Improvement in SHA-3 Algorithm using different Internal Methods and Operations

Vanita Jain[1], Rishab Bansal[2], Mahima Swami[3] and Dharmender Saini[4]

[1]Bharati Vidyapeeth's College of Engineering, New Delhi,
vanita.jain@bharatividyapeeth.edu
[2]Bharati Vidyapeeth's College of Engineering, New Delhi,
rishabbansal.it1@bvp.edu.in
[3]Bharati Vidyapeeth's College of Engineering, New Delhi, mahima.it1@bvp.edu.in
[4]Bharati Vidyapeeth's College of Engineering, New Delhi, dsaini77@gmail.com

Abstract. In this paper, we propose a novel approach to change the internal architecture of Schoolbook implementation of SHA-3 Algorithm. With this new architecture and internal operations we aim at decreasing the time taken by the original algorithm. Specifically, we reduce the number of internal rounds of the algorithm and to compensate for the loss in confusion and diffusion we make SHA-3's internal operations more complex. We change the algorithm on its core bit level and make its internal operation more interdependent on the neighbouring bits to achieve more diffusion which results in better confusion.

Keywords: hashing functions, information security, cryptog- raphy, SHA-3, secure hashing, sponge construction, KECCAK

1 Introduction

Cryptographic hash functions [1] [2], especially for authentication applications, such as message authentication codes, password protection and digital signature, are of major importance for a number of security applications. Cryptography is a form of testing data integrity. It is used to make sure no access or modification is made to the data transmitted in a message. [3] The intensity of SHA [4] mainly depends on a range of factors. These include the facility to calculate the hash value, the failure to generate a message that has a hash, the inability to modify a message without altering its hash. In addition to this, two separate messages of the same hash value can not be identified. Safe hashing algorithms can not only protect data from misuse or modification but also guarantee user authentication. SHA is used to sign content with a digital fingerprint and can be used for encrypting and decrypting passwords by many operating systems. Even in wireless communications, SHA-1 and SHA-2 are very important and are widely used for secure communication [5]. It shows, however, numerous weaknesses and limitations which cause an appropriate replacement to be found. The hash functions [6] [7] [8] can be classified into 2 types: Keyed and Unkeyed. Keyed Hash functions can be used for Message Authentication Code (MAC) [9]. It requires

two inputs, a message and a secret key. Unkeyed Hash functions can be classified into 3 categories: block ciphers, customised functions arithmetic functions. Hash functions have one way property which means computing things in one direction is easy and the vice versa is hard. For example, for a message m, it is easy to calculate H(m)=M, but it is hard to obtain m from a given M. Collision attack [10] means finding a message n such that H(m) = H(n).

2 Related Work

After some successful attacks on SHA-1 [11] [12], NIST launched a public competition to promote the development of a new cryptographic hash function. The winning algorithm of this competition will be named as SHA-3. Team Keccak won the competition and their algorithm is specified in Federal Information Processing Standard (FIPS) 180-3, Secure Hash Standard. Keccak is based on the sponge construction. [13] The benefit of using this type of architecture is to obtain a more secure and robust against generic and known attacks. Sponge construction also makes the use of the compression function more simple and flexible. Keccak has 2 phases: Absorption(input phase) and Squeezing(output phase). In the Absorption Phase, input blocks are XORed into a sub-state, this state is then converted as a whole using the permutation function. In Squeezing Phase, output blocks are obtained from the same subset and altered with the state transformation function. The size of this altered state is called rate(r). The size of the part which remains unaltered is called capacity(c). For a permutation function that computes b-bit blocks, capacity is equal to b-r. Security level of the scheme depends upon the capacity of the algorithm. Maximum security level of the scheme is usually half of capacity $2^{c/2}$. Padding makes sure that the length of the input is divisible by r to break input into blocks. A pattern of 10*1 is used for padding in SHA-3. The size of the input block varies depending on the size of the output: Keccak-512 has a block size of 576 bits, Keccak-384 has 832 bits, Keccak-256 has 1088 bits and Keccak-224 has 1152. [14] SHA-3 has 5 main functions through which the message blocks are passed 24 times, namely Iota, Chi, Pi, Rho, Theta. The input block is broken into a tensor of 1600 bits (5 x 5 x 64). Then these 1600 bits are passed through these functions 24 times (in original implementation). P[u,w] is an array with u,w = 0,1,2,3,4

2.1 Theta (θ)

In this step, each bit is XOR-ed with 10 other neighbouring bits.[10]

$$Q[u] = XOR(P[u,0], P[u,1], P[u,2], P[u,3], P[u,4]) \tag{1}$$

$$R[u] = XOR(Q[u-1], rotate(Q[u+1], 1)) \tag{2}$$

$$P[u,w] = XOR(P[u,w], R[u]) \tag{3}$$

$$[u, w = 0, 1, 2, 3, 4]$$

2.2 Rho (ρ)

In this step, each row of 64 bits is rotated cyclically according to the rotation table given in Table 1[10].

