1. INTRODUCTION

Information has been always very important meta of the people's development process and guarantee that it will be. People from the old ages, by the transmission of information only want to the information to be seen by the authorized people. Therefore people have been using primitive techniques after writing has invented .The first example of crypto-primitives, which known today is Caesar's cipher. [1]

From Caesar's cipher to present day, in almost all the wars that people faced, one of the most important plane that people have to win is to learn what enemy say and not to let them know what they say. As we know, in the 2nd World war, after Enigma encryption was broken, destiny of the war changed to Allied 's side. Many ships and submarines of Germans were destroyed because of they had been decrypted. For this thesis, a new algorithm of cryptography, a new not broken, secure algorithm is analyzed and studied on the hardware implementation of the algorithm on an FPGA by using VHDL.

Today we access many services that accessible to public. Our banking, mailing, and telephoning are in a platform that everybody can access. Here is the claim, how can everybody access only their information no more? It is today provided with the help of cryptology. People have accounts and their account keys that make only them who have the key available to access into the account. But are these codes are non-breakable? Of course not .These codes can be broken, but what here should be done is to find out new algorithms.

NIST for that reason opened a competition, to choose most secure, most applicable algorithm; that would be used for the following year's applications. This thesis is mainly based on one of that competition's candidate Keccak Hash Function. First, **it** is mentioned of a general information and history of cryptography. After that, information given about hash functions which converts a large, possibly variable-sized amount of data into a small datum, usually a single integer that may serve as an index into an array.

After hash function information, some information is given about the NIST competition that mentioned above. As being a candidate Keccak Hash function's specifications are detailed in this thesis. Round function and Keccak sponge function are mathematically expressed. Information about how they implemented is given. As being a candidate that exceeded to first round, Keccak Hash function is one of the important applicant to be widely used in future electronic applications. For that reason it is important for preparing this thesis and designing a hardware that could shed a light for the following thesis and projects.

2. CRYPTOGRAPHY

2.2. General Definition and Background

The word cryptography is derived from the Greek words kr`yptus (hidden) and gráphein (to write). Cryptography, or secret writing, has been used to avoid people to access the secrets from time people began to write. [1]

Cipher A cipher is a cryptographic algorithm that is used to convert readable text (referred to as plaintext) into an encoded form (referred to as cipher text). For a cipher to be useful, there must be a way to convert backwards from cipher text to the original plaintext.

One of the most famed of the ancient practitioners of cryptography was Julius Caesar. He invented a cipher, (Caesar cipher) which is based on a permanent replacement of the letters of the alphabet. Figure 2.1 shows the process of the Caesar cipher. It maps each letter of the alphabet to another letter that is a fixed number of letters (the rotation amount) away. For example, with a rotation of 1, A is mapped to B, B to C, …, Y to Z, and Z to A. (Letters at the end mapped to the beginning of the alphabet.) With a rotation of 12, A is mapped to M, B is mapped to N, …, Y to K, and Z to L.

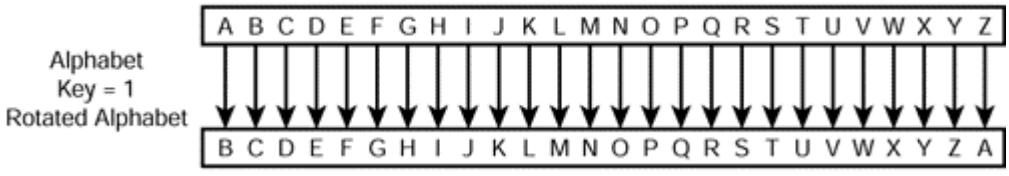

Figure 2.1 Cesar Ciphers[1]

Cryptography furthermore used in Europe, the Middle East, and North Africa during the Middle Ages. Many of the ciphers were substitution ciphers. Refer to Figure 2.2 a substitution cipher is a cipher in which cipher text characters are substituted for plaintext characters. In a simple substitution cipher, a single cipher text character is substituted for a single plaintext character. In a polyalphabetic substitution cipher, multiple cipher text characters are substituted for groups of plaintext characters.

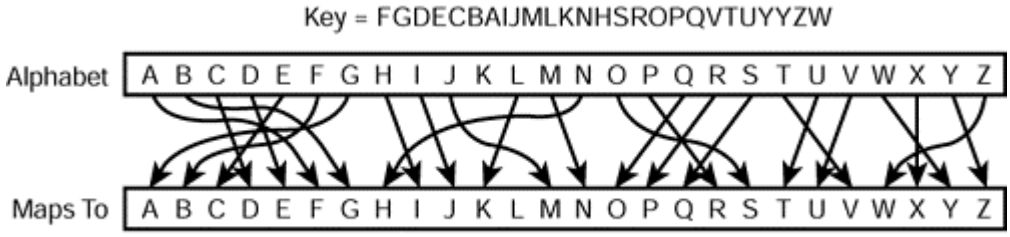

Figure 2.2 Substitution Ciphers.[1]

Polyalphabetic substitution ciphers were invented in the 15th century by Leon Battista Alberti. They remained popular until the 1800s, when they began to be broken. Cryptography was also used in both North and South Civil War and in every big war since then. During World War I, British cryptographers captured and decoded a message from the German Foreign Minister Arthur Zimmermann to the German Minister to Mexico that offered Mexico United States' land in exchange for support to Germany. The big effect of the deciphered message on American civic opinion resulted by the US's decision to join the war.

