ANALYSES, MITIGATION AND APPLICATIONS OF SECURE HASH ALGORITHMS

A Dissertation
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Zeyad Abdel-Hameed Al-Odat

In Partial Fulfillment of the Requirements
for the Degree of
DOCTOR OF PHILOSOPHY

Major Department:
Electrical and Computer Engineering

November 2019

Fargo, North Dakota

NORTH DAKOTA STATE UNIVERSITY

Graduate School

Title

ANALYSES, MITIGATION AND APPLICATIONS OF SECURE HASH ALGORITHMS

By

Zeyad Abdel-Hameed Al-Odat

The supervisory committee certifies that this dissertation complies with North Dakota State University's regulations and meets the accepted standards for the degree of

DOCTOR OF PHILOSOPHY

SUPERVISORY COMMITTEE:

Samee U. Khan
Chair

Jacob Glower

Sudarshan Srinivasan

Ying Huang

Zhang Limin

Approved:

1, November, 2019
Date

Ben Braaten
Department Chair

ABSTRACT

Cryptographic hash functions are one of the widely used cryptographic primitives with a purpose to ensure the integrity of the system or data. Hash functions are also utilized in conjunction with digital signatures to provide authentication and non-repudiation services. Secure Hash Algorithms are developed over time by the National Institute of Standards and Technology (NIST) for security, optimal performance, and robustness. The most known hash standards are SHA-1, SHA-2, and SHA-3.

The secure hash algorithms are considered weak if security requirements have been broken. The main security attacks that threaten the secure hash standards are collision and length extension attacks. The collision attack works by finding two different messages that lead to the same hash. The length extension attack extends the message payload to produce an eligible hash digest. Both attacks already broke some hash standards that follow the Merkle-Damgård construction.

This dissertation proposes methodologies to improve and strengthen weak hash standards against collision and length extension attacks. We propose collision-detection approaches that help to detect the collision attack before it takes place. Besides, a proper replacement, which is supported by a proper construction, is proposed. The collision detection methodology helps to protect weak primitives from any possible collision attack using two approaches. The first approach employs a near-collision detection mechanism that was proposed by Marc Stevens. The second approach is our proposal. Moreover, this dissertation proposes a model that protects the secure hash functions from collision and length extension attacks. The model employs the sponge structure to construct a hash function. The resulting function is strong against collision and length extension attacks. Furthermore, to keep the general structure of the Merkle-Damgård functions, we propose a model that replaces the SHA-1 and SHA-2 hash standards using the Merkle-Damgård construction. This model employs the compression function of the SHA-1, the function manipulators of the SHA-2, and the $10*1$ padding method. In the case of big data over the cloud, this dissertation presents several schemes to ensure data security and authenticity. The schemes include secure storage, anonymous privacy-preserving, and auditing of the big data over the cloud.

ACKNOWLEDGEMENTS

All praises and thanks to Allah almighty, my Creator, my Sustainer, for giving me courage and strength to pursue my PhD and fulfill the requirements of this disquisition.

My heartiest and sincere appreciation and gratitude to my mentor and adviser Dr. Samee U. Khan, who always encouraged me, and persistently conveyed the spirit and guidance required for the research. Without his kind guidance and continuous efforts, this disquisition would not have been possible.

Special thanks to my committee members, Dr. Jacob S. Glower, Dr. Sudarshan K. Srinivasan, Dr. Ying Huang, and Dr. Zhang Limin for their support, guidance and helpful recommendations. Thanks to the Electrical and Computer Engineering staff members for all the unconditional help and favor. I owe my heartiest thanks to all my friends and colleagues here in the US and Jordan, who always helped me in the time of need.

Finally, I would like to thank my family, Father, Mother, Brothers (Maen, Mohammad, Ali, Zaher, Moath), and Sisters (Nisreen, Yasmeen, Haneen, Sadeen). Their continuous support is always a source of motivation and encouragement for me. I especially like to thank my mother and father, who are the only and every reason for whatever I am today and whatever I achieved in my life. I also would like to thank my loving wife Eman and my son Adam, for their patience, time, and support.

DEDICATION

I would like to dedicate this dissertation to my family, especially to my parents, my wife, and my son for all the love, support, and motivation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

1.1. Overview

Authenticity and confidentiality are big concerns for a long time. One of the first used methods for confidentiality sending a message as a tattoo on the shaved head of a trusted slave. Then, when the hair grows, the slave is sent with the instruction to shave the head to read the message. This example shows the historical use of security paradigms.

In cryptography, there is a type of one-way function that is important. A one-way function maps a large message (from different domains) into a finite range of output. The term one-way means that the function is easy to compute but difficult to invert, i.e., for a given output it is difficult to find an input that maps to the given output. This property depends on the security framework of a given function and its hardness against inversion. However, an adversary with unlimited computing power can test every possible input that may map to the given output.

All proposed one-way functions were established based on well-defined assumptions. For cryptography, the class of one-way function is considered as an important class, which is represented by hash functions. Hash functions operate on bit strings, where they map arbitrary length bit strings to a fixed length. This fixed-length bit string is called the *hash*.

In the last decades of the 20^{th} century, we witnessed Message Digest 4 (MD4) developed by Ronald Rivest, for the provision of integrity [1]. The MD4 provided the basis for MD5, which is a strengthened version of MD4 [2]. Both the MD4 and the MD5 worked to compute 128 bits digest. However, weakness in aforesaid algorithms resulted in the development of the Secure Hash Algorithm (SHA-0) by the National Institute of Standards and Technology (NIST) in 1993. The SHA-0 was followed by SHA-1 in 1995 with the hash size of 160 bits. The SHA-1 replaced MD5 that was exposed to collision attacks by then [3]. As with all of the security protocols, the SHA-1 was also under constant scrutiny by the security experts. Wang *et al.* [4] claimed that collisions in SHA-1 can be found with complexity less than 2^{80} compression function calculations. Recently the claim of Wang *et al.* was validated by Marc *et al.* [5] by finding hash collisions for different files.

Hash functions have a variety of applications including, data integrity, verification, and authorization. Fig 1.1 shows the process of verifying a message (M), that is sent over an insecure

medium. The message hash is computed by a hash function in the source side and appended to the end of the message. At the destination side, the message hash is computed and compared with the appended hash value. If both hashes are equal, then the received message at the destination side is unaltered. The hash functions also assist in the digital signature that is part of authorization and validation. Moreover, hash functions are also used for the generation of random numbers and password protection [6].

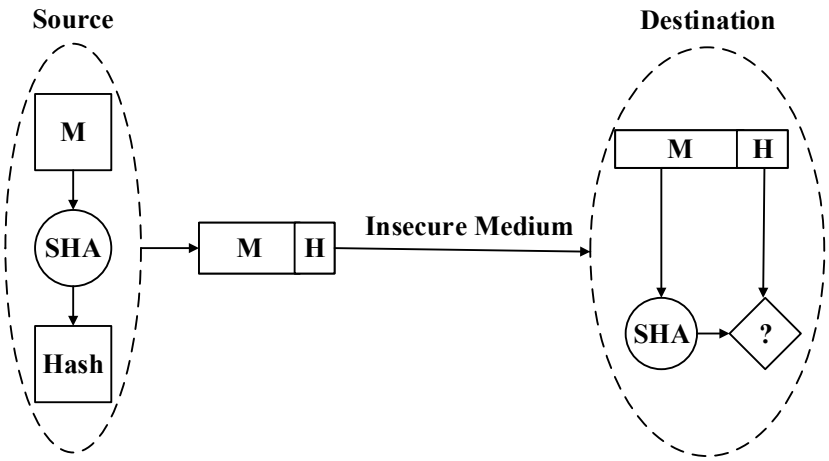

Figure 1.1. Application of secure hash algorithm comparing the hash value of the message sent over insecure channel

1.2. Cryptographic Hash Functions

1.2.1. Cryptographic Requirements of Hash Functions

One of the major applications of the hash functions is the construction of an efficient digital signature. The signature is computed by applying a set of compression operations to produce a fixed-length hash. Breaking the signature requires finding a message M' that has the same hash as a different message M. Therefore, if either M or M' is signed, the corresponding signature is valid to the other message, resulting in a successful forgery.

For cryptography, a set of properties are taken into consideration at the time of design. These properties comprise Pre-image, Sec-Pre-image, Collision, MAC, and PRF. For more details about these properties, we refer to [7, 8, 9].

1. **Pre-image**: Which is based on the problem of finding a pre-image M for a given hash h. Pre, which stands for *pre-image resistance*, requires that given a randomly chosen hash function f and a uniformly randomly chosen hash h it is hard to find a pre-image $M \in f^{-1}(h)$, as shown in Figure 1.2.

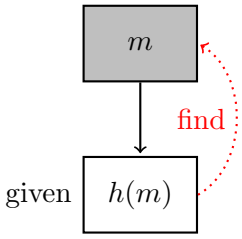

Figure 1.2. Pre-image

2. **Sec-Pre-image**: Which is based on the problem of finding another message M' that has the same hash h as a given message M. *Sec-Pre-image* requires that the function f and message M to be uniformly and randomly chosen, as shown in Figure 1.3.

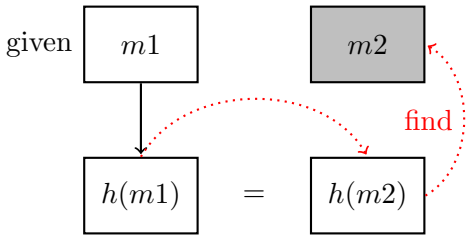

Figure 1.3. Second Pre-image

3. **Collision**: Which is the property that finding two messages M and M' that have the same hash h for the same chosen f. The hash function f must be strong enough to resist any possibility of a collision. Unfortunately, this property was the first to be broken, as shown in Figure 1.4.