Table 1. Rotation Constants (Original)

y \x	0	1	2	3	4
0	0	36	3	41	18
1	1	44	10	45	2
2	62	6	43	15	61
3	28	55	25	21	56
4	27	20	39	8	14

2.3 Pi (π)

In this step, all the bits are changed according this equation[10]:

$$Q[u, 2u + 3w] = P[u, w] \tag{4}$$

$$[u, w = 0, 1, 2, 3, 4]$$

2.4 Chi (χ)

In this step, binary bit wise operations like AND, OR, XOR, NOT are performed in the bits according to this equation:[10]

$$P[u, w] = XOR(Q[u, w, NOT(Q[u + 1, w])ANDQ[u + 2, w])) \tag{5}$$

$$[u, w = 0, 1, 2, 3, 4]$$

2.5 Iota (ι)

To break the uniformity of the process, in this step a unique constant is XOR-ed with the P[0,0] row. This produces entropy in the each step. The constants are given in Table 2[10].

Table 2. Round Constants (Original)

I[01] = 1	I[02] = 32898
I[03] = 9223372036854808714	I[04] = 9223372039002292224
I[05] = 32907	I[06] = 2147483649
I[07] = 9223372039002292353	I[08] = 9223372036854808585
I[09] = 138	I[10] = 136
I[11] = 2147516425	I[12] = 2147483658
I[13] = 2147516555	I[14] = 9223372036854775947
I[15] = 9223372036854808713	I[16] = 9223372036854808579
I[17] = 9223372036854808578	I[18] = 9223372036854775936
I[19] = 32778	I[20] = 9223372039002259466
I[21] = 9223372039002292353	I[22] = 9223372036854808704
I[23] = 2147483649	I[24] = 9223372039002292232

3 PROPOSED ARCHITECTURE

There are 5 main operations in KECCAK Algorithm: theta, rho, pi, chi, iota. We changed theta, rho, chi, iota.

3.1 Theta (θ)

Original implementation This is the original implementation[10] of theta in SHA-3.

$$Q[u] = XOR(P[u,0], P[u,1], P[u,2], P[u,3], P[u,4]) \tag{1}$$

$$R[u] = XOR(Q[u-1], rotate(Q[u+1], 1)) \tag{2}$$

$$P[u,w] = XOR(P[u,w], R[u]) \tag{3}$$

$$[u, w = 0, 1, 2, 3, 4]$$

Our implementation This is our implementation of theta method. These extra operations increases the algorithm's diffusion factor. Instead of 10, it now XORs each bit with 20 neighbouring bits.

$$Q[u] = XOR(P[u,0], P[u,1], P[u,2], P[u,3], P[u,4]) \tag{6}$$

$$R[u] = XOR(Q[u-1], rotate(Q[u+1], 1)) \tag{7}$$

$$S[u] = XOR(R[u-2], rotate(R[u-2], 1)) \tag{8}$$

$$T[u] = XOR(S[u-2], rotate(S[u-3], 1)) \tag{9}$$

$$P[u,w] = XOR(P[u,w], T[u]) \tag{10}$$

$$[u, w = 0, 1, 2, 3, 4]$$

Table 3. Rotation Constants (Original)

y \x	0	1	2	3	4
0	0	36	3	41	18
1	1	44	10	45	2
2	62	6	43	15	61
3	28	55	25	21	56
4	27	20	39	8	14

3.2 Rho (ρ)

Original implementation Table 3[10] shows the original rotation constants.

Our implementation Table 4 shows our modified rotation constants. We tested random values for these constants and selected those which provided maximum diffusion properties.