During World War II, most German codes were mostly based on the Enigma machine. A British cryptanalysis group first broke the Enigma code early in World War II. Some of the first uses of computers were for decoding Enigma ciphers intercepted from the Germans. Decryption of the Enigma machine changed the destiny of the World War II. Their work helped the demolition of a large number of German U-boats, the sinking of the Bismark, and important German losses in military operations, such as the invasion of Crete. [1]

Cryptographers, by 1948 started to use advanced mathematical techniques to calculate ciphers and to avoid computers from unscrambling the ciphers. Symmetric and

asymmetric key algorithms were developed to this end. A symmetric key algorithm uses the same key to encrypt and decrypt a message, whereas an asymmetric key algorithm uses two different keys. Key algorithms are covered in more detail in the next section.[2]

2.2 Cryptographic goals

Of all the information security objectives, the following four form a framework upon which the others will be derived: (1) privacy or confidentiality (2) data integrity (3) authentication and (4) non-repudiation. [3]

1. Confidentiality is a service for the content of the information from all but the certified to access it. Secrecy is a term synonymous with confidentiality and privacy. There are many approaches to contribute confidentiality, from physical protection to mathematical algorithms, what gives the data incomprehensible.

2. Data integrity is a service to the unauthorized modification of data. To safeguard the data integrity, one must have the ability to detect data manipulation by uncertified parties. Data manipulation comprises such things as insertion, deletion and Substitution.

3. Authentication is a service, in connection with identification. This feature applies to both entities and information. Two ingredient, entering into a communication should determine each other. Information, transmitted via a channel should be certified by origin, date of origin, data ingredient, time sent, etc. For these reasons, this aspect of cryptography is usually divided into two broad categories: entity Authentication and Data Origin authentication. Data origin authentication implicitly provides data integrity (if a message is changed, the source has changed).

4. Non-repudiation is a service that prevents a unit from denying early incurrence or proceedings. when the dispute go up due to an entity deny that certain measures that means; to resolve the situation is necessary. For example, one unit can certify acquisition of property by another unit and later deny any such certification was granted. A procedure which includes a trusted third party is required to the dispute. [3]

2.3 Cryptographic Primitives

A main aspire of cryptography is to sufficiently take in hand these four areas in both theory and practice. Cryptography is about avoid and recognition of cheating and other malicious activities. Figure 2.3 presents a representation of the primitives considered and how they relate.

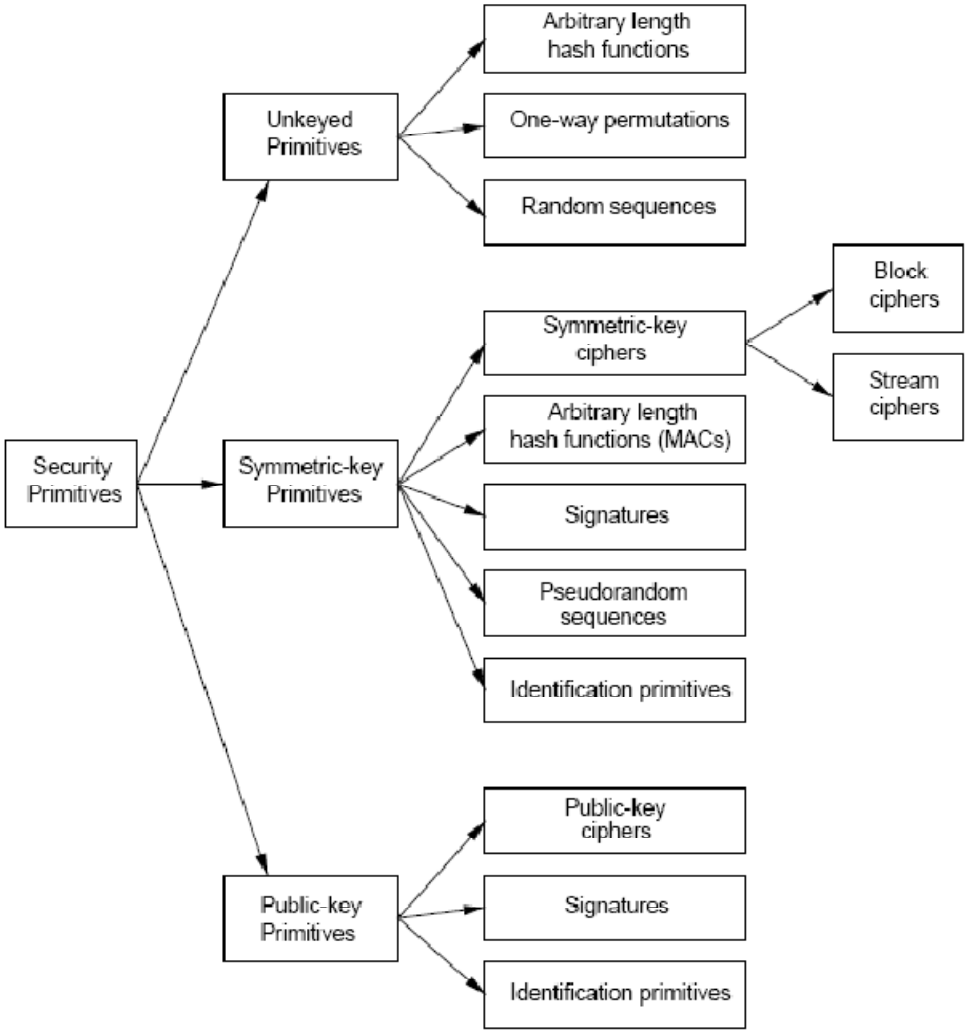

Figure 2.3 Cryptographic Primitives [2]

These primitives should be evaluated with respect to various criteria such as:

These primitives should be estimated with related criteria's;

1. Level of security: Usually difficult to measure. Presented in terms of operation number that necessary (using the best methods currently known) to beat the planned objective. Characteristically the rank of security is characterized by an upper bound on the sum of effort required to beat the objective. That called as the work factor.

2. Functionality: Primitives should be combined to meet various information security

objectives with the basic properties of the primitives, it will be determined that which primitives are most effective for a specified purpose.

