Implementation of 48-Round Keccak for SHA3-1024 in Xilinx against Collisions

Koduri.Nireesha
PG Scholar, Dept. of E.C.E.
University College of Engineering, JNTUK,
Kakinada, India-533003

E.V.Narayana
Assistant Professor, Dept. of E.C.E.,
University College of Engineering, JNTUK,
Kakinada, India-533003

Abstract. In this paper we present a method of Secure Hash Algorithm3-1024 on Xilinx by implementing 48-round Keccak. Merkle-Damgard construction used in SHA-0 (collisions were found), SHA-1, SHA-2(version overcoming drawbacks of SHA-1) is replaced by Sponge Construction in SHA-3, by increasing the number of executable rounds in Sponge Construction the internal permutation of Keccak function is even more increased .Comparing to the results of previous designs our design shows the best security. As number of output bits and internal permutations are increased, it will be too difficult for the opponent to try for cryptanalysis. As no two messages can produce same hash code our Algorithm provides better authentication.

Keywords: Cryptography, SHA3, Xilinx, Keccak, Cryptanalysis.

I INTRODUCTION

Cryptography is a technique of using mathematics of encrypting and decrypting the data. The over arching goal of cryptography is to enable people to communicate privately over an insecure channel in the presence of adversaries. It is a known fact that encryption is a commonly used tool for providing confidentiality. Authentication provides data integrity, and also helps to ensure whether any damage with or corruption of data is detected. It also provides guaranty of original message .Authenticated encryption (AE) algorithms adds both confidentiality along with integrity/authenticity by processing plaintext and producing both cipher text and a Message Authentication Code (MAC).

The ideal cryptographic hash function has four common properties:

1. It is easy to define the hash value for any given message
2. It is impossible to generate a message from its hash
3. It is impossible to modify a message without changing the hash
4. It is unachievable to find two unequal messages with the same hash.

Most of the cryptographic hash functions are designed to take input string of any length and produce a fixed-length output hash value.

A cryptographic hash function is the function that must be able to tolerate all known types of cryptanalytic attack. In order to overcome from attacks it must have the properties like:

Pre-image resistance:

If a hash value h is given then it should be difficult to find any message m such a way that h = hash(m). This is one-way function. Functions that loss this property are sensitive to pre-image attacks.

Second pre-image resistance:

If an input m_1 is given then it should be difficult to find different input m_2 such that $hash(m_1)=hash(m_2)$. Functions that does not obey this property are not resistive to second pre-image attacks.

ISSN: 2278 – 7798

Collision resistance:

It should be impossible to find two different messages m_1 and m_2 such a way that hash(m_1) = hash(m_2). So hash value at least twice than that of required pre-image resistance. [3]

There are many types of hash functions, Since 1990 the MD family of hash functions and its successor SHA family have been most widely used data integrity primitives. In contrast with few cryptanalytic results in 90s recent attacks on MD5 , SHA-0, SHA-1, encouraged the cryptographic community to look for more reliable components and then motivated the recent SHA-3 competition [4][3].

On October 2-nd 2012 NIST made clear about taking up the Keccak scheme as the new SHA-3 hash standard. Keccak replaces Merkle-Damgard Construction of previous hash algorithms by Sponge Construction[4].

This paper is organized as follows Keccak, Implementation of 48-round Keccak, Applications of SHA3-1024, finally conclusion.

II KECCAK

In this section we present a description of Keccak. Keccak uses the sponge construction and hence is a Keccak member of the sponge function family as it uses the sponge construction for generating the hash code[4]. Figure 1 shows the Sponge construction. It can be used as a hash function and it also can be applied for generating infinite bit stream, making it applicable for both stream cipher and also as a pseudorandom bit generator. In this paper we target on the sponge construction for cryptographic hashing. Commonly Keccak function has two main parameters bitrate and capacity, which are denoted by r and c respectively. The sum of bitrate and capacity makes the state size, on which Keccak operates. For our proposed SHA-3,we consider the state size as 1600 bits. Giving different values for bitrate and capacity provides the trade-off between speed and security. If the bitrate is high gives the faster function that gives less security. Keccak follows the sponge two-phase processing[5].