Table 4. Rotation Constants (Modified)

y \x	0	1	2	3	4
0	8	43	26	4	8
1	7	34	21	53	47
2	60	18	37	57	4
3	20	17	19	41	44
4	25	2	58	51	62

3.3 Chi (χ)

Original Implementation This is the original implementation[10] of the Chi function.

$$P[u,w] = XOR(Q[u,w,NOT(Q[u+1,w])ANDQ[u+2,w])) \tag{5}$$
$$[u,w = 0,1,2,3,4]$$

Our Implementation This is our implementation of the Chi function. We added extra XOR and AND operations to increase the inter dependency of the bits.

$$P[u,w] = XOR(Q[u,w],Q[u-2,w],Q[u+1,w])AND(NOT(Q[u-1,w])) \tag{11}$$
$$u,w = 0,1,2,3,4$$

3.4 Iota (ι)

Original Implementation Table 5[10] shows the original round constants. 24 Constants for 24 rounds.

Table 5. Round Constants (Original)

I[01] = 1	I[02] = 32898
I[03] = 9223372036854808714	I[04] = 9223372039002292224
I[05] = 32907	I[06] = 2147483649
I[07] = 9223372039002292353	I[08] = 9223372036854808585
I[09] = 138	I[10] = 136
I[11] = 2147516425	I[12] = 2147483658
I[13] = 2147516555	I[14] = 9223372036854775947
I[15] = 9223372036854808713	I[16] = 9223372036854808579
I[17] = 9223372036854808578	I[18] = 9223372036854775936
I[19] = 32778	I[20] = 9223372039002259466
I[21] = 9223372039002292353	I[22] = 9223372036854808704
I[23] = 2147483649	I[24] = 9223372039002292232

Our Implementation Table 6 shows our modified round constants. 16 Constants for 16 rounds.

Table 6. Round Constants (Modified)

I[01] = 7948088626323794	I[02] = 6988136757012497
I[03] = 39791030927721416	I[04] = 10809755619314387
I[05] = 8147066428805446	I[06] = 9673464656433063
I[07] = 712894840135913	I[08] = 9856031517555377
I[09] = 7519438105630892	I[10] = 2180132335568229
I[11] = 22694087947679686	I[12] = 4791473861756359
I[13] = 9756078218014334	I[14] = 9910404914141336
I[15] = 7734607116733396	I[16] = 4660076906243047

4 Results

To compensate for the loss of Confusion and diffusion factor we made the internal bits more interdependent. We achieved this by adding more internal methods and changing existing bit wise operations. After trying different combinations of methods, operations and constants we settled on those which gave us maximum confusion and diffusion.

Table 7 shows the running time of original (24 round implementation) and our modified (16 round implementation) with our changed internals and reduced number of rounds. Table V shows an average of 16% reduction in time which was observed for over 1,000,000 sample random inputs.

Table 7. Reduction in running time

SIZE (Bytes)	Original (seconds)	Modified (seconds)	% Decrease in time
10	0.2071	0.178205	13.9546
100	1.14742	0.920604	19.7676
1000	13.7301	9.46673	31.0512
10000	120.569	110.511	8.3424
100000	97.4402	80.9203	16.953
1000000	101573	93112.5	8.3295

Table 8 shows some of the hashes computed on both the functions for the same inputs. The diffusion factor can be seen in the last and second last case where just a single change drastically changed the output.

Table 8. Sample Output

Message	Original
	Modified
This is a super secret message	386a35c28e45b7e5783f6cf44ca696d5ae 8db06a5eed53e4
	c86a41008bf7dc782156c01c438f29871a d535b009138ac1
Cryptography	cdc9d5c7a7cd902bfc1e6fc142b00aaa981 2ea835349bdd2
	28c9409a4db203a5ad445cd3e233e04b98 3e550f73ce9c3a
Hashing	81d968688b4bbaff13ac3d884bf91f3e0be2 708c53247c17
	66a89460f04d70a3f18e28da3ddeffa271d1 02e179a0aa44
Hashingg	0cd2b4c75f337d88771fd5625228c693013f 96aa03df3b80
	3a5049c3bb1c571ef0487cbd153bed70c718 d1095283fe59

To check the quality of hash being produced we ran both the implementations on 1000 random inputs of 300+ bytes in size and changed the input by various

degrees, mapped the corresponding changes in both the implementation. The changes was obtained by calculating the number of bytes being changed when the input is changed by various degrees. Table IX shows the % difference in both the cases.