3. Methods of operation: Primitives, when applied in different ways and with different inputs, will typically demonstrate various characteristics; therefore, one primitive could give very different functionality with respect of its function.

4. Performance: Is efficiency of a primitive in a particular operation.(For example, an encryption algorithm may be rated by the number of bits per second which can be encrypted.)

5. Ease of implementation: Is the complexity of realizing the primitive in a practical instantiation. This might comprise the difficulty of implementing the primitive in each a software or hardware .

3. HASH FUNCTIONS

Hash functions are recurring tools in cryptosystems just like the symmetric block ciphers. They are highly flexible primitives that can be used to obtain privacy, integrity and authenticity.

A hash function (formally known as a pseudo random function or PRF) maps an arbitrary sized input to a fixed size output through a process known as compression. This form of compression is not a typical data compression (as we know from a .zip file), but a noninvertible mapping. Loosely speaking, checksum algorithms are forms of "hash functions," and in many independent circles they are called just that. For example, mapping inputs to hash buckets is a simple way of storing arbitrary data that is efficiently searchable. In the crypto-graphic sense, hash functions must have two properties to be useful: they must be one-way and must be collision resistant. [4]

Being one-way means that the output of a given hash function, learning anything useful about the input is nontrivial. This is an important property for a hash, since they are often used in combination with RNG seed data and user passwords. Most small checksums are not one-way, as they are linear functions. Assuming the input from the output is often a simple computation, for short enough inputs.

Being collision resistant implies that an output given from the hash, finding another input that produces the same output (called a collision) is nontrivial. There are two forms of collision resistance that required from a useful hash function. Pre-image collision resistance (Figure 3.1) states that given an output Y, finding another input M' such that the hash of M' equals Y is nontrivial. This is an important property for digital signatures so that they apply their signature to the hash only. An attacker could substitute one signed message for another message, if collisions of this form were easy to find,. Second pre-image collision resistance (Figure 3.2) states that finding two messages M1 (given) and M2 (chosen at random), whose hatches match is nontrivial. [4]

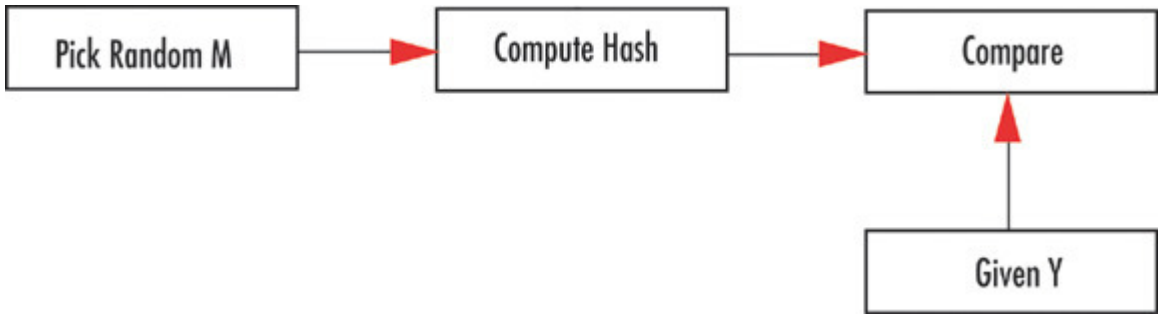

Figure 3.1 Pre-Image Collision Resistance [4]

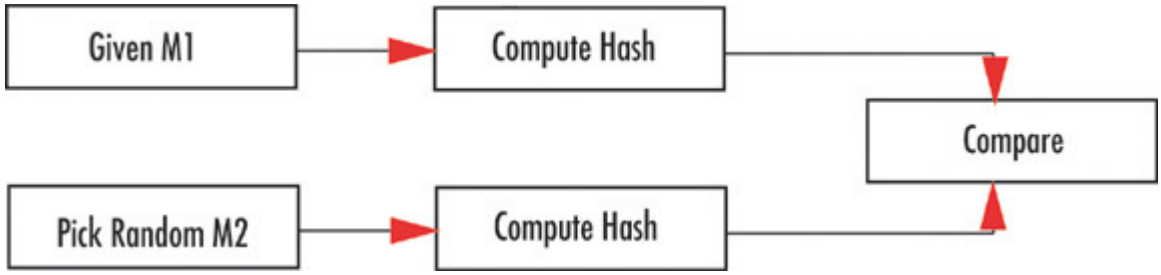

Figure 3.2 Second Pre-Image Collision Resistance[4]

Throughout the years, there have been multiple proposals for secure hash functions. We may have even heard of algorithms such as MD4, MD5, or HAVAL*. All of these algorithms have held their place in cryptographic tools and all already have been broken.

*Haval is a cryptographic hash function

MD4 and MD5 have exposed to be quite insecure as they are not collision resistant. HAVAL is suffering a similar destiny, but the designers were careful enough to over design the algorithm. So far, it is still secure to use. NIST has provided designs for what it calls the Secure Hash Standard (FIPS 180-2), which includes the older SHA-1 hash function and newer SHA-2 family of hashes (SHA stands for Secure Hash Algorithm). [4]

3.1 SHA Hash functions

SHA-1 was the first family of hashes proposed by NIST. Originally, it was called SHA, but a flaw in the algorithm led to a tweak that became known as SHA-1 (and the old standard as SHA-0). NIST only recommends the use of SHA-1 and not SHA-0. [5]

SHA-1 is a 160-bit hash function, which means that the output, also known as the digest, is 160 bits long. Like HAVAL, there are attacks on reduced variants of SHA-1 that can produce collisions, but there is no attack on the full SHA-1 as of this writing. The current recommendation is that SHA-1 is not insecure to use, but people instead use one of the SHA-2 algorithms.