Initially 1600-bit state is filled with 0's, In the first phase .This phase is also called as the absorbing phase. First phase is interleaved with applications of the permutation f,called Keccak-f in the specification. When all message blocks are processed the absorbing phase is completed. In the second phase which is also called the squeezing phase, the first r bits of the state are returned as the output bits, and interlaced with applications of the function f. The squeezing phase is finished after the required length of output digest has been originated.

SHA-3 candidates proposed and the hash length multiplied by 2 is taken as value of parameter c. For example, the SHA-3 with 1024-bit hash length is Keccak with c =1024bits and r = 576bits (r + c = 1600). In this paper we intend variants proposed as SHA-3 candidates by Keccak-512bit, Keccak-384bit, Keccak-256bit, and Keccak-224bit. (The number is output hash length.)

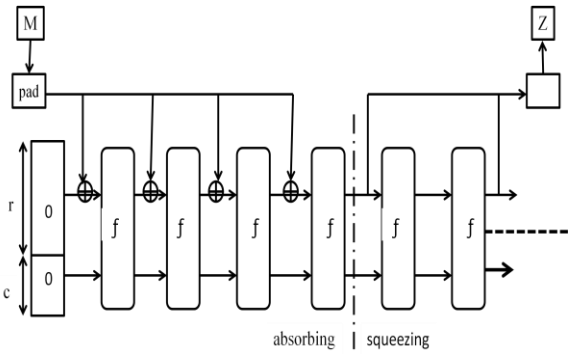


Fig1 Sponge construction

The submitted Keccak functions for the SHA-3 competition have bitrate as 1600 and c = 2n, where n ε {224; 256; 384; 512}. The 1600-bit state can be stated as an array of 3-dimensional bits, a[5][5][64], and each state bit consists of 3 integer coordinates

a[x][y][z], where x and y are taken as modulo 5, and z is taken as modulo 64.

Each and every round of the Keccak permutation consists of five mappings

$$R = i \circ X \circ \pi \circ \rho \circ \theta.$$

DESIGN OF SPONGE CONSTRUCTION

SHA3-1024 uses the sponge construction which contains a subset of XORed message blocks in the state, which is then transformed completely. In the version used in SHA-3, the state consists of 5x5 array of 64 bit words,1600bits total. On an Intel core Keccak

ISSN: 2278 – 7798

claimed 12.5 cycles per byte. However, in hardware implementations, it is faster than all other finalists.[5]

THE BLOCK PERMUTATION

The Keccak permutation consists of 48 rounds, which operate on the 1600 state bits. Keccak uses the conventions which are presented below, and also helpful in deriving its round function:

A. Theta(θ)Step :

Theta is a linear map which adds each bit in a column. It consists of three equations that involves XOR and bitwise cyclic shift operations. Equation (1) .Lanes of each row of the state matrix A are Xored in equation(1), that results in five output lanes the set consists of 64-bits along the constant x and y co-ordinates.

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

Step1: On the five output lanes the left circular shift will be applied in such a way that last lane becomes first and last but one lane becomes last lane in (2).

Step2: The right circular shift will be carried out on the lanes so that first lane replaces the last lane and last but one becomes the first lane and then in order to change the positions of the bits within each lane the left circular shift will be applied on each lane

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

XORing between input state matrix and output lanes obtained from (2). Is given by Equation(3)

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

B. Rho (ρ) and Pi (π) step

The next two steps Rho (ρ) and Pi (π) can be combined and expressed 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). In the new array B place the above rotated lanes at the different locations (called Pi (π) step). Note that all the indices are taken modulo 5.

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

	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

Table1. THE CYCLIC SHIFT OFFSETS R(X,Y) FOR KECCAK

C. Chi (ϰ) 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. It takes the lane at location [x,y] and logical AND of the lane at address location of [x+1,y] are XORed and the complement at location [x+2,y].Following equation is illustrating the function Chi (χ).