4. **MAC**: Which stands for Message Authentication Code, is the property that canvasses the hash function f as a keyed hash function (f_k) where k is a randomly chosen secret key that is associated with f. The MAC property requires that the dilemma of finding a message M and the hash h with the presence of secret key k, is difficult to achieve, as shown in Figure 1.5.

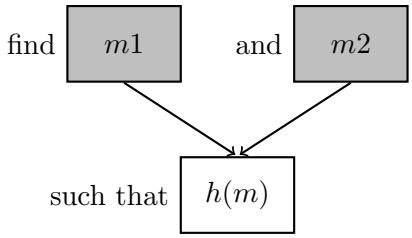

Figure 1.4. Collision

5. **PRF**: Which stands for Pseudo-Random Function, is the property that the problem of determining the actual function f_k for a randomly chosen k with a randomly chosen hash h must be difficult, as shown in Figure 1.5.

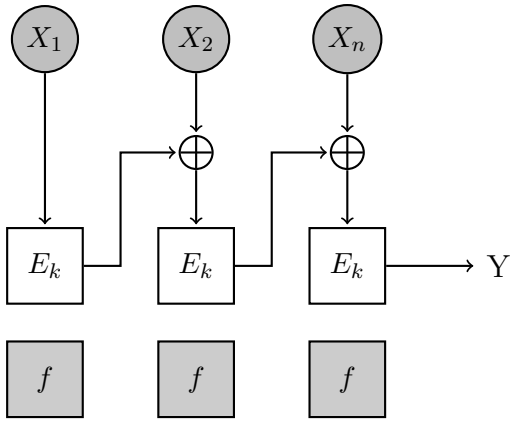

Figure 1.5. Message Authentication Code (MAC) and Pseudo Random Function (PRF).

In the case of fixed-length hash functions, only three properties are considered: pre-image resistance, 2^{nd}-pre-image resistance, and collision resistance. The other two properties, MAC and PRF, are considered when the Hashed Message Authentication Code (HMAC) is present, where the hash function is employed with the secret key.

1.2.2. Attacks on Hash Functions

There exists a general attack that breaks a security requirement of the secure hash functions. A hash function is considered as *broken* when there is an explicit attack that is faster than the general attack for one of the security requirements of a hash function. With the increase of the computational power, all secure hash functions are vulnerable to the general attack due to the small bit length that is generated by the hash functions.

1.2.2.1. Brute Force Attack

The most known general attack to break the security requirements (Pre and 2^{nd}-Pre) is a brute force attack. The brute force attack is applied to Pre and 2^{nd}-Pre by computing $f(M')$ for a randomly chosen message M' until M' is found and $M' \neq M$. In this case, this attack is considered as infeasible in reality because it requires massive hash computations ($\approx 2^{100}$ hash computations).

1.2.2.2. Birthday Paradox

For collision, a general attack based on the birthday paradox is better than the brute force attack. The birthday paradox is based on a counter-intuitive basis, where for groups of as less as 23 people there exists a chance of one half to find two people that were born on the same day [10]. Assuming that all birthdays are equally likely and neglecting leap years, for 23 independent possibilities the success probability is $\frac{23}{365} \approx 0.06$, where there are $\frac{23 \times 22}{2} = 263$ pairs of people. Then the overall success probability is 0.5 (noting that the pairs are not independently distributed, therefore 0.5 doesn't equal $\frac{253}{365} \approx 0.7$). For a hash function of hash size N, the birthday paradox attack succeeds after $\sqrt{\pi/2} \times 2^{N/2}$ computations of the hash function [11]. For instance, for a hash size of $N = 128$ bits, which is the hash size of the MD5 hash function, a collision can be found with approximately $2^{64} \approx 22 \times 10^{18}$ computations, which is in reach of a high-performance computer.

1.2.2.3. Differential Attack

The most successful attack on the Merkle-Damgård hash functions is the differential attack. The other two properties, MAC and PRF, are considered when the Hashed Message Authentication Code (HMAC) is present, where the hash function is employed with the secret key. The differential attack works by looking at two different evaluations of a hash function. These evaluations are examined at the same time by computing the difference between them and analyzes the relationship between the difference and the next evaluations' differences. This attack is used to generate a successful collision attack. The differential attack was first used against Data Encryption Standard (DES) [12, 13] based on the XOR difference. Then it was generalized to be used with the modular difference to break many hash standards. Wang *et al.* were able to break several hash standards (MD4, MD5, SHA-0, HAVAL-128, and RIPEMD) using the modular differential attack [3, 14, 4]. Because of the collision attack against Merkle-Damgård hash functions, the National Institute of

Standards and Technology (NIST) held a competition to find a new hash function SHA-3 that will be used in the future. This competition concluded by announcing Keccak as the new SHA-3 standard [15].

1.2.2.4. Same-Prefix Collision Attack

The collision attack targets the Merkle-Damgård hash functions. Particularly, the hash standards before the SHA-2. Due to the iterative nature of the Merkle-Damgård construction and the fixed size output hash, the collision attack can be structured according to predefined inputs. Figure 1.6 shows the structure of the same-prefix collision attack. Two different messages (Message1, Message2) have the same prefix (P), suffix (S) and initialized with the same Initial Hash Value (IHV). Then a pair of two different blocks (C, C^*) can be computed such that $SHA(P\|C\|S) = SHA(P\|C^*\|S)$, where IHV_n is the Intermediate Hash Value after processing the messages.

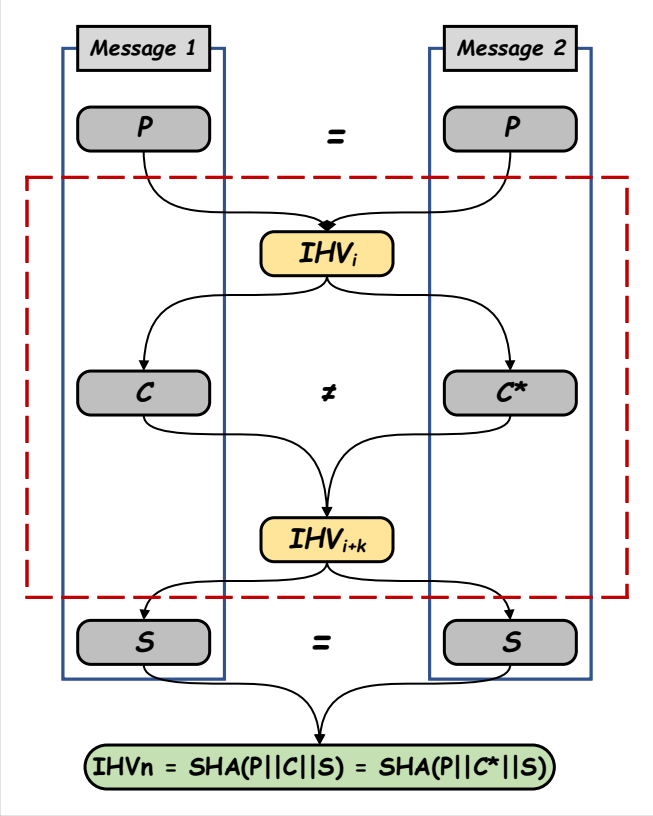

Figure 1.6. Same prefix collision attack

1.2.2.5. Chosen-Prefix Collision Attack

Stevens *et al.* improved the same-prefix collision attack into a chosen-prefix by formulating two different prefixes with two different messages. However, this approach requires more internal blocks to produce successful collision [16].

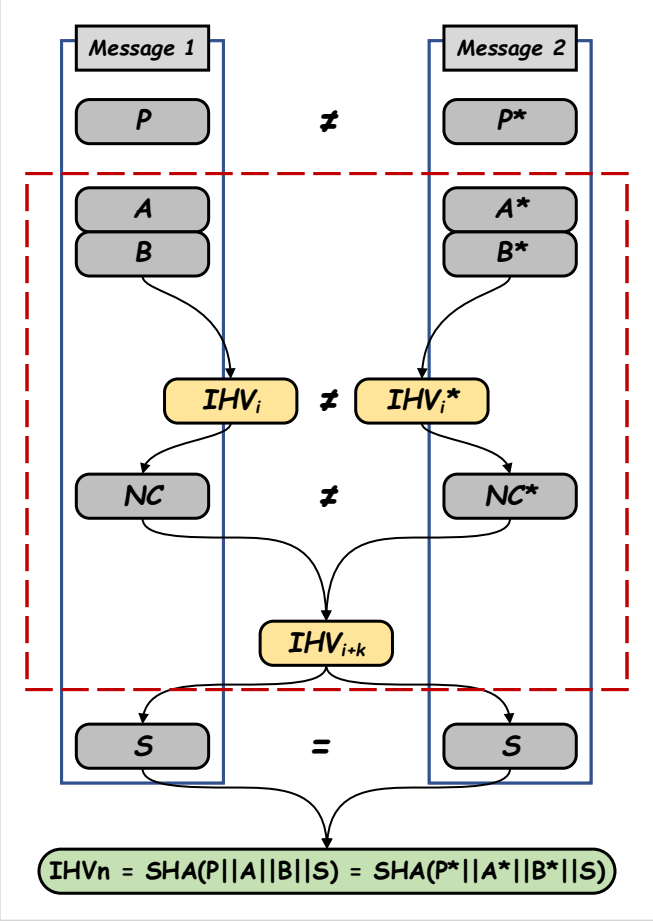

Figure 1.7. Chosen-Prefix collision attack

Figure 1.7 shows the structure of the Chosen-Prefix collision attack. This attack succeeds because of the iterative nature of the Merkle-Damgård construction. Two different messages (Message1, Message2) started with two different prefixes (P, P^*). These prefixes may contain different length strings, then both messages are padded using blocks A and A^* to make them equal in size. According to predefined conditions, a birthday search is applied to find two blocks B and B^* such that $P\|A\|B$ and $P^*\|A^*\|B^*$ have the same size and produce two different intermediate hash values (IHV_i and IHV_i^*) that have predefined structures. Finding the proper predefined structures leads

to generate two near-collision blocks (NC and NC^*) that produce the same IHV_{i+k}. This attack requires that both messages have the same suffix (S). Finally, the successful chosen-prefix collision attack maintains $SHA(P\|A\|B\|S) = SHA(P^*\|A^*\|B^*\|S)$.