Table 9. changes produced in the output

Percentage change in input (%)	Total changes in original 24 round implementation	Total changes in our modified 16 round implementation	Percentage Difference (%)
10	23913	23920	-0.07
20	23916	23908	+0.08
30	23887	23904	-0.17
40	23898	23887	+0.11
50	23914	23908	+0.06

5 CONCLUSIONS

We were able to produce a 16% reduction in the running time of the schoolbook implementation of SHA-3 without decreasing the confusion and diffusion factor of the algorithm.

6 FUTURE SCOPE AND LIMITATIONS

This algorithm can be used in Web Browsers. SHA-2 is used in browsers because of it's less computation time. This algorithm can replace SHA-2 in Web Browsers or some other light weith applications which need secure hashing in less time. KECCAK algorithm is more secure than SHA-2 and is not vulnerable to attacks known for SHA-1 and SHA-2 like length extension attacks. We reduced 8 rounds in the algorithm from original 24. The algorithm is still secure from brute forcing and general crypt-analysis. This is called computational security, because no one has the computation power right now to brute force 16 round SHA-3 algorithm and crack it within the relevant time frame. These changes on internal operations and methods are done on the schoolbook implementation of SHA-3 without using any optimisation techniques. In real world many space and time optimisation techniques like multi-threading are done to make the code run faster. Same practices can also be applied on this algorithm as well.

References

1. Sobti, Rajeev Ganesan, Geetha. (2012). Cryptographic Hash Functions: A Review. International Journal of Computer Science Issues, ISSN (Online): 1694-0814. Vol 9. 461 - 479.

2. Preneel B. (2010) Cryptographic Hash Functions: Theory and Practice. In: Gong G., Gupta K.C. (eds) Progress in Cryptology - INDOCRYPT 2010. INDOCRYPT 2010. Lecture Notes in Computer Science, vol 6498. Springer, Berlin, Heidelberg
3. Cryptographic hash function, Aug. 2014, accessed on Dec. 2019[online] available: en.wikipedia.org/wiki/Cryptographic_hash_function.
4. Sahu, Aradhana, Ghosh, Samarendra. (2017). Review Paper on Secure Hash Algorithm With Its Variants. 10.13140/RG.2.2.13855.05289.
5. A. Moh'd, N. Aslam, H. Marzi and L. A. Tawalbeh, "Hardware Implementations of Secure Hashing Functions on FPGAs for WSNs," Proceedings of the 3rd International Conference on the Applications of Digital Information and Web Technologies (ICADIWT 2010), Istanbul, 12-14 July 2010.
6. Alfred M., Oorschot P., and Vanstone S., Handbook of Applied Cryptography, CRC press, 1997
7. Bruce S., Applied Cryptography: Protocols, Algorithms and Source Code in C, John Wiley and Sons, Canada, 1996.
8. Stallings W., Cryptography and Network Security Principles and Practices, Prentice Hall Press Upper Saddle River, 2010.
9. Bellare M., Canetti R., Krawczyk H. (1996) Keying Hash Functions for Message Authentication. In: Koblitz N. (eds) Advances in Cryptology — CRYPTO '96. CRYPTO 1996. Lecture Notes in Computer Science, vol 1109. Springer, Berlin, Heidelberg
10. Andreeva, E., Bogdanov, A., Mennink, B. et al. On security arguments of the second round SHA-3 candidates. Int. J. Inf. Secur. 11, 103–120 (2012). https://doi.org/10.1007/s10207-012-0156-7
11. Stevens, Marc Bursztein, Elie Karpman, Pierre Albertini, Ange Markov, Yarik. (2017). The First Collision for Full SHA-1. 570-596. 10.1007/978-3-319-63688-7_19.
12. Wang X., Yin Y.L., Yu H. (2005) Finding Collisions in the Full SHA-1. In: Shoup V. (eds) Advances in Cryptology – CRYPTO 2005. CRYPTO 2005. Lecture Notes in Computer Science, vol 3621. Springer, Berlin, Heidelberg
13. G. Bertoni, J. Daemen, M.el Peeters and G. Van Assche, "Keccak Sponge Function Family Main Document," NIST, University of California Santa Barbara, Santa Barbara, 2009.
14. G. Bertoni, et al., The Keccak reference, Version 3.0, January 14, 2011 http://keccak.noekeon.org/Keccak-reference-3.0.pdf