ϰ is only non-linear mapping of keccak, working

$$A[X,Y]=B[X,Y]\oplus((NOTB[X+1,Y]] \text{ AND } B[X+2,Y] \quad 0\leq X, Y\leq 4 \quad (5)$$

D. Iota(i) Step

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

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

The operation of Keccak hash function divided in three stages. First initialization proceeded by absorbing and finally squeezing stage as shown in Fig. 1, where the message is denoted by M and the hash output is denoted by Z. Every single bit of state matrix 'A' is initialized to zero during initialization phase. In 2nd stage, absorbing stage, recent matrix state is XORed with all r-bit blocks ,in this way Keccak permutation for 48 rounds are accomplished sequentially. Finally, in squeezing stage, basically truncation of matrix state to required length of output

ISSN: 2278 – 7798

hash is done and from current state matrix extra bits are truncated to achieve desired hash length. If wanted hash value is more than r-bit(bit-rate), then additional Keccak permutations are actively performed and further their results are awaited until hash width obtains its desired length.

III IMPLEMENTING 48-ROUND KECCAK:

In this paper, we present an iterative design of SHA-3 1024-bit for implementation as shown in Fig. 2. Just to save extra input bits the architecture has 128-bit input data. The padding operation is done by next block i.e., padded block , with the input data In order to form 1600-bit state required number of zeros are added and then on each byte inversion is applied.

The output from the padded block is given to 2x1 Multiplexer (MUX) which sends the output data from padded to the compression-box and selects the input data for the first round and feedback data with the help of controlling signal (Ctrl 1) for other forty seven rounds of Keccak. If Ctrl 1 is low, MUX select the input data and If Ctrl 1 is high, MUX will select the feedback data.

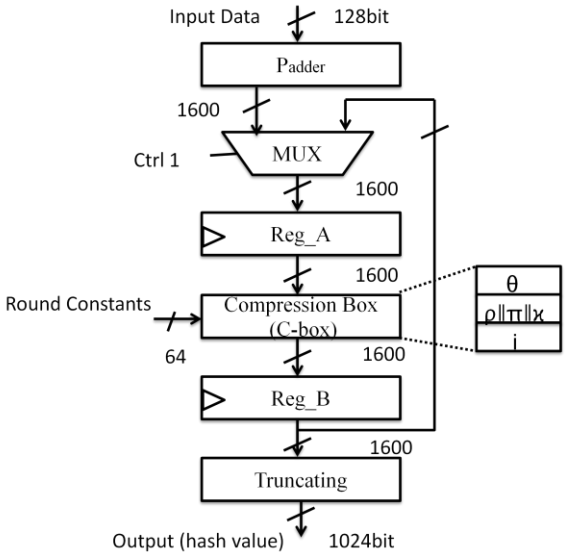


Fig2. 128 bit keccak sequential architecture

First Reg A stores the padded message which is previously placed with all zeroes and that bits are forward to Compression-Box (C-Box) .The implementation of compression function in SHA-3 algorithm consists of theta (Θ), rho (ρ), pi (π), chi (χ) and iota (i) step. We logically enhance our design by combine implementing rho (ρ), pi (π) and chi (χ) steps. After completing 48 iterations, Reg B stores the final output in order to synchronize the data-path. Truncating component is the last component in the architecture is where the output bits are per byte inverted and then after truncating desired length of hash output can be obtained.

Implementation of Compression-Box:

The details of steps for implementing Compression is given as follows

1) Theta Step (Θ):

As Theta step consists of three main steps that mainly require bitwise XOR operation in terms of equations. Bitwise XOR operation between the 64-bit lanes of each row parallel operation can be applied on these lanes , here every lane of each row is independent of each other given by Equation 1. We have used parallel 64-bit conventional XOR operator to perform XORing between the five lanes in each row of the state array 'A' and results are stored in intermediate registers. Our design is made fast and more efficient because of parallel XOR operation in terms of performance. Second step of theta involves one bit left circular rotation which is followed by simple rewriting the bit pattern of each row, and then XORed with the previous output lanes. Intermediate registers stores the results in the form of five lanes. Input state matrix A[x,y] are again XORed with register stored lanes to form new 5 x 5 state matrix A[x,y]. All the operations are done on modulo 5.

