International Journal of Emerging Engineering Research and Technology
Volume 3, Issue 12, December 2015, PP 146-153
ISSN 2349-4395 (Print) & ISSN 2349-4409 (Online)

Design of SHA-3 Algorithm using Compression Box (3200 bit) for Digital Signature Applications

Avinash Kumar[1], C.H Pushpalatha[2]

[1]*Department of ECE, GONNA INSTITUTE OF TECHNOLOGY, Vishakhapatnam, India (PG Scholar)*
[2]*Department of ECE, GONNA INSTITUTE OF TECHNOLOGY, Vishakhapatnam, India (Associate Professor)*

ABSTRACT

SHA3 algorithm had proposed by five people with five different approaches. In that NIST (National Institute of Standards and Technology) selected one approach, that approach was proposed by Keccak. The Keccak-f permutation is the basic component of Keccak Hash function and supports 224-bit, 256-bit, 384-bit and 512-bit hash variants. It consists of number of rounds and each round is the combination of logical operations and bit permutations. Keccak is generated from sponge function with Keccak [r, c] members. It is categorized by these additional functions i.e. bit rate (r) and capacity (c). The addition of r + c gives width of the Keccak function permutation and is it is further limited to values as indicated 25, 50, 100, 200, 400, 800, 1600. After that Keccak, SHA3 algorithm using with memories but it will take more area. SHA3 have different variants like sha224, sha256, sha512, sha1024. The basic SHA3 using 512 bits converts 128 bits input into 1600 bits in the intermediate stage with using one C-box and performs 24 rounds. In our paper, for improving the security margin with respect to 512-bit, we are designing 128 bit Keccak sequential architecture for SHA-1024 variant by converting it into 3200 bits in the intermediate stage using two C-boxes and 24 rounds , which is to be implemented using Xilinx 13.2.

Keywords: Theta, Rho, Pi, Chi, Iota.

INTRODUCTION

MD5 is one in a series of message digest algorithms designed by Professor Ronald Rivest of MIT (Rivest, 1992). When analytic work indicated that MD5's predecessor MD4 was likely to be insecure, Rivest designed MD5 in 1991 as a secure replacement. (Hans Dobbertin did indeed later find weaknesses in MD4.)In 1993, Den Boer and Baseliners gave an early, although limited, result of finding a "pseudo-collision" of the MD5 compression function; that is, two different initialization vectors which produce an identical digest. In 1996, Dobbertin announced a collision of the compression function of MD5 (Dobbertin, 1996). While this was not an attack on the full MD5 hash function, it was close enough for cryptographers to recommend switching to a replacement, such as SHA-1 or RIPEMD-160.The size of the hash value (128 bits) is small enough to contemplate a birthday attack. MD5CRK was a distributed project started in March 2004 with the aim of demonstrating that MD5 is practically insecure by finding a collision using a birthday attack. SHA-1 produces a message digest based on principles similar to those used by Ronald L. Rivest of MIT in the design of theMD4 and MD5 message digest algorithms, but has a more