1.2.3. Constructions of Hash Functions

1.2.3.1. The Merkle-Damgård Construction

The need for a structured model for use in the digital signature formulation was needed. The Merkle-Damgård construction (which takes the name of two authors of two different works [17, 18], in 1989) describes constructing a hash primitive based on a compression function F. Figure 1.8 shows the general structure of the Merkle-Damgård construction, where a message (M) is divided into blocks and processed sequentially using a compression function (F). This structure requires initial values (IV) that are used to process the first block. Then, the output of each block is used to process the next block. The final output is produced after processing the last block of the message.

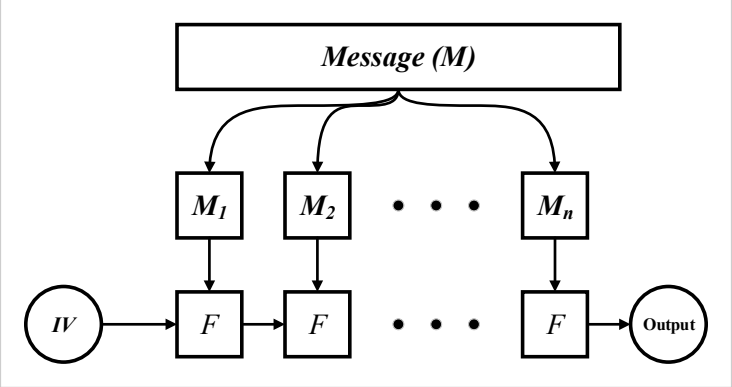

Figure 1.8. The general structure of the Merkle-Damgård construction

Based on the idea that was presented by Merkle and Damgård [19], Rivest, in 1990, was the first to present the MD4 hash function based on the Merkle-Damgård construction [1, 20]. Due to security issues, MD4 was repudiated by MD5 in 1992 [2]. MD4 and MD5 formed the base for the following hash functions that come after, which have the same Merkle-Damgård construction. We list examples of these functions as follows:

- **SHA-0**: Which was designed by the National Security Agency (NSA), in 1993 [21].

- **SHA-1**: Which was placed as a replacement to the SHA-0 after undisclosed security concerns, in 1995 [22].
- **SHA-2**: Which was designed starting from 2002 with four different flavors SHA-224, SHA-256, SHA-384, and SHA-512 [23, 24, 25].
- **RIPEMD**: Which stands for RACE Integrity Primitives Evaluation Message Digest, which was published in 1995 [26].
- **RIPEMD-160**: RIPEMD-160 is a strengthened version of RIPEMD with a 160-bit hash result. Which was published in 1996 [27].

Each of Merkle-Damgård hash functions has a distinct compression function that runs repeatedly for several steps. Besides, each standard has a distinct number of working state variables that differ from the other. For instance, the SHA-1 hash function uses five working state variables which seen as 32-bit strings. The SHA-1 compression function processes each block through 80 steps, where the working state variables are updated after every step. After processing all blocks of a message, the final hash is formulated by the concatenation of the working state variables.

1.2.3.2. The Sponge Construction

The SHA-3 was published in 2012, after a competition that was held by the NIST. The competition was divided into three rounds, the first round includes 51 candidates, the second round includes 14 candidates, and the final round includes five candidates (Keccak, Grøstl, BLAKE, JH, and Skien). Keccak won the competition as the next standard for the SHA-3 [25]. Unlike the previous standards (SHA-1 and SHA-2), the SHA-3 relies mainly on absorbing and squeeze structure [28, 29], where the data are absorbed in and squeezed out to generate the hash output. The Sponge construction comprises two phases, absorb and squeeze. After a message is preprocessed, it is divided into equal size of blocks (p_i). The values bit-rate (r) and capacity (c) reflect the permutation level that the function f will perform. Given an input length N and an output length d, the output $Z = sponge[f, r](N, d)$ is generated with d bit length. The permutation function f is applied to the output Z until the required number of bits for the d output is produced, as shown in Figure 1.9.

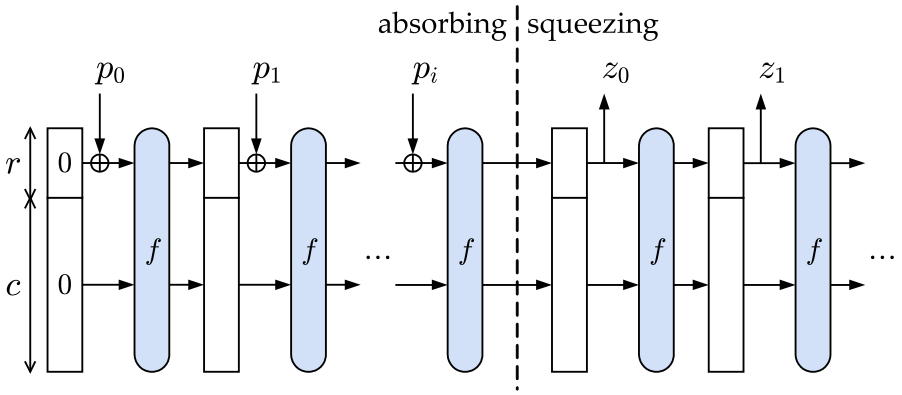

Figure 1.9. Sponge construction

1.2.3.3. HAIFA Construction

The HAsh Iterative FrAmework (HAIFA) construction was developed in 2006 by Biham *et al.*[30]. The construction of HAIFA is depicted in Figure 1.10. An optional `Salt` is added to the iterative computations of the classical Merkle-Damgård construction. Moreover, HIAFA keeps track of the number of hashed bits after each block computation. Researchers proved the strength of HAIFA construction against 2^{nd}-preimage and collision attacks if the compression function performs optimally, as shown in [31].

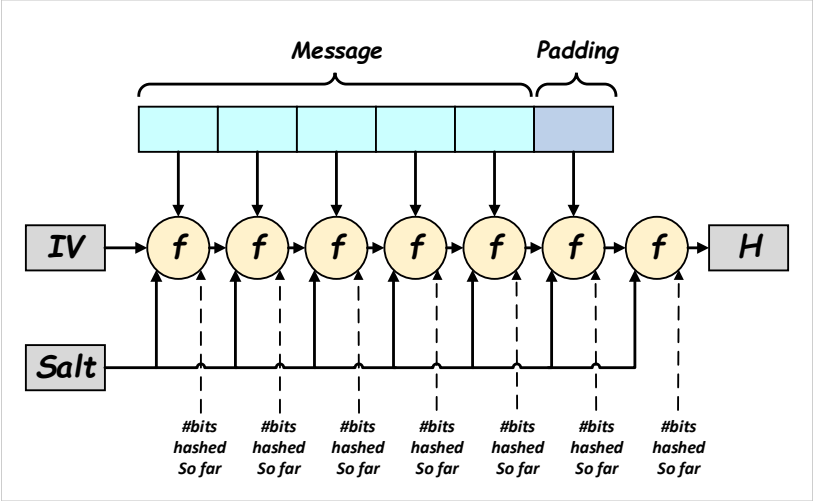

Figure 1.10. HAIFA construction

1.2.3.4. Wide-Pipe Construction

The Wide-Pipe construction behaves as the HAIFA construction without the `Salt` variables. The Wide-Pipe construction maintains two levels of compression function computations, as shown in Figure 1.11. The message compression function (f_1) that processes the message blocks. The second level performs the output compression function (f_2) to produce the output hash. The resulting hash (H) is produced after the transformation of f_1 compression function using f_2 compression function.

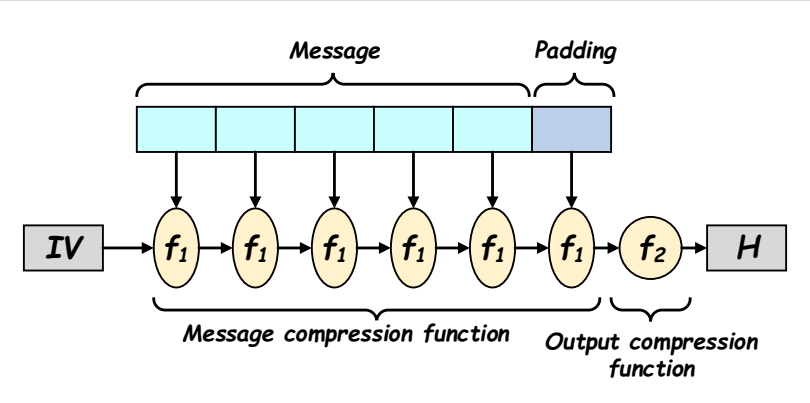

Figure 1.11. Wide-Pipe construction

1.2.3.5. Parallel-Tree Construction

The Parallel-Tree construction works by processing message blocks in parallel and performs a reduction algorithm to the generated output, as shown in Figure 1.12. A message with a long length takes time to be processed using the iterative structure. In this model, the message is processed in parallel with reduced time compared to other structures. However, vulnerabilities of this model have not been studied, which makes this model away from attentions [32].