2) Integrated Rho (ρ), Pi (π) and Chi (χ) Step:

The rho and pi are basic permutations and each lane requires cyclic shift by some fixed numbers ,If this step is implemented separately, Extra 48 slices for the operation of rho and pi step would be needed. In order to save these slices we have logically reduced our design by merging the rho and pi step into the chi step. Manually all calculations required in rho and pi step are performed and applied cyclic shift on each lane of state matrix obtained at the output of theta step and change them after calculating its new position according to (4).

3) Iota (i):

ISSN: 2278 – 7798

In Iota (i) step, To perform XORing between the 64-bits(least significant) of state array and round constant RC we have used conventional 64-bit XOR operator. For every round the values of round constant are fixed and different. These round constants are stored in registers[1].

IV SHA-3 PARAMETERS:

Message digest size	224	256	384	512	1024
Message size	No limit	No limit	NO limit	No limit	No limit
Block size (Bitrate r)	1152	1088	832	576	576
Word size	64	64	64	64	64
Number of Rounds	24	24	24	24	48
Capacity c	448	512	768	1024	1024
Collision Resistance	2^{112}	2^{128}	2^{192}	2^{256}	2^{512}
Second Pre-image Resistance	2^{224}	2^{256}	2^{384}	2^{512}	2^{1024}

Table2:Sizes of SHA-3 and Security levels

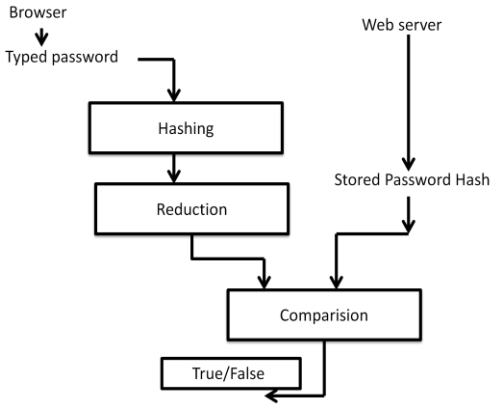
Secure Hash Algorithm takes different sizes of input and produces fixed message digest, bitrate and capacity are two inputs given to sponge construction. Our algorithm implements 1024 message digest by maintaining bitrate 576 and capacity 1024.

V APPLICATIONS:

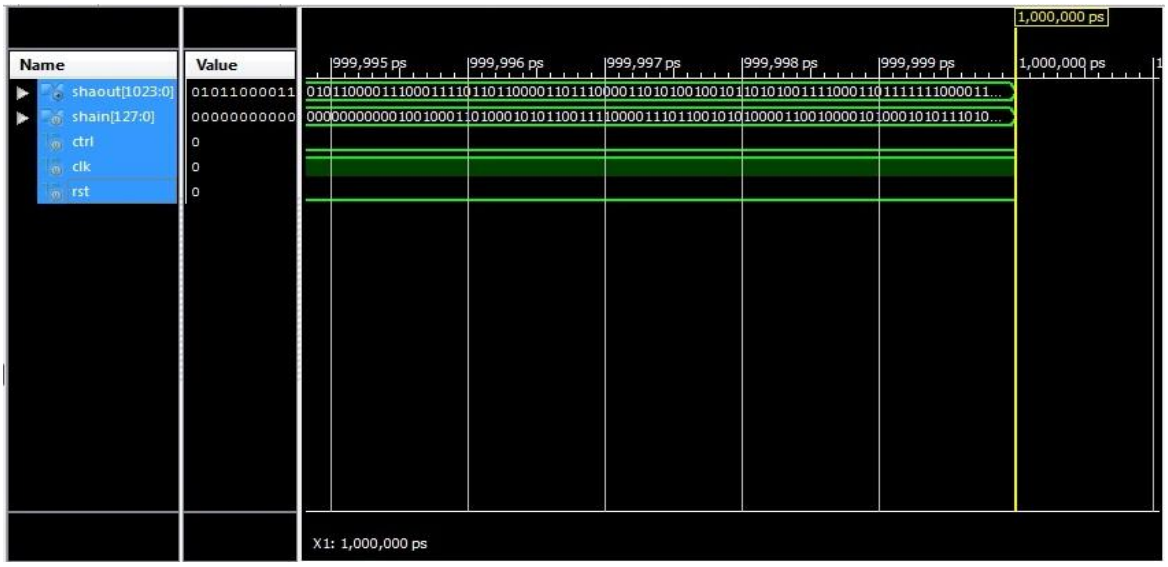
- E-Learning
- Web servers
- Online Banking
- Digital Signatures
- Online Shopping

wherever you need to log yourself in to access data wherever relevant data's are handled secure hashing is used .