SHA-2 is the informal name for the second round of SHS algorithms designed by NIST. They include the SHA-224, SHA-256, SHA-384, and SHA-512 algorithms. The number previous SHA indicates the digest length. In the SHA-2 series, there are actually only two algorithms. SHA-224 uses SHA-256 with a small modification and truncates the output to 224 bits. Likewise, SHA-384 uses SHA-512 and truncates the output. The current recommendation is to use at least SHA-256 as the default hash algorithm, especially if we are using AES-128 for privacy.

SHA-1 is the most common of the existing SHA hash functions, and applied in many security applications and protocols. In 2005, security defects were recognized in SHA-1, namely that a possible mathematical weakness might be, indicating that a stronger hash function should be needed. [5]

Although no attacks have yet been reported on the SHA-2 deviations, they are algorithmically similar to SHA-1 and so efforts are in progress to develop improved alternatives. [6]

A new hash standard, SHA-3, is currently under development .A function will be selected via an open competition running between 2008 and 2012.Keccak function, that we want to explain and implement on hardware is one of the candidates of that open competition named Cryptographic Hash Algorithm Competition.

3.2 Sponge Functions

A cryptographic hash function which we demand secure should behave like a random oracle: it should avoid weaknesses that a random oracle does not have. Iterated hash functions can never satisfy this ideal because they have inner collisions. Here the offer is a construction with a finite state called a sponge and demonstrates that a random sponge can only be distinguished from a random oracle with respect to inner collisions. The strength of random sponges are evaluated by computing the probability of success for a number of attacks as a function of their workload and that these results brought new wievpoints on the classical methods to build Hash functions. In Keccak it is proposed the usage of random sponges that given parameters as a reference for specifying security claims for hash functions, and also MAC functions and some types of stream ciphers. Main idea of sponge functions is to be able to formulate a compact security argue. [7]

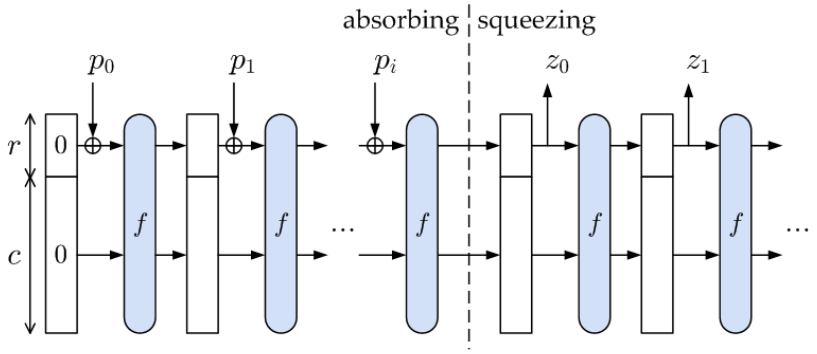

Figure 3.3 Sponge Construction [10]

3.3. NIST Cryptographic Hash Algorithm Competition

3.3.1 Secure Hash Standard

In 1993 first hash standard was concerned by NIST as Federal Information Processing Standard (FIPS) 180, *Secure Hash Standard*, which specified the hash algorithm SHA-0. This standard was revised and issued as FIPS 180-1 in 1995 and as FIPS 180-2 in 2002. These revisions substituted the original SHA-0 with more secure algorithms: the 160-bit SHA-1 and the SHA-2 family of hash functions, which includes SHA-224, SHA-256, SHA-384, and SHA-512. [8]

Cryptanalysts have found ways to attack numerous commonly used hash functions recently, and weaknesses have been published on SHA-1. Although no practical attacks have been successful to date against SHA-1, NIST determined that a new hash algorithm is required to enhance the hash algorithms that are currently available and to provide strengthened security for digital signature and other applications for upcoming years.

NIST has released a public competition to build up a new cryptographic hash algorithm, which converts a variable length message into a short "message digest" that can be available for digital signatures, message authentication and other applications. The competition is NIST's response to recent advances in the cryptanalysis of hash functions. The new hash algorithm will be called "SHA-3" and will augment the hash algorithms currently specified in FIPS 180-2, Secure Hash Standard.[8]

3.3.2Technical Considerations for SHA-3

NIST does not plan to leave the SHA-2 algorithms unless serious failings are found, and is interested in SHA-3 candidates that can be replace for SHA-2 in current applications and that can supply message digests of 224, 256, 384, and 512 bits. The 160-bit hash value formed by SHA-1 is becoming less popular to use in digital signatures; hence, a 160-bit replacement hash algorithm is not considered. [9]

SHA-3 should preserve the following properties of SHA-2 hash functions:

• input parameters

• output sizes

• collision resistance, preimage resistance, and second-preimage resistance

• “one-pass” streaming mode of execution.

It is also desirable that SHA-3 offer features or properties that exceed, or improve upon, SHA-2.

The security strength of SHA-3 should be as close to the theoretical maximum as achievable for the different required hash sizes, and for both the collision resistance and one-way properties. SHA-3 algorithms should be designed so that potentially successful attacks on SHA-2 would not be successful on SHA-3 functions. Additionally, SHA-3 should be implementable on a big range of platforms, and should be more proficient than the hash algorithms currently specified in FIPS 180-2. [9]

The algorithm “Keccak”, for which the workouts of the hardware implementation are done in this thesis, is one of the candidates that submitted for the first round that has technical requirements as an SHA-3 candidate. Here subsequently we will give information about the general structure, sub blocks and functions that included Keccak algorithm.

4. KECCAK HASH FUNCTION

4.1 Choosing the Sponge Construction

The Keccak hash function makes use of the sponge construction, following the definition. Usage of the sponge construction brings following advantages related to constructions that compose use of a compression function: [10]

Provability It has a proven upper bound for the success probability, and hence also a lower bound for the expected workload, of generic attacks.

Simplicity When we compare to the other constructions for which upper bounds have been established for the success of generic attacks, the sponge construction is very simple, and it also offers a bound that can be articulated in an easy way.