1.3. Motivation

Secure Hash Algorithms are the most popular cryptography primitives. All hash functions are used to ensure data integrity and authenticity. However, particular attacks on the hash functions make them weak and vulnerable. The most popular attack on the hash functions is the collision attack. This attack broke Merkle-Damgård structure hash functions including MD4, MD5, and SHA-1. Part of these functions was dropped from being used anymore, but the MD5 and SHA-1

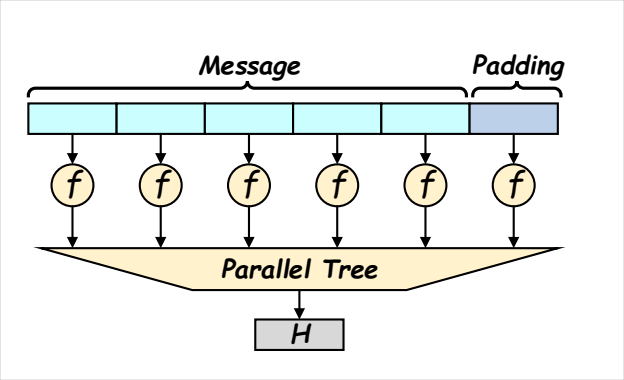

Figure 1.12. Parallel-Tree construction

are still used by entities and applications. This is because migrating from hash standard to another requires time and money. The length extension attack represents a serious threat to the hash functions. It is targeted to the Merkle-Damgård structure standards including the MD5, SHA-1, and SHA-2.

One of the hash functions' duties is to generate a secure digital signature for data. The security of the hash functions reflects the security of the data, particularly, the sensitive data. For instance, an electronic bank check that uses the digital signature can be forged using the collision attack. The forgery includes changing the withdrawal value and the name of the recipient while keeping the same signature. Moreover, the length extension attack can intercept a money transaction between two parties and generate a valid signature. These attacks also target big data applications over the cloud.

In the case of attacks on big data, an attacker tries to tamper the big data contents using the collision attack. On the cloud, data are not only stored on multiple locations, but also shared across multiple users. The integrity of data on the cloud is subject to doubt due to cloud failures. Therefore, many techniques are available to verify the existence of data on the cloud without retrieving the entire file. However, the data auditing techniques reveal confidential information to the designated verifier, which breaches the identity of the data uploaders.

This dissertation proposes methodologies to improve and strengthen weak hash standards from collision and length extension attacks. We propose a collision detection approaches that help to detect the collision attack before it takes place. Moreover, in case of a successful attack, our approach can determine a document is constructed with a collision attack or not. Proper replace-

ments are proposed with different constructions and designs. The collision detection methodology helps to protect weak primitives from any possible collision attack using two approaches. The first approach employs a near-collision detection mechanism that was proposed by Marc Stevens. The second approach is the proposed work that employs two blocks calculation scheme. Moreover, this dissertation proposes a model that protects the secure hash functions from collision and length attack. The model employs the sponge structure to construct a hash. The resulting function is strong against the attacks and replaces the weak hash standards. Furthermore, to keep the general structure of the Merkle-Damgård functions, we propose a model that replaces the SHA-1 and SHA-2 hash standards using the Merkle-Damgård structure.

1.4. Contributions

In this thesis, we were able to show several contributions. First, we present a methodology to detect the collision attack. Then we present the proposed improvements that we applied to the hash function to circumvents collision and length extension attacks. Next, we show the application of used secure hash algorithms in the big data over the cloud.

1.4.1. Collision Detection of the SHA-1 Hash Function

We present two approaches to detect the collision attack before it takes place. We introduce a collision detection methodology and an improved version of the Secure Hash Algorithm (SHA-1). The proposed work helps to protect weak primitives from any possible collision attack. Two designs are implemented to help protect and improve the SHA-1 standard. The first design employs a near-collision detection approach that was proposed by Marc Stevens. The second design is the proposed work that employs the two-block calculation scheme. Both designs are tested and verified for examples of collided messages. The designs can detect collision probability and produce different hashes for weak messages.

1.4.2. Improving the Merkle-Damgård Hash Functions to Overcome Security Issues

We present a Sponge structure modulation of the MD5 and SHA-1 hash functions. The work employs the Keccak permutation functions to build the proposed scheme. The work discusses the main two security breaches that threaten the cryptography hash standards, which are collision and length extension attacks. Through analyzing several examples of collided messages of both algorithms (SHA-1 and MD5), we describe the potentials to overcome the collision and length extension attacks. Moreover, a proper replacement technique to avoid such attacks is discussed.

Moreover, we introduce an improved version of the secure hash algorithms, SHA-1 and SHA-2. The proposed work produces a strengthened secure hash function using the fusion between SHA-1 and SHA-2 hash standards. This design helps to protect the SHA-1 and SHA-2 against collision and length extension attacks. The fusion process is incorporated in the round steps of the hash functions using the function manipulators of the SHA-1 and SHA-2 algorithms. The proposed design is verified by the official test vectors that were confirmed by the National Institute of Standard and Technology. Moreover, the avalanche effect, hamming distance, and bit-hit properties were studied using different samples that were generated randomly. The proposed design gives over 52 percent of the avalanche effect and over 84 hamming distance. The proposed design was tested against collision attack and produced unique hash values for real collided examples and shows resistance against length extension attack.

1.4.3. Randomness Analyses of the Secure Hash Algorithms

We present a security analysis scheme for the most famous secure hash algorithms SHA-1 and SHA-2. Both algorithms follow Merkle-Damgård structure to compute the corresponding hash function. The randomness of the output hash reflects the strength and security of the generated hash. Therefore, the randomness of the internal rounds of the SHA-1 and SHA-2 hash functions is analyzed using Bayesian and odd ratio tests. Moreover, a proper replacement for both algorithms is proposed, which produces a hash output with more randomness level. The experiments were conducted using a high-performance computing testbed and CUDA parallel computing platform.

1.4.4. Big Data Applications Using Secure Hash Algorithms

On the cloud, data are not only stored on multiple locations, but also shared across multiple users. The integrity of data on the cloud is subject to doubt due to cloud failures. Therefore, many techniques are available to verify the existence of data on the cloud without retrieving the entire file. However, the data auditing techniques reveal confidential information to the designated verifier, which breaches the identity of the data uploaders. We introduce a privacy-preserving scheme for multiple data uploaders over the cloud. In the proposed design, the identities of the data block signers are hidden from a Third Party Auditor (TPA). A Secure Hash Algorithm (SHA) is also employed to provide an integral auditing report between the data owner(s) and the TPA. The proposed design is tested and verified mathematically and experimentally using a set of synthesized data.

The computational cost of data encryption and decryption grows with the increase of data on the cloud. The enhancements on the data processing techniques may affect data privacy. We introduce an anonymous privacy-preserving scheme for Big Data over the cloud. The proposed design helps to enhance the encryption/decryption time of Big Data by utilizing the MapReduce framework. The Hadoop distributed file system and the secure hash algorithm is employed to provide the anonymity, security and efficiency requirements for the proposed scheme. The experimental results show a significant enhancement in the computational time of data encryption and decryption.

We introduce a secured and distributed Big Data storage scheme with multiple authorizations. It divides the Big Data into small chunks and distributes them through multiple Cloud locations. The Shamir's Secret Sharing and Secure Hash Algorithm are employed to provide the security and authenticity of this work. The proposed methodology consists of two phases: the distribution and retrieving phases. The distribution phase comprises three operations of dividing, encrypting, and distribution. The retrieving phase performs collecting and verifying operations. To increase the security level, the encryption key is divided into secret shares using Shamir's Algorithm. Moreover, the Secure Hash Algorithm is used to verify the Big Data after retrieving from the Cloud. The experimental results show that the proposed design can reconstruct a distributed Big Data with good speed while conserving the security and authenticity properties.

1.5. Thesis Outline

The remainder of the thesis consists of the following chapters: Chapter 2 presents the main three secure hash algorithms and the corresponding Field Programmable Gate Arrays (FPGA) design. In particular, it provides background information about the SHA-1, SHA-2, and SHA-3 and their constructions. Moreover, the FPGA design comparison, between the three standards, is conducted.

In Chapter 3, we introduce the collision detection method based on two approaches. This chapter shows the ability to detect the collision attack before it takes place by applying Marc Stevens approach and our approach. Both approaches were able to detect the collision attack, but our design got the upper hand in terms of speed.

Chapter 4 presents a modification to the MD5 and SHA-1 hash standards. In this chapter, we show the implementation of both functions using Sponge Construction. The results show that the modification is helpful against the collision attack.

Chapter 5 presents a modified version of the SHA-1 and SHA-2 hash functions to circumvent collision and length extension attacks. In this chapter, we show the fusion between both algorithms can strengthen them. Moreover, we used a different padding method to prepare a message before compression.

Chapter 6 introduces a security analysis scheme for the most famous secure hash algorithms SHA-1 and SHA-2. Both algorithms follow Merkle-Damgård structure to compute the corresponding hash function. The randomness of the output hash reflects the strength and security of the generated hash. Therefore, the randomness of the internal rounds of the SHA-1 and SHA-2 hash functions is analyzed using Bayesian and odd ratio tests. Moreover, a proper replacement for both algorithms is proposed, which produces a hash output with more randomness level. The experiments were conducted using a high-performance computing testbed and CUDA parallel computing platform.