- Password Verification



VI SIMULATION RESULTS:



ISSN: 2278 – 7798

VII CONCLUSION:

In order to know the unknown message for the known hash code, opponent should have capability of breaking the properties of hash algorithm

a) Pre-image and Second Pre-image attacks:

Finding a value y such that H(y) is equal to given h.

For m-bit hash level of effort is 2^m i.e., we require 2^m attempts for getting the value of y

To know the message for a known hash value of 1024 bit, an opponent should try 2^{1024} messages which is highly impossible.

b) Collision Resistant Attacks:

Finding two messages or data blocks x and y having same hash value i.e., H(x)=H(y), For m-bit hash it requires $2^{m/2}$ attempt

EX: Birthday attack

It requires $2^{1024/2}$ attempts to verify whether two messages are collision resistant or not, which is too difficult to perform

Since 2012 Researchers had been trying to find cryptanalysis for SHA3-512, they succeeded for attacking only 3 rounds. It will take even more years to find cryptanalysis for SHA3-1024 which is implemented in our paper.

SHA3 is made standard hash algorithm by NIST in August 2015 because of its strong cryptanalytic property.

VIII FUTURE SCOPE:

Our paper can be extended for future by implementing hardware for secure hash algorithm (SHA) hash functions. significantly decreasing the critical path and required area. on a Xilinx VIRTEX II Pro. [6].

IX ACKNOWLEDGMENT:

I wish to express my deep sense of gratitude and respect to my project guide, Mr.E.V.Narayana , Assistant Professor, Dept. of ECE, University College of Engineering, Kakinada, who has persistently and determinedly guided me all throughout my project.

X REFFERENCES:

1. Alia Arshad, Dur-e-Shahwar kundi, Arshad Aziz.,Compact Implementation of SHA3-512 on FPGA, IEEE Conference on Information Assurance and Cyber Security (CIACS) June 2014 .
2. National Institute of Standards and Technology (NIST), "Cryptographic Hash Algorithm Competition," 2007.
3. Whitfield Diffie,. Hellman, Martin.E .Member , IEEE New Directions In Cryptography Information Theory , IEEE Transactions on Information Theory.Vol:22 Issue:6 Pages: 644-654 , 06 Jan 2003
4. R. Merle Member, IEEE, Secure Commucation Over Insecure Channel., Submitted to Comm of ACM.
5. Borawski,M; Crptology Div.,Mil, Commun,Inst., Zegrze,Poland., The Sponge Construction as a source of Secure Cryptographic Primitives, Millatary Communications and Information Systems Conference(MCC), Oct 2013.
6. Ricardo Chaves, Student Member ,IEEE, Georgi Kuzmanov, Member, IEEE, Leonel Sousa, Seniour Member, IEEE, and Stamatis Vassiliadis, Fellow, IEEE., Cost-Efficient SHA Hardware Accelerators .,IEEE Transactions on very large scale integration(VLSI) Systems ,VOL.16., NO.8 August 2008.

KODURI.NIREESHA received the B.Tech degree in Electrical and Electronics Engineering in 2012 from Pace Institute Of Technology And Sciences, Ongole , and currently pursuing M.Tech in Instrumentation And Control Systems In University College Of Engineering, Kakinada.

E.V.NARAYANA received B.E. and M.E. degrees from College of Engineering, Andhra University, Visakhapatnam. He started his career as lecturer in 1985. Now he is working as asst. Professor in the department of Electronics and Communication Engineering, University College of Engineering, JNTUK, Kakinada. He published 63 papers in national & international conferences and journals.

ISSN: 2278 – 7798