Address for correspondence:
avinashknsingh@gmail.com

conservative design. The original specification of the algorithm was published in 1993 under the title Secure Hash Standard, FIPS PUB 180, by U.S. government standards agency NIST (National Institute of Standards and Technology). This version is now often namedSHA-0. It was withdrawn by the NSA shortly after publication and was superseded by the revised version, published in 1995 in FIPS PUB 180-1 and commonly designated SHA-1. SHA-1 differs from SHA-0 only by a single bitwise rotation in the message schedule of its compression function; this was done, according to the NSA, to correct a flaw in the original algorithm which reduced its cryptographic security. However, the NSA did not provide any further explanation or identify the flaw that was corrected. Weaknesses have subsequently been reported in both SHA-0 and SHA-1. SHA-1 appears to provide greater resistance to attacks. SHA-2 is a set of cryptographic hash functions designed by the NSA (U.S. National Security Agency).[3] SHA stands for Secure Hash Algorithm. Cryptographic hash functions are mathematical operations run on digital data; by comparing the computed "hash" (the output from execution of the algorithm) to a known and expected hash value, a person can determine the data's integrity. For example, computing the hash of a downloaded file and comparing the result to a previously published hash result can show whether the download has been modified or tampered with.[4] A key aspect of cryptographic hash functions is their collision resistance: nobody should be able to find two different input values that result in the same hash output.SHA-2 includes significant changes from its predecessor, SHA-1. The SHA-2 family consists of six hash functions with digests (hash values) that are 224, 256, 384 or 512 bits: SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256.SHA-256 and SHA-512 are novel hash functions computed with 32-bit and 64-bit words, respectively. They use different shift amounts and additive constants, but their structures are otherwise virtually identical, differing only in the number of rounds. SHA-224 and SHA-384 are simply truncated versions of the first two, computed with different initial values. SHA-512/224 and SHA-512/256 are also truncated versions of SHA-512, but the initial values are generated using the method described in FIPS PUB 180-4. SHA-2 was published in 2001 by the NIST as a U.S. federal standard (FIPS). The SHA-2 families of algorithms are patented in US 6829355. The United has released the patent under a royalty-free license. In 2005, an algorithm emerged for finding SHA-1 collisions in about 2000-times fewer steps than was previously thought possible.[6]Although (as of 2015) no example of a SHA-1 collision has been published yet, the security margin left by SHA-1 is weaker than intended, and its use is therefore no longer recommended for applications that depend on collision resistance, such as digital signatures. Although SHA-2 bears some similarity to the SHA-1 algorithm, these attacks have not been successfully extended to SHA-2. In October 2012, the National Institute of Standards and Technology (NIST) chose the Keccak algorithm as the new SHA-3 standard. Keccak offers many benefits, such as performance and good resistance traits. In this article, I take a concise look at Keccak's workings. I examine its engine and see how it renders the message text into a hash. In addition, I compare Keccak against SHA-1 and SHA-2 using four standard tests. A notable problem with SHA-1 and SHA-2 is that they both use the same engine, called Merkle-Damgard, to process message text. This means that a successful attack on SHA-1 becomes a potential threat on SHA-2.Consider SHA-1 for instance. A brute force attack usually takes at least 280 rounds (a round is a single cycle of transformation of the interim hash value) to find a collision in a full-round SHA-1. But in February 2005, Xiaoyun Wang and colleagues used a differential path attack to break a full-round SHA-1, and it took only 269 cycles to succeed. That same attack was later corroborated by Martin Cochran in August 2008.In 2012, Mark Stevens used a series of cloud servers to perform a differential path attack on SHA-1. His attack produced a near-collision after 258.5 cycles. He also

estimated a modified attack can manage a full-collision after 261 cycles. As to SHA-2, the only successful attacks were those against a limited round SHA-2 hash. The most effective attack was against a 46-round SHA-2 (512-bit variant) and against a 41-round SHA-2 (256-bit variant). It took 2253.6 cycles to break the 256-bit variant and 2511.5 cycles for the 512-bit variant. The fact remains that, while no successful attacks against a full-round SHA-2 have been announced, there is no doubt that attack mechanisms are being developed in private. This is one reason why NIST sponsored the SHA-3 competition, which led to the development and recent adoption of Keccak.