Chapter 7 introduces a secured and distributed Big Data storage scheme with multiple authorizations. It divides the Big Data into small chunks and distributes them through multiple Cloud locations. Shamir's Secret Sharing and Secure Hash Algorithm are employed to provide security and authenticity. The proposed methodology consists of two phases: the distribution and retrieving phases. The distribution phase comprises three operations of dividing, encrypting, and distribution. The retrieving phase performs collecting and verifying operations.

Chapter 8 introduces an anonymous privacy-preserving scheme for Big Data over the cloud. The proposed design helps to enhance the encryption/decryption time of Big Data by utilizing the MapReduce framework. The Hadoop distributed file system and the secure hash algorithm is employed to provide the anonymity, security and efficiency requirements for the proposed scheme. The experimental results show a significant enhancement in the computational time of data encryption and decryption.

Chapter 9 introduces a privacy-preserving scheme for multiple data uploaders over the cloud. In the proposed design, the identities of the data block signers are hidden from a Third Party Auditor (TPA). A Secure Hash Algorithm (SHA) is also employed to provide an integral

auditing report between the data owner(s) and the TPA. The proposed design is tested and verified mathematically and experimentally using a set of synthesized data.

Chapter 10 concludes the dissertation and provides future guidelines for possible research in this area of study.

1.6. References

[1] Ronald L Rivest. "The MD4 message digest algorithm". In: *Conference on the Theory and Application of Cryptography*. Springer. 1990, pp. 303–311.

[2] Ronald Rivest and S Dusse. *The MD5 message-digest algorithm*. Tech. rep. MIT Laboratory for Computer Science, 1992.

[3] Xiaoyun Wang and Hongbo Yu. "How to break MD5 and other hash functions". In: *Annual international conference on the theory and applications of cryptographic techniques*. Springer. 2005, pp. 19–35.

[4] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. "Finding collisions in the full SHA-1". In: *Crypto*. Vol. 3621. Springer. 2005, pp. 17–36.

[5] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov. "The first collision for full SHA-1". In: *Annual International Cryptology Conference*. Springer. 2017, pp. 570–596.

[6] Krystian Matusiewicz. *Analysis of Modern Dedicated Cryptographic Hash Functions*. Citeseer, 2007.

[7] Phillip Rogaway and Thomas Shrimpton. "Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance". In: *International workshop on fast software encryption*. Springer. 2004, pp. 371–388.

[8] Mihir Bellare and Thomas Ristenpart. "Multi-property-preserving hash domain extension and the EMD transform". In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2006, pp. 299–314.

[9] Mihir Bellare and Thomas Ristenpart. "Hash functions in the dedicated-key setting: Design choices and MPP transforms". In: *International Colloquium on Automata, Languages, and Programming*. Springer. 2007, pp. 399–410.

[10] Mario Cortina Borja and John Haigh. "The birthday problem". In: *Significance* 4.3 (2007), pp. 124–127. DOI: 10.1111/j.1740-9713.2007.00246.x. eprint: https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/j.1740-9713.2007.00246.x. URL: https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.1740-9713.2007.00246.x.

[11] Paul C Van Oorschot and Michael J Wiener. "Parallel collision search with cryptanalytic applications". In: *Journal of cryptology* 12.1 (1999), pp. 1–28.

[12] Eli Biham and Adi Shamir. "Differential cryptanalysis of DES-like cryptosystems". In: *Journal of CRYPTOLOGY* 4.1 (1991), pp. 3–72.

[13] Eli Biham and Adi Shamir. "Differential cryptanalysis of the full 16-round DES". In: *Annual International Cryptology Conference*. Springer. 1992, pp. 487–496.

[14] Xiaoyun Wang, Hongbo Yu, and Yiqun Lisa Yin. "Efficient collision search attacks on SHA-0". In: *Annual International Cryptology Conference*. Springer. 2005, pp. 1–16.

[15] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. "Keccak". In: *Annual international conference on the theory and applications of cryptographic techniques*. Springer. 2013, pp. 313–314.

[16] Marc Stevens, Arjen K Lenstra, and Benne De Weger. "Chosen-prefix collisions for MD5 and applications". In: *International Journal of Applied Cryptography* 2.ARTICLE (2012), pp. 322–359.

[17] Ralph C Merkle. "One way hash functions and DES". In: *Conference on the Theory and Application of Cryptology*. Springer. 1989, pp. 428–446.

[18] Ivan Bjerre Damgård. "A design principle for hash functions". In: *Conference on the Theory and Application of Cryptology*. Springer. 1989, pp. 416–427.

[19] Donald W Davies and Wyn L Price. *The application of digital signatures based on public key cryptosystems*. National Physical Laboratory Teddington, Middlesex, England, 1980.

[20] Ronald L. Rivest. "The MD4 Message Digest Algorithm". In: *Advances in Cryptology-CRYPTO' 90*. Ed. by Alfred J. Menezes and Scott A. Vanstone. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, pp. 303–311. ISBN: 978-3-540-38424-3.

[21] Patrick Gallagher and Acting Director. *Secure hash standard (shs)*. Tech. rep. National Institute of standards and technology, FIPS PUB, 1995, p. 183.

[22] PUB FIPS. "180-1. secure hash standard". In: *National Institute of Standards and Technology* 17 (1995), p. 45.

[23] National Institute of Standards and technology. "Secure Hash Standard". In: *FIPS PUB and FIPS (180-2)* 180 (2002).

[24] NIST FIPS Pub. *180-3: Federal Information Processing Standards Publication, Secure Hash Standard (SHS)*. Tech. rep. Technical report, National Institute of Standards and Technology, 2008.

[25] National Institute of Standards and Technology. "Secure Hash Standard (SHS) 180-4". In: *FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION* (Mar. 2012).

[26] Antoon Bosselaers and Bart Preneel. *Integrity Primitives for Secure Information Systems: Final Ripe Report of Race Integrity Primitives Evaluation*. 1007. Springer Science & Business Media, 1995.

[27] Hans Dobbertin, Antoon Bosselaers, and Bart Preneel. "RIPEMD-160: A strengthened version of RIPEMD". In: *International Workshop on Fast Software Encryption*. Springer. 1996, pp. 71–82.

[28] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. "The keccak sha-3 submission". In: *Submission to NIST (Round 3)* 6.7 (2011), p. 16.

[29] Bertoni Guido, Daemen Joan, P Michaël, and VA Gilles. *Cryptographic sponge functions*. Tech. rep. Keccak.Team Official Site, 2011, pp. 1–93.

[30] Eli Biham and Orr Dunkelman. *A Framework for Iterative Hash Functions—HAIFA*. Tech. rep. Computer Science Department, Technion, 2007.

[31] Elena Andreeva, Bart Mennink, and Bart Preneel. “Security reductions of the second round SHA-3 candidates”. In: *International Conference on Information Security*. Springer. 2010, pp. 39–53.

[32] Masumeh Damrudi and Norafida Ithnin. “Parallel RSA encryption based on tree architecture”. In: *Journal of the Chinese Institute of Engineers* 36.5 (2013), pp. 658–666.

2. BACKGROUND

2.1. Overview

In cryptography, integrity is provided with the hash functions. The hash functions compress the message to assist the process of the digital signatures. In the last decades of the 20^{th} century, we witnessed Message Digest 4 (MD4) developed by Ronald Rivest, for the provision of integrity [1]. The MD4 provided the basis for MD5, that is a strengthened version of MD4 [2]. Both the MD4 and the MD5 worked to compute 128 bits digest. However, weakness in aforesaid algorithms resulted in the development of the Secure Hash Algorithm (SHA-0) by the National Institute of Standards and Technology (NIST) in 1993. The SHA-0 was followed by SHA-1 in 1995 with the hash size of 160 bits. The SHA-1 replaced MD5 that was exposed to collision attacks by then [3]. As with all of the security protocols, the SHA-1 was also under constant scrutiny by the security experts. Wang *et al.* [4] claimed that collisions in SHA-1 can be found with complexity less than 2^{80} compression function calculations. Recently the claim of Wang *et al.* was validated by Marc *et al.* [5] by finding hash collisions for different files. Because SHA-1 was prone to the collision attack, theoretically (as shown by Wang *et al.*), development of SHA-2 started in 2001. The SHA-2 has different flavors, such as SHA-256, SHA-384, and SHA-512 with hash values of 256, 384, and 512 bits, respectively [6]. Later on, SHA-224 was also introduced in 2004 to provide security strength of 3DES. The FIPS-180-3 defines the aforementioned four algorithms to be the part of SHA-2 standard [7].

The SHA-1 and SHA-2 follow Merkle Damgard (MD) structure. The MD structure takes the message of pre-defined size and subsequently divides it into equal size blocks. The final hash is computed using the dedicated compression function [8]. As well as SHA-1 and SHA-2 followed the same structure model, this created a fear among the experts that SHA-2 will also be exposed to the same kind of attacks. Therefore, NIST held a competition for the selection of a new hash standard (SHA-3). There were 64 submissions in the competition. Round 1 reduced the number

The content of this chapter has been submitted to the ACM Computing Surveys. The material in this chapter was co-authored by Zeyad Al-Odat, Mazhar Ali, Assad Abbas, and Samee Khan. Zeyad Al-Odat had primary responsibility for conducting experiments and collecting results. Zeyad Al-Odat was the primary developer of the conclusions that are advanced here. Zeyad Al-Odat also drafted and revised all versions of this chapter. Mazhar Ali and Assad Abbas drafted and revised all versions of this chapter. Samee U. Khan served as proofreader.

of algorithms to 51. The second round of filtration selected 14 candidate algorithms. Finally, there were five algorithms (Grostl, BLAKE, JH, Keccak, and Skein) in the decisive round. In October 2012, Keccak was announced as a new standard becoming SHA-3 [9, 10, 11, 12].