Variable-length output Keccak can generate outputs of every length and that's why a single function can be used for different output lengths.

Flexibility High level of security can be obtained at the cost of speed by simply increasing the capacity, using the same permutation (or transformation).

Functionality With the help of its long outputs and proven security bounds with respect to generic attacks, a sponge function can be used in a straightforward way as a MAC function, stream cipher, deterministic pseudorandom bit generator and a mask generating function to support arbitrary bit strings as input, the sponge construction requires a padding function.

4.1.2 Choosing an Iterated Permutation

The sponge construction requires an underlying function f, either a transformation or a permutation. In Keccak, a permutation have chosen, that constructed as a sequence of (more or less) identical rounds due to the following leads: [10]

Block cipher experience An iterated permutation is an iterated block cipher with a fixed key. Cryptanalysis and on knowledge from block cipher design can be built in it's design.

Memory efficiency a transformation is usually built by taking a permutation and adding a feed forward loop. This means that (at least part of) the input must be reserved during the complete computation.

Compactness Iteration of a single round allows to a compact specification and potentially compact code and hardware circuits. When we consider previous specifications of Keccak, we can have a sight that Keccak is one of the important candidates of the NIST cryptographic hash algorithm competition.

4.2.Keccak Permutations

There are 7 Keccak-f permutations, indicated by Keccak-f[b], where $b = 25 * 2^l$ and l ranges from 0 to 6. Keccak-f[b] is a permutation over $s \in Z_2^b$, where the bits of s are numbered from 0 to b - 1. It is called as the width of the permutation. The permutation used for that thesis is SHA-3 candidate, Keccak-f[1600].

In the development, the permutation Keccak-f[b] is described on a state a that is a three-dimensional array of elements of GF(2). The mapping between $w = 2^l$ the bits of s and those of a is $s[w(5y + x) + z] = \alpha[x][y][z]$.The expression a[x][y][z] with x,y$\in$ Zs and $z \in Z_w$ denotes the bit in position (x; y; z). Indexing starts from zero; expressions in the x and y coordinates should be taken modulo 5 and expressions in the z coordinate modulo w. Keccak-f[b] is an iterated permutation, consisting of a sequence of nr rounds R, indexed with i_r from 0 to n_r -1. A round consists of five steps:

$$R = \iota o \chi o \pi o \rho o \theta \qquad (4.1)$$

The additions and multiplications between the terms are in GF(2). [11]

Design and hardware implementation of the Keccak Round function is most important part of this thesis. Keccak is one of strong candidates from the SHA-3 applicants and may be used in many applications for digital signature in the following years. The code that could be implemented on a FPGA is the source code that is prepared for the competition. Hardware implementation of the Keccak function is important as it can be an illustration for the following projects that might be done about Keccak function.

This thesis includes general properties of the Hash algorithm and Keccak function. We decided to build the implementation on an FPGA by using VHDL(Very high speed Integrated circuit Hardware Description Language).After enough information was obtained about VHDL(code, simulating) and Keccak, it was started to write code to equivalent Keccak following sub functions.

4.2.1 Realization of θ

(4.2) θ is a linear mapping aimed at diffusion and is translation-invariant in all directions.

Without it, the Keccak-f round function would not provide diffusion of any significance.

The θ mapping has a branch number as low as 4 but provides a high level of diffusion on the average.

The round function starts with θ because of the usage of Keccak-f in the sponge construction. It gives availability for mixing between the inner and outer parts of the state. Typically, the inner part is the part that is unknown to, or not under the control of the adversary. The order of the other step mappings is arbitrary. [10]

$$\theta : \alpha[x][y][z] \leftarrow \alpha[x][y][z] + \sum_{y'=0}^{4} a[x-1][y'][z] + \sum_{y'=0}^{4} \alpha[x+1][y'][z-1] \qquad (4.2)$$

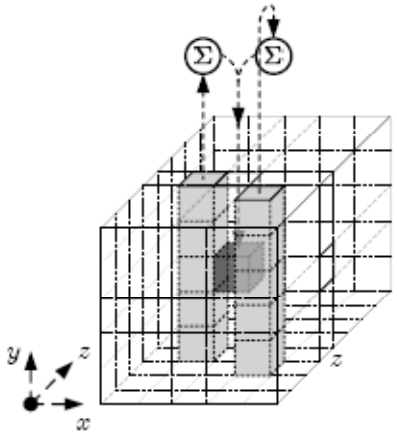

Figure 4.1 θ Applied to a Single Bit [10]

It is first comprehended that for the realization of the Keccak-f permutation, a 4x4x63 matrix needed. For that reason a component in VHDL as consisting of 4x4x63 matrix was defined.

After we define our necessary matrix component, the mathematical realization of the θ was started. From the equation that could be seen (4.1) it is estimated that for loops for the sum operators and XOR for the intermediate calculations. What here should be considered is, z can be negative for the z<1 situations .To solve this problem the state that z<1 for the (4.2) is written under an else-if case in VHDL to avoid possible negative terms. Afterward with the help of for loops, matrix that used from the component was filled. And obtained *b* output from a *a* input matrix with regard to equation (4.2)

4.2.2 Realization of `r`

The mapping `r` consists of translations within the lanes expected at providing inter-slice dispersion. Without `r`, diffusion between the slices would be very late. It is translation invariant in the z-direction. [10]