SHA3

To be considered for the SHA-3 standard, candidate hash functions had to meet four conditions set by NIST. If a candidate failed to meet these conditions, it was disqualified: The candidate hash function had to perform well regardless of implementation. It should expend minimal resources even when hashing large amounts of message text. Many proposed candidates were actually unable to meet this requirement. The candidate function had to be conservative about security. It should withstand known attacks, while maintaining a large safety factor. It should emit the same four hash sizes as SHA-2 (224-, 256-, 384-, or 512-bits wide), but be able to supply longer hash sizes if need be. The candidate function had to be subjected to cryptanalysis. Both source code and analytical results were made public for interested third-parties to review and comment. Any weaknesses found during analysis were to be addressed, through tweaks or through redesign. The candidate function had to exercise code diversity. It could not use the Merkle-Damgard engine to produce the message hash. The SHA-3 competition saw 51 candidate functions enter the first round of evaluations. Out of those, 14 managed to advance to the second round. Round three saw the candidates whittled down to five. And from those five, Keccak was declared the winner. Keccak is recognized as a new Secure Hash Algorithm-3 i.e. SHA-3 [3] announced by NIST. Gilles Van Assche, Guido Bertoni, Michael Peeters and Joan Daemen designed and proposed the construction of Keccak Hash function. The Keccak-f permutation is the basic component of Keccak Hash function and supports 224-bit, 256-bit, 384-bit and 512-bit hash variants. It consists of number of rounds and each round is the combination of logical operations and bit permutations. Keccak is generated from sponge function with Keccak [r, c] members. It is categorized by these additional functions i.e bit rate (r) and capacity (c). The addition of r + c gives width of the Keccak function permutation and is it is further limited to values as indicated 25, 50, 100, 200, 400, 800, 1600. The Keccak team introduced the Keccak [1600] function for SHA3 proposal with different values of 'r' and 'c'. Keccak [1600] was selected because of its increased number of rounds in order to provide improved security margin. For 256-bit hash value r = 1088 and c = 512. For 512-bit hash output, the values of r and c are 576 and 1024 respectively. The 1600-bit state matrix of Keccak composed of 5x5 matrixes of 64-bit words. Initially, the message block should undergo the inversion procedure so that last byte should come first and first byte should become last. Every single compression function of Keccak composed of 24 rounds and each round is sub-divided into five steps i.e. Theta (Θ), Rho (ρ) and Pi (π), Chi (χ), Iota (i) explained in below section

Theta (Θ) Step

Theta function comprises of three equations that involves simple XOR and bitwise cyclic shift operations. Equation (1) involves the XOR operation between lanes (set composed of 64-bits along the constant x and y co-ordinates) of each row of the state matrix A that results in five output lanes. Initially left circular shift will be applied on the five output lanes in such a way that last lane becomes first and second last lane becomes last lane in (2). After that right circular shift will be

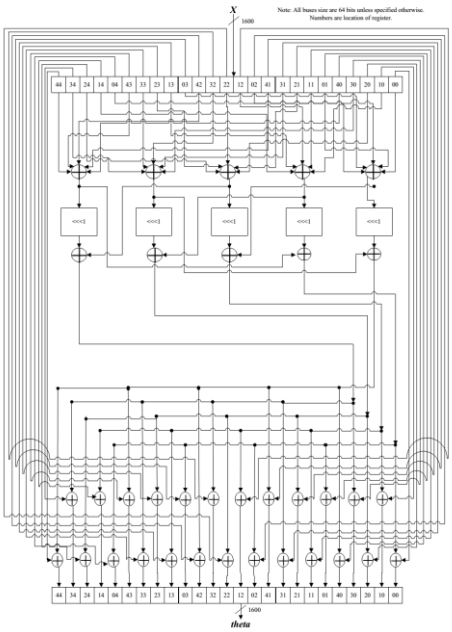
carried out on the lanes so that first lane becomes the last and second lane becomes the first lane and then left circular shift will be applied on each lane in order to change the positions of the bits within each lane. Equation (3) of Theta just involves XORing between the input state matrix and output lanes obtained from (2).

Theta step consists of three main steps in terms of equations that mainly require bitwise XOR operation. Equation (1) involves bitwise XOR operation between the 64-bit lanes of each row where every lane of each row is independent of each other so parallel operations can be applied on these lanes. We have used conventional 64-bit XOR operator in parallel to perform XORing between the five lanes in each row of the state array 'A' and results are stored in intermediate registers. The above parallel XOR operations make our design fast and more efficient in terms of performance. Second step (2) of step theta involves one bit left circular rotation which is accompanied by simple rewiring or replacing the bit pattern of each row, then XORed with the previous output lanes. The results are stored in an intermediate registers in the form of five lanes. These lanes are again XORed with input state matrix A[x, y] to form new 5 x 5 state matrix A'[x,y]. All the operations are done on modulo 5.

