 512, 768 and 1024 in respective versions in CBC-mode [5]. The compression function, C, comprises a shuffling function(S) followed by a T-function (T) followed by a LFSR(L). These three functions act on 32-bit words. The most interesting thing is that our attack is independent of the details of these functions. A modification in any of these functions does not make khichidi as secured against our attack except that the computation of tablexC changes. Hash computation for a message using khichidi involves mainly two parts one is compression of the message, "Round1", second is final hash computation. This second part involves in total 5 rounds, "Round 2 to 6". The padded message is divided in to blocks of respective lengths and these blocks are compressed to a fixed length through a recursive operation of function C in CBC-mode. Hence, the output of Round1 has a fixed length depending on the version of Khichidi-1.

This fixed length output is fed to “Round2” followed by “Round3”, “Round4”, “Round5” and “Round6”. The output of “Round6” is the hash value of the message.

Our attack concentrates on the compressed output of Round1. Using our attack for a given message, we can find huge number of messages, which generates the same output as that of given message after Round1. Hence, irrespective of the details of “Round2-6” the strategy of our attack works. In finding collision/2ndpreimage it dose not even required to compute these last five rounds.

Attack: The only pre-requirement for finding collisions or 2ndpreimage is that tablexC involves x, C(x), for all the values of x, where x is a 32-bit number. Single table computed once is sufficient for finding collisions or 2ndpreimages for any message for all the versions of khichidi.

Procedure of the attack is explained for 224 version with a general example. The procedure is same for finding a 2ndpreimages or collisions. We are going to find a collision/2ndpreimage for a (448-32) bit message, M. Length of the message can be expressed as, (448(p)-32)-bits where p=1, 2, 3, 4…etc. M^i is the i^{th} block of message M and M^i_j is the j^{th} word, 32-bit, of M^i. When M is padded with L^1(which is M^2_6=0X800001A0 in this case) padded-message length becomes 448 bits. The 2ndpreimage/collision for padded-message M is the message of length (448(p + q)-32) where q=1, 2, 3… etc. One restriction on the lengths of unpadded-messages M, m is that the padding process should be restricted to 32 bits. That is the length of m should be such that its binary form can be expressed using not more than 31-bits. For present case let q is also 1. So, the length of unpadded-message m is 864 bits, and the padding word is L^2, which is m^4_6=0X80000360. Hence, the padded-message m has length of 896 bits. Now the “Round1” of M is shown in Fig. 1(a). The output of “Round1” for padded-message M is shown in Fig. 1(a). The 4 blocks and corresponding words of the padded-message m and their compression is as shown in Fig. 1(b). The output of Round1 of this “padded-message m” should be equal to the output of Round1 of “padded- message M”. If this can happen the result is either a collision or a 2ndpreimage. The only word fixed in “padded-message m” is L^2. The requirements of message m to become a 2ndpreimage/collision of M and achieving these requirements are given below systematically.

1) The first requirement is D^2_6 should be equal to $C(L^2 ☺ D^2_5 ☺ d^3_6)$. To make it happen, find x from tablexC mentioned earlier such that $D^2_6= C(x)$. Now, chose $d^3_6 = (L^2 ☺ D^2_5 ☺ x)$.This makes $D^2_6=C(L^2 ☺ D^2_5 ☺ d^3_6)$.

2) But, $d^3_6 = C(d^2_6 ☺ m^3_6 ☺ d^3_5)$. Now from tablexC find y such that $d^3_6 = C(y)$. Hence, m^3_6 should be equal to $(d^2_6 ☺ y ☺ d^3_5)$. $m^3_6=(d^2_6 ☺ y ☺ d^3_5)$. Now, Chose words from m^1_0 to m^3_5 randomly and calculated the values of d^i_js upto d^3_5. As d^3_5, d^2_6 and y are known and hence m^3_6.