The SHA-3 was different from the rest of the SHA family in a sense that it follows a sponge construction instead of Merkle-Damgård.Sponge construction follows an absorbing squeezing structure where data is read in and processed in the absorbing phase followed by squeezing phase that gives the output. Besides the working model, SHA-3 also differs in the length of message size and hash length. More details for each of the aforementioned standards will be discussed in Section 2.2.

To distinguish between different hash standards, a fair comparison need to be applied. Therefore, hardware was one choice to implement and compare different hash standards. Field Programmable Gate Array (FPGA) is the best choice for hardware implementation of hash algorithms due to flexibility and adaptability [13]. Moreover, the FPGA is faster than the Central Processing Unit (CPU) and Graphic Processing Unit (GPU) [14]. FPGA is usually part of the Application Specific Integrated Circuit (ASIC), which implement a predefined function(s). ASIC is used when the speed matters most [15]. For instance, hardware security modules commonly use ASICs to accelerate the execution of cryptographic operations (like AES) [16]. Besides CPUs, GPUs, and ASICs, Hybrid Hardware System (HHS) which is a combination of different types of aforementioned hardware, can also be utilized. However, HHS tends to increase the price of the resulting modules. More details will be discussed in Section 2.3.

Unlike other works, [17, 18, 19, 20], this chapter not only discusses SHA families but also focuses on FPGA optimization techniques for implementing the SHA. In this survey special attention is paid to the FPGA optimization techniques of three hash standards (SHA-1, SHA-2, SHA-3). The study covers several types of optimization techniques and highlights their contributions to the performance metrics of FPGAs. Moreover, this survey discusses the pros and cons of each of the optimization methods and investigates their effects on the performance. Furthermore, the chapter outlines various methods of optimization and organizes them according to their contribution to the SHA design.

On the other hand, previous surveys on the same area provide a specific explanation about the SHA standards and provide concepts of deploying SHA standards using the FPGA. The work

presented in [17] provides a survey and analysis of the hash functions as a combination between stream cipher and hash function. Their work focuses on the Linear Feedback Shift Register (LFSR) hash block. Jarvinen *et al.* conducted a comparative survey of high-performance cryptography algorithms [18]. This study analyzes the FPGA-based implementations of the most widely used cryptography implementations (AES, IDEA, DES, and SHA). However, the hash algorithm's analyses were minor and only includes MD5, SHA-1, and part of the SHA-2. A theoretical survey on the secure hash algorithms is presented in [19]. The work shows the different kind of hash algorithms and their applications. Chaves *et al.* presented a work that analyzes the implementations of the secure hash algorithms SHA-1 and SHA-2 on FPGA [20]. The work studied the possibilities to improve the SHA-1 and SHA-2 hash functions by properly explore the data dependencies and the hardware reuse. Moreover, the study compares between the FPGA implementations of the SHA-3 finalist (Blake, JH, Skein, Gr∅stl, and Keccak). The study showed that the Keccak outperforms the other candidates in term of throughput and area. In this survey, a comprehensive study of FPGA-based implementation of the secure hash algorithms (SHA-1, SHA-2, and SHA-3) family is carried out. Besides, we discuss the design parameters and optimization techniques in more detail.

The rest of chapter is organized as follows: Section 2.2 describes SHA standards and their functions. The hardware implementation of SHA and FPGA's performance metrics are discussed in Section 2.3. A literature review of different optimization techniques is presented in Section 2.4. Chapter discussion and analysis are carried out in Section 2.5.

2.2. Secure Hash Algorithm Families

SHA takes a message with an arbitrary size, then through some calculations produces the message hash[1] [21]. The process is defined in Eq. (2.1).

$$h = H(M), \tag{2.1}$$

where M is the input message, and h is the generated digest using the hash algorithm H.

Different parameters of the SHA family are compared in Table 2.1. SHA-1 accepts messages of size less than 2^{64}, divides it into equal size blocks of 512-bit each, processes it through 80 steps round computations and provides the final hash of 160 bits. The SHA-2 follows the same structure

[1]Some references use the phrase "digest".

as SHA-1, but it differs in the diversity of the output. The SHA-2 has four different output options (number beside the hash represents the output length), with extra two options that take the truncated value of SHA-2/512. On the other hand, the SHA-3 follows a different structure model but has the same options for the output hash as the SHA-2 standard with extra two extensible output functions. The number beside each standard reflects the output hash, but for SHAKE-128 and SHAKE-256 the number reflects the security level that each one of them supports against brute force attack.

Table 2.1. The secure hash algorithms SHA-1, SHA-2, and SHA-3 and their corresponding parameters

SHA Family				
Algorithm	**Output**	**Block Size (bit)**	**Max Msg Size (bit)**	**Internal Round**
SHA-1	160	512	$2^{64} - 1$[22]	80
SHA2				
224	224	512	$2^{64} - 1$[6]	64
256	256	512	$2^{64} - 1$[6]	64
384	384	1024	$2^{128} - 1$[6]	80
512	512	1024	$2^{128} - 1$[6]	80
512/224	224	1024	$2^{128} - 1$[6]	80
512/256	256	1024	$2^{128} - 1$[6]	80
SHA3				
224	224	1152	unlimited	24
256	256	1088	unlimited	24
384	384	832	unlimited	24
512	512	576	unlimited	24
SHAKE-128	Arbitrary[12]	1344	unlimited	24
SHAKE-256	Arbitrary[12]	1088	unlimited	24

For any hash algorithm, the following properties must hold to consider it as secure:

(a) Preimage resistance: a property of easily get the hash from a given message but difficult to extract the message back from a given hash.

(b) 2^{nd} preimage resistance: means that it is difficult to find two messages that both generate the same hash.

(c) Collision resistance: the property of resisting the probability to generate the same hash for two different messages or more.

2.2.1. SHA-1

The SHA-1 was proposed after MD5 hash algorithm. Despite the collision attack announced in 2017 by Stevens *et al.* [5], it is convenient to look into the details of SHA-1 because SHA-1 is still used by many entities and applications. The SHA-1 follows Merkle Damgard (MD) structure model. With 160 bits output hash, the SHA-1 goes through several steps and compression operations before the output hash is produced. [8]. Details about SHA-1 will be explored in the following text.

(a) **Message Padding:** In this step a '1' is appended at the end of input message and followed by least number of 0's until it congruents to (448 *mod* 512). The size of the original message M is appended as big-endian 64-bits, as seen in Fig. 2.1. The resulting message becomes: $\widehat{M} = N \times 512$, for real $N \geqslant 1$.

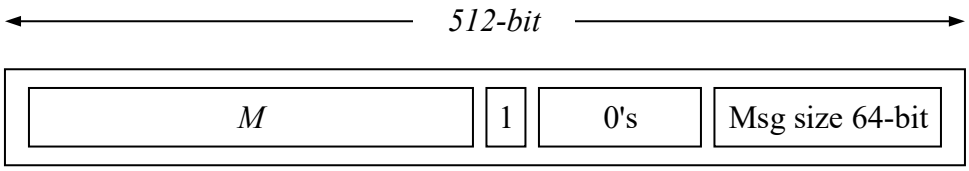

Figure 2.1. Message padding mechanism of the Merkle-Damgård functions

(b) **Message divide:** in this step, the padded message $\widehat{M}$ is divided into N 512-bit blocks (M_0, M_1,...., M_{N-1}).

(c) **Initial Hash Values (IHV):** SHA-1 needs five 32-bits Initial Hash Values ($IHVs$) $H0$, $H1$, $H2$, $H3$ and $H4$, initialized with fixed values ($\texttt{67452301}_{16}$, $\texttt{EFCDAB89}_{16}$, $\texttt{98BADCFE}_{16}$, $\texttt{10325476}_{16}$, $\texttt{C3D2E1F0}_{16}$), respectively. Moreover, five 32-bit working state variables (A, B, C, D, E) initialized with the values of $IHVs$, accordingly.

(d) **Processing:** To compute the hash value of N blocks message, the process goes through SHA-1 compression function for $N + 1$ states (state for each block plus the initial state). Eq. 2.2 shows that the $IHVs$, of the current message block, are used to compute the $IHVs$ for next block[2].

$$IHV_i = SHA1\_Compress(IHV_{i-1}, M_{i-1}), \tag{2.2}$$

[2]In case of the current message block is the last block then the resulting $IHVa$ conform the final hash value.

where $SHA1\_Compress(IHV, M)$ is the compression function with inputs of 32-bit words, IHV_i are the intermediate hash values (A, B, C, D, E), and M is the 512-bits message block (B).

The compression function consists of 80 steps (0 to 79), divided into four consecutive rounds of 20 steps each. Every step t uses modular addition, left rotation, round function $\mathbf{F}_t$, and round constant $\mathbf{K}_t$, as shown in Table 2.2. The message Block B is partitioned into sixteen consecutive words $(m_0, m_1.....m_{15})$, then expanded to 80 words W_t, according to Eq. 2.3.

$$W_t = \begin{cases} M_t & ,0 \leq t \leq 15 \\ (W_{t-16} \oplus W_{t-14} \oplus W_{t-8} \oplus W_{t-3} \oplus)_{\lll 1} & ,16 \leq t, \leq 79, \end{cases} \tag{2.3}$$

where $(\oplus)$ is logic XOR operation, and $(\lll)$ is the cyclic shift left operation.