$$C[X] = A[X,0] \oplus A[X,1] \oplus A[X,2] \oplus A[X,3] \oplus A[X,4] \quad 0 \le X \le 4 \qquad (1)$$

$$D[X] = C[X-1] \oplus ROT(C[X+1,1]) \quad 0 \le X \le 4 \qquad (2)$$

$$A[X, Y]' = A[X, Y] \oplus D[X] \quad 0 \le X, Y \le 4 \qquad (3)$$



RHO (P) AND PI (II) STEP

The next two steps Rho (ρ) and Pi (π) can be expressed jointly by (4) that compute an auxiliary 5 x 5 array B from the state array 'A'. The operation of Rho (ρ) and Pi (π) take each of the 25 lanes of the state array 'A', perform circular rotation on it by the fixed number of values depending upon the

'x' and 'y' co-ordinates i.e r[x, y] given in Table I [3] (called Rho (ρ) step) and then place the above rotated lanes at the different location in the new array B (called Pi (π) step). Note that all the indices are taken modulo 5.

Table1. *The Cycle Shift Offsets "R(X,Y)" For Keccak*

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

$$B[Y,2X+3Y] = ROT(A[X, Y], r[X, Y]) \quad 0 \le X, Y \le 4 \qquad (4)$$

Chi (χ) Step

The Chi (χ) step operates on the lanes, i.e. words with 64-bits and manipulates the B array obtained in the previous Rho (ρ) and Pi (π) step and replaces the result in the state array A. We can say that the Chi (χ) step takes the lane at location [x,y] and XOR it with the logical AND of the lane at address location of [x+1,y] and the complement at location [x+2,y]. Following equation is illustrating the function Chi (χ).

$$A[X,Y] = B[X,Y] \oplus ((\text{NOT } B[X+1,Y]) \text{ AND } B[X+2,Y]) \quad 0 \le X, Y \le 4 \qquad (5)$$

Iota (i) Step

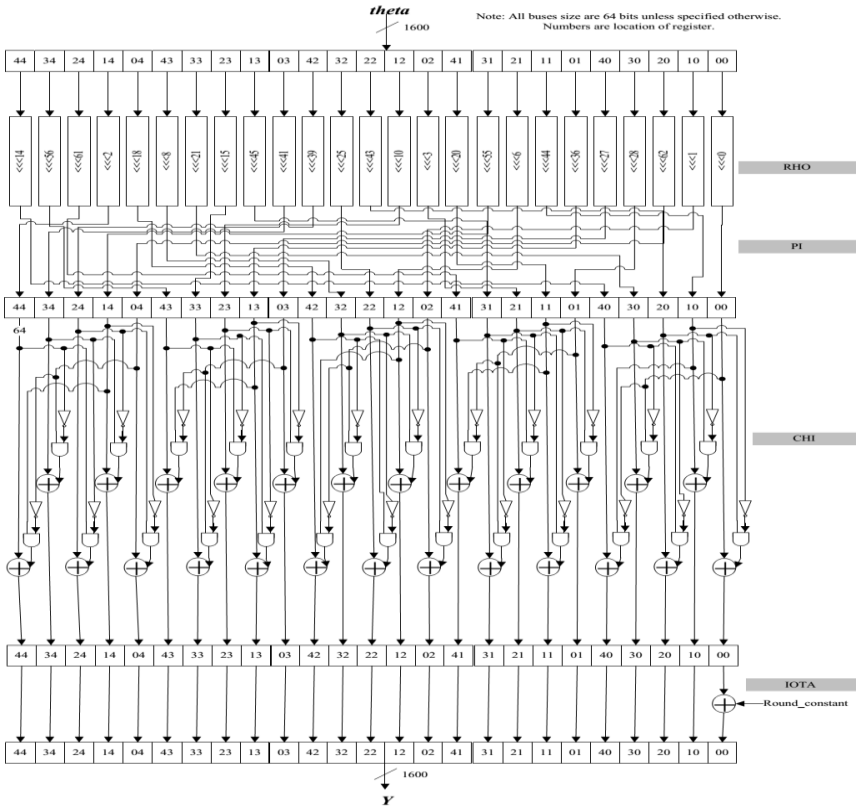
The Iota step is the simplest step of Keccak algorithm. It just performs the XOR operation of predefined 64-bit constant RC given in [3] with the lane at location [0,0] of the new state matrix 'A'.