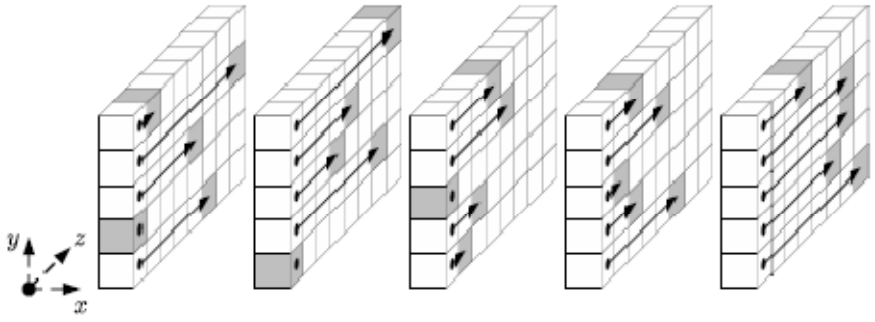

Figure 4.2 r applied to the lanes [10]

$$\rho : \alpha[x][y][z] \leftarrow \alpha[x][y][z-(t+1)(t+2)/2],$$

with t satisfying $0 \leq t < 24$ and $\begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^t \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix}$ $GF(5)^{2*2}$ (4.3)

It can be clearly remarked that (4.3) to solve the r mathematical equation, t value should be calculated. To avoid more computing in FPGA, values of t are calculated in MATLAB(the powers of the given matrix) and used directly in VHDL code. After t is calculated, similar to θ algorithm, for the negative values of z; else-if statements were created and z assigned as zero for the possible negative values. Afterward with the help of for loops, matrix that used from the component was filled. And obtained *b* output from a *a* input matrix with regard to equation (4.3).

4.2.3 Realization of p

The mapping p is a transposition of the lanes that gives dispersion meant at long-term diffusion. Without it, Keccak-f would exhibit periodic trails of low weight. p operates in a linear way on the coordinates (x; y).

$$\pi : \alpha[x][y] \leftarrow \alpha[x'][y'] \quad \text{with} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^t \begin{pmatrix} x' \\ y' \end{pmatrix} \tag{4.4}$$

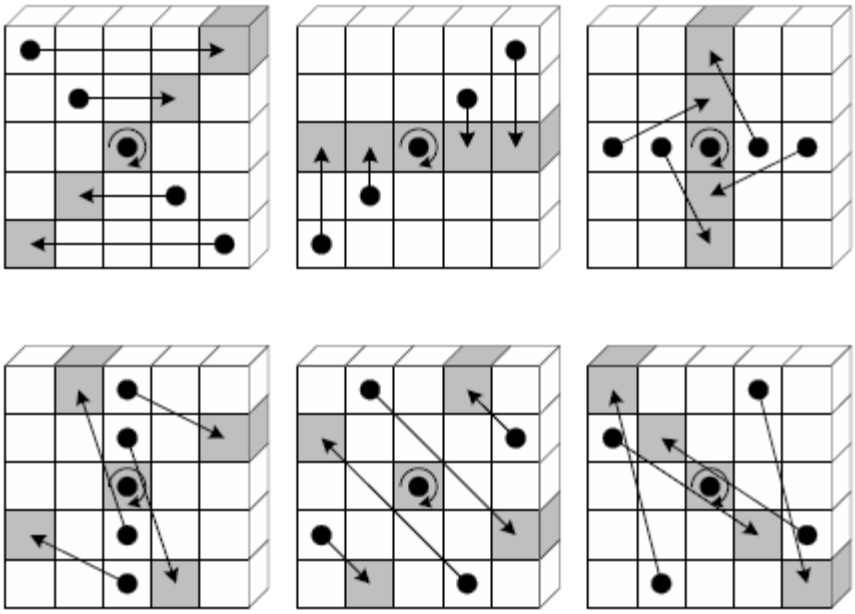

Figure 4.3 p applied to a slice [10]

With respect to mathematical equation of p it can assumed that x' and y' values should be found. Again avoiding from FPGA's overwork, x' and y' values are calculated with the help of MATLAB. Consequently these values that are founded with MATLAB , are integrated to VHDL code. Also obtained *b* output from a *a* input matrix with regard to equation (4.4)

4.2.4 Realization of c

c is the only nonlinear mapping in Keccak-f., the Keccak-f round function would be linear without using of c . It can be observed, as the parallel application of 5w S-boxes operating on 5-bit rows. c is translation-invariant in all directions and has algebraic degree two, which has the following consequences:

- The cardinality of a differential (a0; b0) over c is either zero or a power of two. The related (restriction) weight $w(a_0; b_0) = w(a_0)$ is an integer that only depends on the input difference pattern a0.
- For a given input difference a0, the space of possible output differences forms a linear affine variety with $2^{w(a_0)}$ elements.

- A possible differential imposes w(a0) linear conditions on the bits of input a.
- For a given output selection vector u, the space of input selection vectors v whose parities have a non-zero correlation with the parity determined by u form a linear affine variety with 2w(u;v) elements, with w(v; u) = w(u) the (correlation) weight function that returns an even integer that only depends on the output selection pattern u.
- The magnitude of a correlation over c is either zero or equal to 2w(u).

$$\chi : \alpha[x] \leftarrow \alpha[x] + (a[x+1]+1])a[x+2] \quad (4.4)$$

From the equation that given above, it can be seen here in c we have a one dimension matrix. But for fitting that sub function to others and to not to have problems during the sum of the sub functions , in VHDL again a three dimensional matrix is used to realize the c function. And obtained b output from a a input matrix with regard to equation (4.4)

4.2.5 Realization of ι

The mapping ιconsists of the addition of round constants and is aimed at disrupting symmetry.

Without it, the round function would be translation-invariant in the z direction and all rounds of Keccak-f would be equal making it subject to attacks exploiting symmetry such as slide attacks. The number of active bit positions of the round constants, i.e., the bit positions in which the round constant can differ from 0, is $l + 1$. As l increases, the round constants add more and more asymmetry.

The bits of the round constants are different from round to round and are taken as the output of a maximum-length LFSR. The constants are only added in a single lane of the state. Because of this, the disruption diffuses through θ and $\rho$to all lanes of the state after a single round.