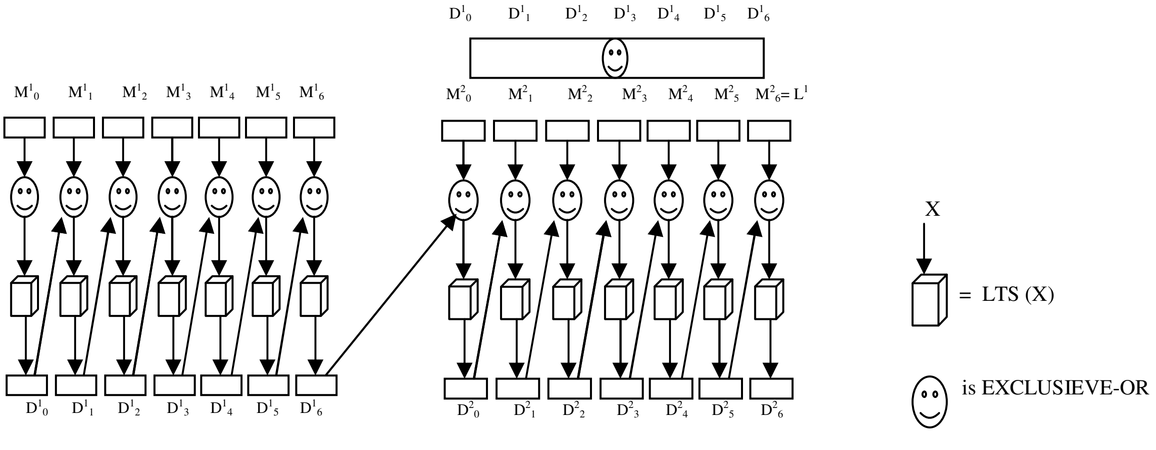
D^1_0 D^1_1 D^1_2 D^1_3 D^1_4 D^1_5 D^1_6
M^1_0 M^1_1 M^1_2 M^1_3 M^1_4 M^1_5 M^1_6
M^2_0 M^2_1 M^2_2 M^2_3 M^2_4 M^2_5 $M^2_6= L^1$
D^1_0 D^1_1 D^1_2 D^1_3 D^1_4 D^1_5 D^1_6
D^2_0 D^2_1 D^2_2 D^2_3 D^2_4 D^2_5 D^2_6
X
= LTS (X)
is EXCLUSIEVE-OR

Fig. 1(a) Round1 of padded message M of length 448 bits

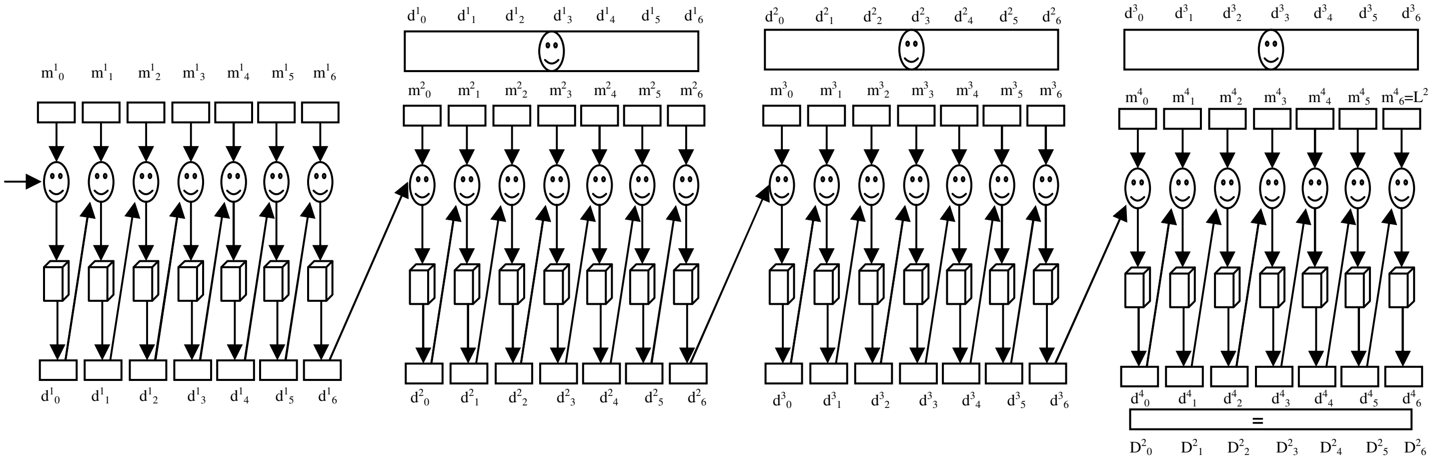
d^1_0 d^1_1 d^1_2 d^1_3 d^1_4 d^1_5 d^1_6
d^2_0 d^2_1 d^2_2 d^2_3 d^2_4 d^2_5 d^2_6
d^3_0 d^3_1 d^3_2 d^3_3 d^3_4 d^3_5 d^3_6
m^1_0 m^1_1 m^1_2 m^1_3 m^1_4 m^1_5 m^1_6
m^2_0 m^2_1 m^2_2 m^2_3 m^2_4 m^2_5 m^2_6
m^3_0 m^3_1 m^3_2 m^3_3 m^3_4 m^3_5 m^3_6
m^4_0 m^4_1 m^4_2 m^4_3 m^4_4 m^4_5 $m^4_6=L^2$
d^1_0 d^1_1 d^1_2 d^1_3 d^1_4 d^1_5 d^1_6
d^2_0 d^2_1 d^2_2 d^2_3 d^2_4 d^2_5 d^2_6
d^3_0 d^3_1 d^3_2 d^3_3 d^3_4 d^3_5 d^3_6
d^4_0 d^4_1 d^4_2 d^4_3 d^4_4 d^4_5 d^4_6
=
D^2_0 D^2_1 D^2_2 D^2_3 D^2_4 D^2_5 D^2_6

Fig. 1(b) Round1 of padded message m of length 896 bits

3) The next requirement, from Fig. 1(b), to have a collision/attack is D^2_0 should be equal to $C(m^4_0$ ☺ d^3_6 ☺ $d^3_0)$. As, d^3_0 and d^3_6 are known already, using tablexC m^4_0 is found.