$$A[0,0] = A[0,0] \oplus RC \qquad (6)$$

In this work, we present an iterative design of SHA-3 512-bit for compact implementation as shown in Fig. The architecture has 128-bit input data just to save extra input bits. The next block in proposed design is padder block which pads the required number of zeros with the input data in order to form 1600-bit state and then inversion is applied on each byte. The output from the padder block is forwarded to 2 x 1 Multiplexer (MUX) which drives the output data from padder to the compression-box of the architecture and selects the input data for the first round and feedback data for other twenty three rounds of Keccak with the help of controlling signal (Ctrl 1).

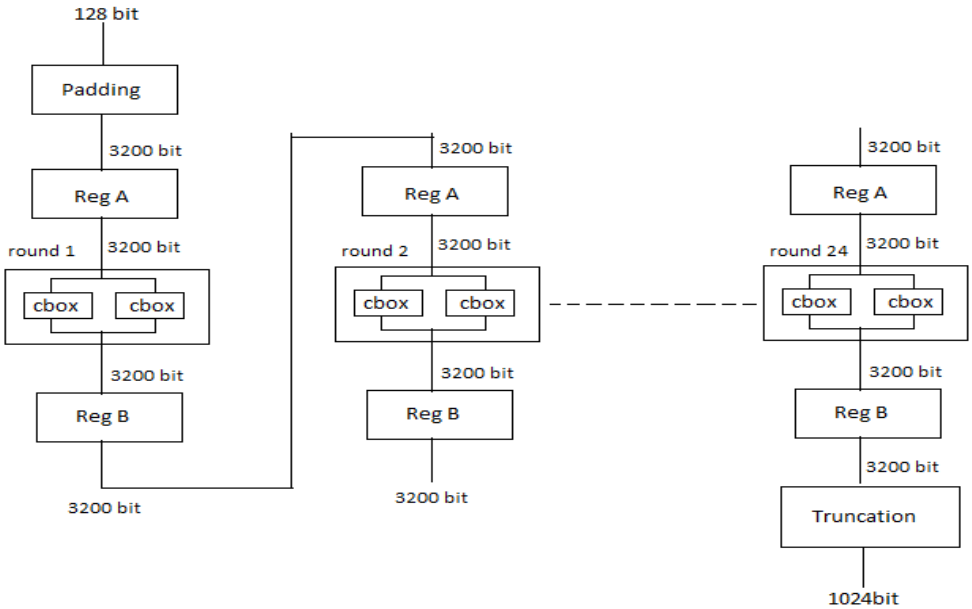
When Ctrl 1 is low, MUX select the input data and at high, MUX will select the feedback data. First padded message is directly copied to Reg A which previously initialized with all zeroes and

resulting bits are forward to Compression-Box (C-Box). It is basically the implementation of compression function in SHA-3 algorithm which comprises of theta (Θ), rho (ρ), pi (π), chi (χ) and iota (i) step. For performance, we logically optimized our design by implementing rho (ρ), pi (π) and chi (χ) steps as a single step.