The working state variables (A, B, C, D, E) change after each step according to (2.4), (2.5), (2.6), (2.7), and (2.8).

$$A_t = RL^5(A_{t-1}) \boxplus F_t(B_{t-1}, C_{t-1}, D_{t-1}) \boxplus E_{t-1} \boxplus W_t \boxplus K_t, \tag{2.4}$$

$$B_t = A_{t-1}, \tag{2.5}$$

$$C_t = RL^{30}(B_{t-1}), \tag{2.6}$$

$$D_t = C_{t-1}, \tag{2.7}$$

$$E_t = D_{t-1}, \tag{2.8}$$

where $(\boxplus)$ represents modular addition, and $(RL^n(X))$ is the left rotation of variable (X) by (n) bits.

(e) **The output hash:** after all steps are computed, the resulting working state values are added to the intermediate hash values before processing according to (2.9),(2.10),(2.11),(2.12), and (2.13).

$$A = A_0 + A_t, \tag{2.9}$$

$$B = B_0 + B_t, \tag{2.10}$$

$$C = C_0 + C_t, \tag{2.11}$$

$$D = D_0 + D_t, \tag{2.12}$$

$$E = E_0 + E_t \tag{2.13}$$

Table 2.2. The SHA-1 round functions and constants

Round and Steps	**Round Function** $F(B, C, D)$	**Round Constant** (k_t)
Round1 (0-19)	$(B \wedge C) \vee (\neg B \wedge D)$	`0x5A827999`
Round2 (20-39)	$(B \bigoplus C \bigoplus D)$	`0x6ED9EBA1`
Round3 (40-59)	$(B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$	`0x8F1BBCDC`
Round4 (60-79)	$(B \bigoplus C \bigoplus D)$	`0xCA62C1D6`

For the last block, the output hash is the concatenation $(A \parallel B \parallel C \parallel D \parallel E)$ of the five hash values together to form 160-bit hash [7]. Alternatively, the new $IHVs$ will be fed as new $IHVs$ for the next block calculation.

2.2.2. SHA-2

The SHA-2 has four fixed output standards (224, 256, 384, 512), and two truncated versions (SHA-512/224, SHA-512/256). The SHA-224 and SHA-256 work on 512-bit block size, with 16-words of 32-bit each. While SHA-384 and SHA-512 (and the truncated versions) work on a 1024-bit block with 16-words of 64-bit words size. The SHA-2 has 8 working state variables (a, b, c, d, e, f, g, h), each of size equal to the word size of respective flavor [6].

Like SHA-1, SHA-2 performs padding process first by adding 1 and 0's to the end of the message followed by the message size. Then divides it into 16 equal size blocks according to the desired output hash as depicted in Table 2.1. Afterward, expands the message into 64 blocks using

SHA-2 expansion equations (2.14), (2.15), and (2.16), for $t = 16$ to n,

$$\sigma_0 = ROTR^{r1}(W_{t-15}) \oplus ROTR^{r2}(W_{t-15}) \oplus SHR^{r3}(W_{t-15}) \tag{2.14}$$

$$\sigma_1 = ROTR^{q1}(W_{t-2}) \oplus ROTR^{q2}(W_{t-2}) \oplus SHR^{q3}(W_{t-2}) \tag{2.15}$$

$$W_t = W_{t-16} + \sigma_0 + W_{t-7} + \sigma_1, \qquad 16 \leq t \leq n \tag{2.16}$$

where $ROTR^n(X)$ rotates word X to the right by n bits, and $SHR^n(X)$ shift right word X by n bits. For SHA-224 and SHA-256 $n = 63$, $r1 = 7$, $r2 = 18$, $r3 = 3$, $q1 = 7$, $q2 = 19$, and $q3 = 10$, while $n = 79$, $r1 = 1$, $r2 = 8$, $r3 = 7$, $q1 = 19$, $q2 = 61$, and $q3 = 6$ for SHA-384 and SHA-512.

The eight working state variables (a, b, c, d, e, f, g, h) are initialized with fixed hexadecimal values defined by the SHA-2 definition[3][10]. Two common functions called Choose (Ch) and Majority (Maj), which are the SHA-2 manipulator equations (2.17) and (2.18),

$$Ch(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z), \tag{2.17}$$

$$Maj(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z), \tag{2.18}$$

where symbols: $\oplus$, $\wedge$ and $\neg$: are the logical XOR, AND, and NOT operations, respectively.

To further strengthen the SHA-2 hash function, two more functions are used $\text{Sum}_0(\sum_0)$ and $\text{Sum}_1(\sum_1)$ as depicted in (2.19) and (2.20),

$$\sum\nolimits_0(x) = ROTR^{r1}(x) \oplus ROTR^{r2}(x) \oplus ROTR^{r3}(x), \tag{2.19}$$

$$\sum\nolimits_1(x) = ROTR^{q1}(x) \oplus ROTR^{q2}(x) \oplus ROTR^{q3}(x), \tag{2.20}$$

where $r1 = 2$, $r2 = 13$, $r3 = 22$, $q1 = 6$, $q2 = 11$, and $q3 = 25$ for SHA-224 and SHA-256, while $r1 = 28$, $r2 = 34$, $r3 = 39$, $q1 = 14$, $q2 = 18$, and $q3 = 41$ for SHA-384 and SHA-512.

Algorithm 1 shows the full SHA-2 computations. The eight working state variables[4] W_t (a, b, c, d, e, f, g, h) change after each step as can be seen in the Algorithm 1. The longest data path is the one corresponding to the a value as it contains seven terms to be modified after each step.

[3]Each flavor of SHA-2 has designated IHV_s differ from other

[4]T1 and T2 are temporary variables used to calculate the value of a.

Moreover, round constant K_t appears at the same step[5]. The value of j corresponds to the number of iterations (rounds) performed by the algorithm. For SHA-224 and SHA-256 j=64, whereas for SHA-384 and SHA-512 j=80. After processing all n blocks of message M, the final message digest is obtained by concatenating all or parts of hash values $H_0^i, ..., H_7^i$. The message digest for each version of SHA-2 algorithm is given by the concatenation symbol (||).

2.2.3. SHA-3

The SHA-3 was published in 2012, after a competition that was held by the NIST. Five candidates were selected for the final round (Keccak, Grøstl, BLAKE, JH, and Skien). Keccak won the competition as the next standard for the SHA-3 [10]. Unlike the previous standards (SHA-1 and SHA-2), the SHA-3 relies mainly on absorb and squeeze structure [23, 24], as shown in Fig. 2.2.

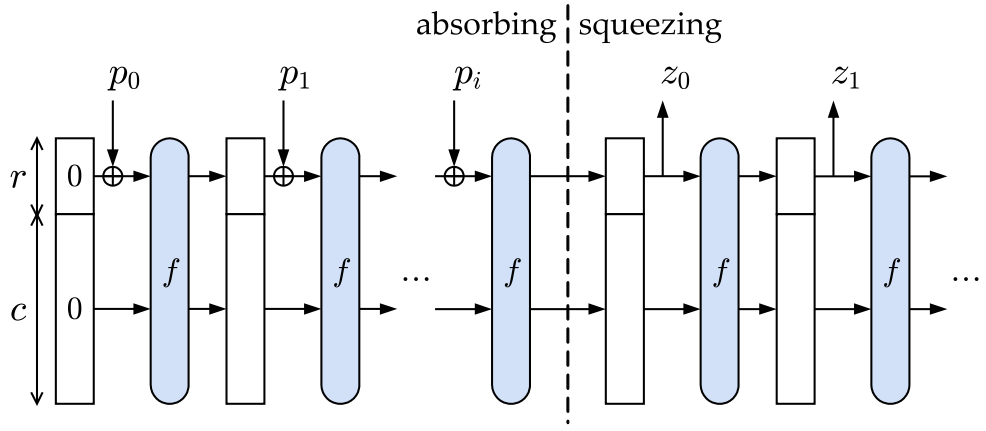

Figure 2.2. General structure of the sponge construction.

A preprocessing phase is accomplished before starting with the sponge operations. The message is padded first, then divided into equal size blocks (P_i) each of r bits size [11]. In the absorbing phase, the state of b-bit is initialized with 0's. Each state is represented by two values, bit-rate (r) and capacity (c), where, r is the block size P_i, and c complete the state size to the desired permutation level. The XOR operations are carried out for the message blocks P_i with r bits of the state. A permutation function f is applied to get next state value [23]. In the squeezing

[5]Round constant K_t represents a fixed value, defined by SHA-2 definition. Each flavor of SHA-2 has its own K_t as described in [10].