In hardware, the computational cost of ι is a few XOR's and some circuitry for the generating LFSR. In software, it is a single bitwise XOR instruction.

$$\iota : a \leftarrow a + RC[i_r] \tag{4.5}$$

With the exception of the value of the round constants RC[ir], these rounds are identical. The round constants are given by (with the first index denoting the round number)

$$RC[i_r][0][0][2^j - 1] = rc[j + \Gamma i_r] \qquad \text{for all } 0 \leq j \leq l \tag{4.6}$$

and all other values of $RC[i_r][x][y][z]$ are zero. The values rc[t] $\in$ GF(2) are defined as the output of a binary linear feedback shift register (LFSR):

$$rc[t] = (x^t \bmod x^8 + x^6 + x^5 + x^4 + 1) \text{ mod x in GF(2)[x]:} \tag{4.7}$$

The number of rounds nr is determined by the width of the permutation, namely,

$$n_r = 12 + l \tag{4.8}$$

In this subsection of realization of the round function, first a part of VHDL code is written with respect to equation (4.5). Afterwards a package under VHDL has created as round constant signal which has the constants that are necessary for realization of the`i`. This package is used as an sub package of the `i` function.

4.2.6 Realization of the Round Function

After all sub functions of R (4.1) are created in VHDL editor, a new VHDL module source file has opened , and mapping between sub functions is created. First input of the round function comes to θ and output of the round function created by `i`.R is set as a top module and for testing R, a test bench in VHDL as R_tb is created. With the help of simulation tool ,test of R is done.

4.3. Keccak sponge function

Keccak's special sponge function has also been studied. As can be seen from Figure(4.4) Keccak has a special sponge algorithm. During the thesis this algorithm was also learned and worked on the realization of that algorithm on an FPGA.

Algorithm 1 KECCAK$[r, c, d]$

Input $M \in \mathbb{Z}_2^*$
Output $Z \in \mathbb{Z}_2^*$
$P = \text{pad}(M, 8)||\text{enc}(d, 8)||\text{enc}(r/8, 8)$
$P = \text{pad}(P, r)$
Let $P = P_0||P_1||\ldots||P_{|P|-1}$ with $P_i \in \mathbb{Z}_2^r$
$s = 0^{r+c}$
for $i = 0$ to $|P| - 1$ **do**
  $s = s \oplus (P_i||0^c)$
  $s =$ KECCAK-$f[r + c](s)$
end for
$Z =$ empty string
while output is requested **do**
  $Z = Z||\lfloor s \rfloor_r$
  $s =$ KECCAK-$f[r + c](s)$
end while

Figure 4.4 Keccak sponge Function [11]

In Keccak Sponge function:

- pad(M; n) refers to returns a bit string obtained by appending to the bit string M a single 1 and the smallest number of zeroes such that the length is a multiple of n. A state machine is used, that collects the data received. And than adds zeros till bit length became dividable to 8.

- enc(x; n) returns a string of n bits coding the integer x, from the least significant bit to the most significant bit, i.e., it returns the string x0; x1…. ; xn-1.Code that refers enc shows x's n bits shown in binary system. A combinational circuit is designed to realize enc function.

4.4 Simulation Results and Comments

After the algorithm that are necessary for the Keccak function is created a top module as Keccak is created and other modules are implemented as components. After the top module is created, with Xilinx simulator Synthesis/Implementation steps are done and Map Report is analyzed. The results that can be seen from Map Report ;

Logic Utilization:
Number of Slice Flip Flops: 2,686
Number of 4 input LUTs: 5,773
Logic Distribution:
Number of occupied Slices: 3,408
Number of Slices containing only related logic: 3,408

And with respect to timing constraints best frequency achieved is found as:

23.25MHz

After these results was seen, Behavioral Simulation of the VHDL codes are done. With the behavioral simulation , the following graphics are obtained.

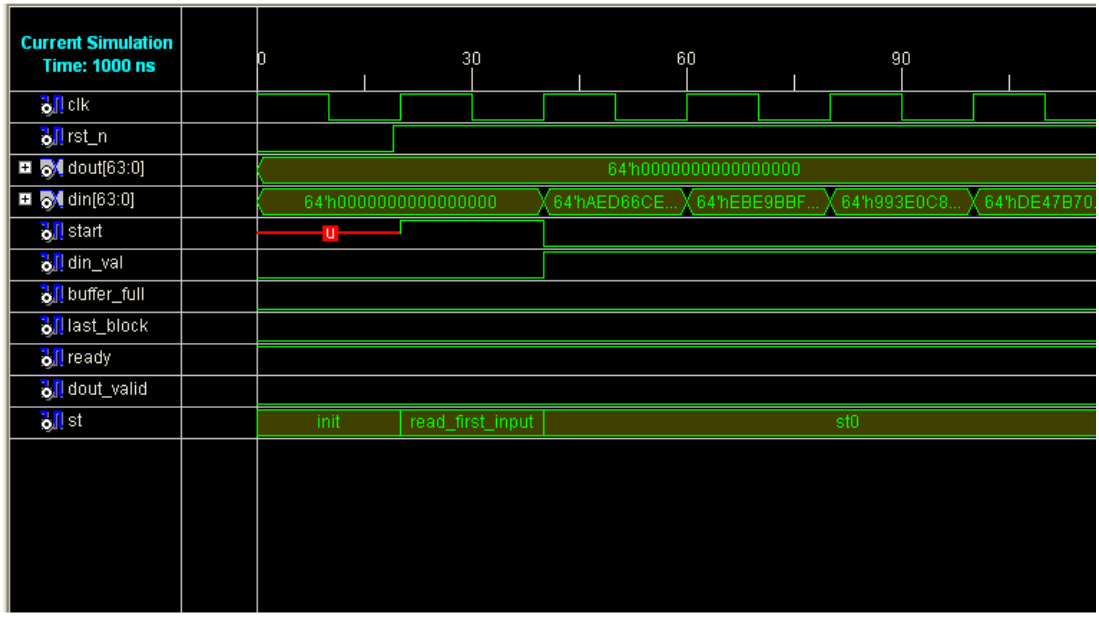

Figure 4.5 Behavioral Simulation Result

When control pin(st) is read_first_input and rising , with rising edge of clock and din_val, Keccak stats to read inputs(test vectors)