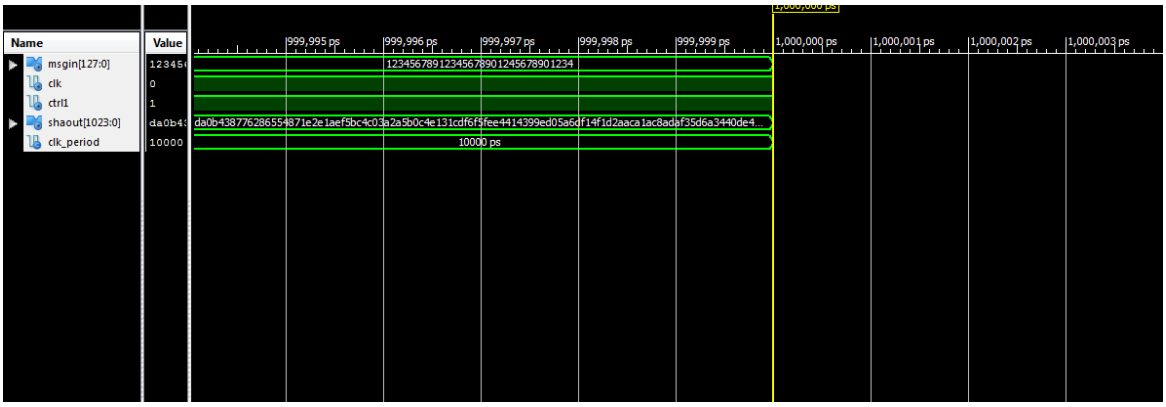


This results in saving of hardware resources in term of 48 slices. After completing 48 iterations, final output is forwarded to Reg B for storage in order to synchronize the data-path. The last component in the architecture is Truncating component where inversion per byte is performed on the output bits and then truncated to the desired length of hash output.

128BIT KECCAK SEQUENTIAL ARCHITECTURE



SIMULATION WAVEFORMS



CONCLUSION

The SHA 3 algorithm provides a good security for the data using the authentication format by generating hash code. Our logical optimization by merging the three transforms i.e. rho, pi and chi in to single transform and by exploring maximum parallelism in the algorithm are contributing factor And the whole design has a simple hardware structure and fast running speed and can be widely used in digital signatures and 3DES key generation systems.

REFERENCES

[1] X. Wang, D. Feng, X. Lai, and H. Yu, "Collisions for hash functions md4, md5, haval-128 and ripemd," IACR, August 2004.

[2] National Institute of Standards and Technology (nist), "Cryptographic hash algorithm competition," 2007.

[3] FIPS-202, "Federal information processing standards publication fips-202, secure hash algorithm-3 (sha-3)," 2014.

[4] Xilinx, "Virtex 2.5 V field programmable gate arrays". [5] F.R. Henrquez, A.D. Prez, N.A. Saqib, and C.K. Koc, Cryptographic Algorithms on Reconfigurable Hardware. Signals and Communication Technology, Springer, 2007.

[5] S. Kerckhof, F. Durvaux, N.V. Charvillon, F. Regazzoni, G.M. de Dormale, and F.X. Standaert, "compact fpga implementations of the five sha-3 finalists," Springer Berlin Heidelberg, vol. 7079, pp. 217–233, 2011.

[6] A. Akin, A. Aysu, O.C. Ulusel, E. Savas, "Efficient hardware implementations of high throughput sha-3 candidates keccak, luffa and blue mid night wish for single- and multi-message hashing," ACM, pp. 168–177, 2010.

[7] K. Gaj, E. Homsirikamol, and M. Rogawski, "Comprehensive comparison of hardware performance of fourteen round 2 sha-3 candidates with 512-bit outputs using field programmable gate arrays," 2ndSHA-3 Candidate Conference, pp 23-24, August 2010.

[8] B. Baldwin, N. Hanley, M. Hamilton, L. Lu, A. Byrne, M. ONeill, and W.P. Marnane, "FPGA implementations of the round two sha-3 candidates," The second SHA-3 Candidate Conference, 2010.

AUTHORS' BIOGRAPHY

Avinash Kumar, has completed his B.Tech Degree in 2010 in Electronics and communication engineering from Cambridge Institute of Technology, Ranchi, Jharkhand and pursuing M.Tech in VLSI SYSTEM DESIGN (VLSISD) from GONNA INSTITUTE OF INFORMATION TECHNOLOGY AND SCIENCES, Vishakhapatnam, and Andhra Pradesh. His current area of research is on cryptography.

C.H Pushpalatha, has completed her B.Tech in 2011 Electronics and communication engineering from NIE Guntur and completed her M.Tech in AVANTHI INSTITUTE OF ENGINEERING AND TECHNOLOGY AND SCIENCES, Vishakhapatnam. Now working as an associate professor in GONNA INSTITUTE OF TECHNOLOGY Vishakhapatnam.