4) The next requirement, from Fig. 1(b), to have a collision/attack is D^2_1 should be equal to $C(m^4_1$ ☺ D^2_0 ☺ d^3_1. As d^3_1 and D^2_0 are already known, using tablexC m^4_1 is found.

5) The next requirement, from Fig. 1(b), to have a collision/attack is D^2_2 should be equal to C(m^4_2 ☺ D^2_1 ☺ d^3_2) . As d^3_2 and D^2_1 are already known, using tablexC m^4_2 is found.

6) The next requirement, from Fig. 1(b), to have a collision/attack is D^2_3 should be equal to C(m^4_3 ☺ D^2_2 ☺ d^3_3) . As d^3_3 and D^2_2 are already known, using tablexC m^4_3 is found.

7) The next requirement, from Fig. 1(b), to have a collision/attack is D^2_4 should be equal to C(m^4_4 ☺ D^2_3 ☺ d^3_4) . As d^3_4 and D^2_3 are already known, using tablexC m^4_4 is found.

8) The next requirement, from Fig. 1(b), to have a collision/attack is D^2_5 should be equal to C(m^4_5 ☺ D^2_4 ☺ d^3_5) . As d^3_5 and D^2_4 are already known, using tablexC m^4_5 is found.

9) The last word $m^4_6=L^2$ is the padding word to m.

Now the message m is completely known with random values from m^1_0 to m^3_5 and calculated values from m^3_6 to m^4_5 and the padding word $L^2=m^4_6$=0X80000360. This padded-message m generates same output in Round1 as that of message M. Hence, both M, m generates the same hash value.

More collisions/2ndpreimages: Here, padded-message m has 2 blocks and its first collision M has 4 blocks and the compression function has been called for 5 (2+4 -1) times. Finding a second collision/2ndpreimage is much easier. For the padded-message M append (448-32) bits. Now the new padded-message $\mathcal{M}$ is

$$M\ || m^5_{j=0,1,2,3,4,5,6}\ ||\ m^6_{j=0,1,2,3,4,5,}\ || m^6_{6=} L^3$$

Where, the last word m^6_6 is the padding word. Now, as already explained above chose the values of $m^5_{j=0,1,2,3,4,5}$ randomly and follow the steps mentioned above to obtain the values of $m^5_{6,}$ $m^6_{j=0,1,2,3,4,5}$ to make $\mathcal{M}$ as a collision/2ndpreimage to both m, M, $\mathcal{M}$. In finding the second collision/2ndpreimage, the compression function C has been called only once. Proceeding further for more collisions by appending (448-32) bits as explained to the latest message requires a call to compression function only once. The above attack is applicable to all the versions of the algorithm khichidi-1. The procedure is same. But, as the number of words in each block increase with 256, 384 and 512 versions, corresponding changes are required in the procedure. As these changes are very straightforward we are not presenting those in this paper.

Complexity: From the above discussion, complexity of the attack can be summarized as follows. To find N collisions/2ndpreimages for a padded-message M with N number of blocks requires 3N calls of compression function. In addition to this, a pre-computation is required for tablexC.

Similar attacks: As above similar kind of construction of collisions/2ndpreimages for padded-message M is also possible where the collisions/2ndpreimages constructed are shorter than M by even number of blocks Similar to the above attack where the collisions found to padded-message M are lengthier than M by even number of blocks. But, as the procedure is almost same we are not giving the complete details of this construction here.

Conclusion: In this paper, we presented a way constructing very huge number of 2ndpreimages to a given message and a way of finding collisions for khichidi-1. Our procedure complexity is very less with a pre-computed table. The procedure requires the number of calls to compression function linearly dependent on the total number of blocks of messages involved in finding a collision/2ndpreimage. The procedure is same for finding collisions/2ndpreimages for all the versions of khichidi. Our attack clearly exploited the weaknesses involved in the design of the algorithm.

References

[1] Natarajan Vijayarangan, “A NEW HASH ALGORITHM: Khichidi-1”, available at http://www.nist.gov/groups/ST/hash/sha-3/Round1/submissions_rnd1.html

[2] www.nist.gov/hash-competition.

[3] *Nicky Mouha* – “Collision for Khichidi-1”, available at http://www.nickymouha.be/software-en.html

[4] A. Menezes, P. van Oorschot, and S. Vanstone, Handbook of Applied Cryptography, CRC press, 1996

[5] en.wikipedia.org/wiki/Block_cipher_**mode**s_of_operation