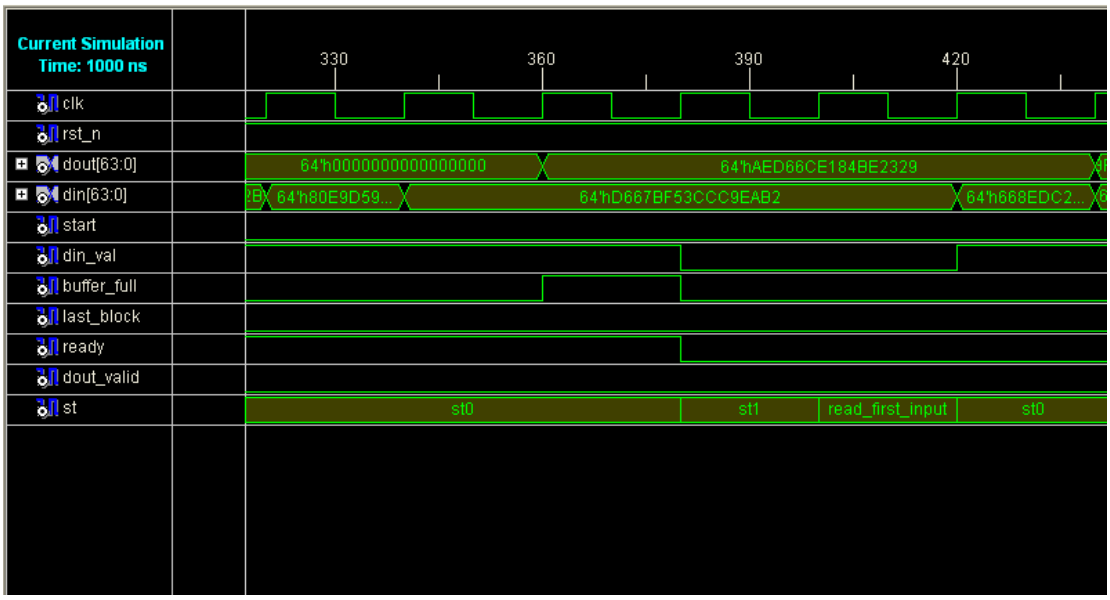

Figure 4.6 Behavioral Simulation Result

When buffer_full and clk are rising edge ,under din_val=1 and ready=1 and st0, Keccak starts to give outputs.

When we compare the outputs that we have and the test vector results that Keccak's creators published as test vector results; it can be seen that we have the same test vector outputs that supposed to be found.

5. RESULTS

In the project of Hardware implementation of Keccak Hash function, general information about cryptology is given in the beginning. Afterwards different varieties of Hash functions are mentioned and talked which are also commonly in use. It is also learned why we need new cryptographic functions and what are being made to establish new cryptographic functions that could be in common use in the following years.

It is given information about NIST Hash Algorithm Competition. It is asserted, due to which features of the Keccak Hash function, it is a secure, original function and is a strong applicant of the competition. The sub functions of the Keccak algorithm are learned, and mathematical functions that required for realization of the keccak Round function are given. The workouts which are necessary for the hardware implementation are done. It is mentioned what works are done by outside sources during the project, to avoid the FPGA's overwork.

This thesis come into prominence as being the Keccak's one of the strongest candidates of the possible future's crypto function and the workouts for the Keccak function can be a source for the projects that could be done in the future.

REFERENCES

[1] **Jaworski J., Perrone P.,** 2000. Java Security Handbook ,Sams Press ,London

[2] **De Laet G., Schauwers G.**, 2004. Network Security Fundamentals ,Cisco Press , Indianapolis

[3] **Menezes A., Van Oorschot P., and Vanstone S.,** 1996. Handbook of Applied Cryptography, CRC Pres, New York

[4] St **Denis T.,** 2006. Cryptography for Developers, Syngress Publishing , Rockland

[5] Schneier on Security: Cryptanalysis of SHA-1

http://www.schneier.com/blog/archives/2005/02/cryptanalysis_o.html

[6] **Wobst R., Schmidt J.,** Hash cracked; The consequences of the successful attacks on SHA-1

http://www.schneier.com/blog/archives/2005/02/cryptanalysis_o.html

[7] **Bertoni G., Daemen J., Peeters M., Van Assche1G.**
Sponge Functions

http://sponge.noekeon.org/SpongeFunctions.pdf

[8] Cryptographic hash Algorithm Competition

http://csrc.nist.gov/groups/ST/hash/sha-3/index.html

[9] **Radack S.** New Cryptographic Hash Algorithm Family: NIST Holds a Public Competition to Find New Algorithm

http://csrc.nist.gov/publications/nistbul/b-May-2008.pdf

[10] **Bertoni G., Daemen J., Peeters M., Van Assche1G.** Keccak sponge function family main document, http://keccak.noekeon.org/

[11] **Bertoni G., Daemen J., Peeters M., Van Assche1G.** Keccak specifications, http://keccak.noekeon.org/

APPENDIXES

A-1: Source codes and project of the realization (Saved on a CD).

RESUME

Özgür YONKUÇ was born in 1986 in Sivas. After he graduated from A.T.O. Anatolian High School (Adana) in 2004, he has started his education at I.T.U. in electronics engineering. He attended for one year at TUHH (Technical University of Hamburg) as an Erasmus exchange student. He knows English and German.