Algorithm 1: SHA-2 compression function

Input: Padded Message $M = \{M_0, M_1 \ldots, M_n\}$ Blocks
Output: Output Hash (224 or 256 or 384 or 512)

Initialize_IHV :=
$a = H_0^{(i-1)}$ $b = H_1^{(i-1)}$ $c = H_2^{(i-1)}$ $d = H_3^{(i-1)}$
$e = H_4^{(i-1)}$ $f = H_5^{(i-1)}$ $g = H_6^{(i-1)}$ $h = H_7^{(i-1)}$
for $t \leftarrow 0$ ***to*** n **do**
  if $t < 16$ **then**
    $W_t \leftarrow M_t$
  else
    $W_t \leftarrow \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16}$
for $t \leftarrow 0$ ***to*** $j - 1$ **do**
  $T_1 = h + \sum_1(e) + Ch(e, f, g) + K_t + W_t$
  $T_2 = \sum_0(a) + Maj(a, b, c)$
  $h = g$
  $g = f$
  $f = e$
  $e = d + T_1$
  $d = c$
  $c = b$
  $b = a$
  $a = T_1 + T_2$
for $t \leftarrow 0$ ***to*** n **do**
  $H_0^{(t)} = a + H_0^{(t-1)}$
  $H_1^{(t)} = b + H_1^{(t-1)}$
  $H_2^{(t)} = c + H_2^{(t-1)}$
  $H_3^{(t)} = d + H_3^{(t-1)}$
  $H_4^{(t)} = e + H_4^{(t-1)}$
  $H_5^{(t)} = f + H_5^{(t-1)}$
  $H_6^{(t)} = g + H_6^{(t-1)}$
  $H_7^{(t)} = h + H_7^{(t-1)}$
return *Hash*
SHA-224 $\leftarrow H_0^{(n)} \| H_1^{(n)} \| H_2^{(n)} \| H_3^{(n)} \| H_4^{(n)} \| H_5^{(n)} \| H_6^{(n)}$
SHA-256,512 $\leftarrow H_0^{(n)} \| H_1^{(n)} \| H_2^{(n)} \| H_3^{(n)} \| H_4^{(n)} \| H_5^{(n)} \| H_6^{(n)} \| H_7^{(n)}$
SHA-384$\leftarrow H_0^{(n)} \| H_1^{(n)} \| H_2^{(n)} \| H_3^{(n)} \| H_4^{(n)} \| H_5^{(n)}$

phase, the output hash (z) is taken from z_0 for the fixed size hashes (224, 256, 384 and 512). An extra permutation is applied in case of arbitrary length as in (SHAKE128 and SHAKE256).

The SHA-3 Keccak supports two modes of operations. A fixed output hash mode support hashes (224, 256, 384, 512), and a variable-length hash mode (SHAKE128 and SHAKE256) produces an output of variable length according to the desired application[6]. In Keccak, b represents the permutation function's level (25, 50, 100, 200, 400, 800, 1600). Yet, the most common permutation is ($b = 1600$, $l = 6$), where b is computed using Eq. (2.21).

$$b = 25 * 2^l \qquad , l = 0, 1, 2, 3, 4, 5, 6 \tag{2.21}$$

In the fixed mode operation, two groups are allowed. Group-1 ($r = 1344$,$c = 256$) is used to compute hash values of length 224 and 256, while group-2 ($r = 1088$,$c = 512$) is used to calculate the hashes of length 384 and 512 [25].

The fixed-size output hash is taken from the first step of the squeezing phase (z_0) by selecting the least significant bits (according to the desired hash output 224, 256, 384 and 512). When variable length hash is required, all bits of Z can be used according to the desired output. Moreover, the output can be taken from any Z_i [10].

The function of Keccak is depicted in Fig.2.3. The state $b = r + c$ is initialized with 0's. The length of b depends on the level of the permutation selected. As mentioned earlier, $b = 1600$ is the most commonly used permutation. Each block is processed through several rounds that are determined by the l value, according to Eq. (2.22).

$$Rounds = 12 + 2l, \qquad l = (0, 1, 2, 3, 4, 5, 6) \tag{2.22}$$

Keccak handles the state b as a 3D-Matrix ($A \times B \times C$), which is shown in Fig. 2.4. Each of Keccak's rounds has distinct constant $RC[i]$ used inside permutation function. Fig 2.3, shows that each round consists of five steps denoted by Greek letters, $theta(\theta)$, $rho(\rho)$, $pi(\pi)$, $chi(\chi)$ and $iota(\iota)$. Each step manipulates the state matrix ($A \times B \times C$). The values of A and B are fixed to

[6]The number (128 and 256) reflects the security strength of each one of them.

5, while the value of C is represented by w according to Eq. (2.23).

$$w = 2^l, \qquad l = (0, 1, 2, 3, 4, 5, 6) \tag{2.23}$$

The resultant b Matrix: $b = 5 \times 5 \times w$ bits. If l chosen to be 6, then the matrix becomes $5 \times 5 \times 64 = 1600$ bits, consequently number of rounds will be 24.

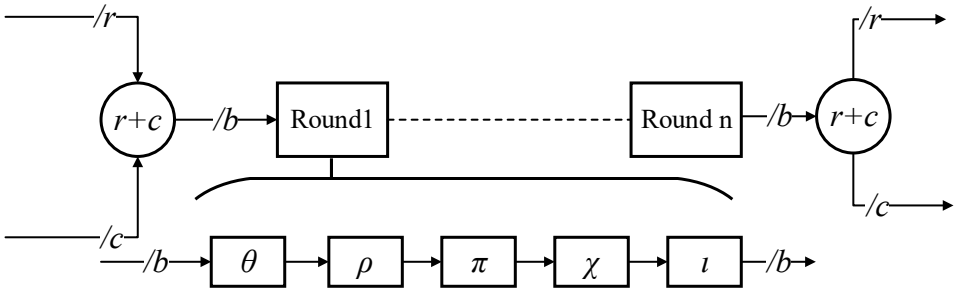

Figure 2.3. Keccak round steps theta (θ), rho (ρ), pi (π), chi (χ), and iota (ι)

- *Theta* (θ) step: which operates on a 2D-Array (5x5), where each element contains w bits. a single 5×5 array can be seen as a slice, where the 1D-array of w bits is a lane, as shown in Fig. 2.4. Theta step manipulates the state array according to Eq. (2.24). Where $C[x]$ and $D[x]$ represent lanes and $A[x, y]$ represents slice. Theta computes the parity of each column, then combines them with the XOR operator using Eq. (2.24).

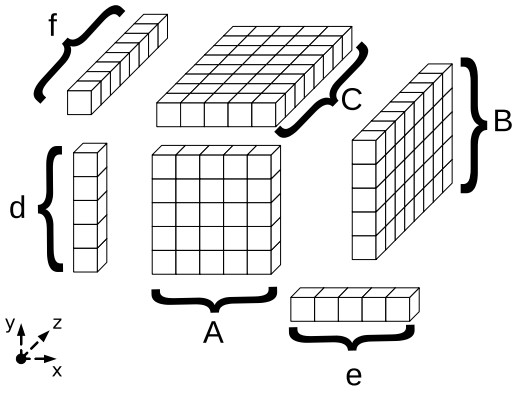

Figure 2.4. Keccak state matrix ($A \times B \times C$), represented as 3D-Matrix. Each square represents one bit, A)slice, B)sheet, C)plane, d)column, e)row, f)lane.

$$step(\theta) = \begin{cases} C[x] = A[x,0] \oplus A[x,1] \oplus A[x,2] \oplus A[x,3] \oplus A[x,4] & ,x = 0,1,2,3,4 \\ D[x] = C[x-1] \oplus ROT(C[X+1],1) & ,x = 0,1,2,3,4 \\ A[x,y] = A[x,y] \oplus D[x] & ,x = 0,1,2,3,4 \end{cases} \quad (2.24)$$

- *Rho* (ρ) step, this step rotates one element (lane) of the state matrix $A[x,y]$ (which is 5×5) by i bits, as seen in Eq. (2.25). The rotation offset value denoted by $r[x,y]$ is a constant value assigned according to Table 2.3.

$$step(\rho) = ROT(A[x,y], r[x,y]), \qquad (x,y) = 0,1,2,3,4 \quad (2.25)$$

Table 2.3. Values of the ρ step constants r[x,y] of the Keccak

	$x=3$	$x=4$	$x=0$	$x=1$	$x=2$
$y=2$	25	39	3	10	13
$y=1$	55	20	36	44	6
$y=0$	28	27	0	1	62
$y=4$	56	14	18	2	61
$y=3$	21	8	41	54	15

- *Pi* (π) step: is a complement step to *rho*, as it takes the rotated lanes from *rho* step and put them in different positions in the array matrix ($B[x,y]$), without modifying any value. It only permutes the matrix according to Eq. (2.26),

$$step(\pi) \Rightarrow B[y, 2x+3y] = ROT(A[x,y], r[x,y]), \quad (2.26)$$

where $x, y = 0,1,2,3,4$.

- *Chi* (χ) step: in this step the B matrix, which was generated from the previous step, is manipulated according to Eq. (2.26) and put the result back in array matrix A according to Eq. (2.27).

$$step(\chi) \Rightarrow A[x,y] = B[x,y] \oplus ((\bar{B}[x+1,y]) \wedge B[x+2,y]), \quad (2.27)$$

where $x, y = 0,1,2,3,4$, $\wedge$ bit-wise AND operation, and $\bar{B}[]$ is the bit-wise complement of B.

- *Iota* (ι) step: adds the round constant $RC[i]$ to the state matrix A at location $A[0,0]$, where each round has a distinct 64-bit round constant. *Iota* step represented by Eq. (2.28).

$$step(\iota) \Rightarrow A[0,0] = A[0,0] \oplus RC[i], \tag{2.28}$$

where $RC[i]$ are 24 different 64-bit round constants, they were depicted from reference [26].

2.3. Hardware Implementations of the SHA Standards

Hash algorithms can be implemented in software and hardware. However, performance in hardware becomes an important criterion for breaking a tie between algorithms; when all the security-related parameters are equally good. In the subsequent text, a discussion about different hardware that are used to implement SHA standards will be presented.

2.3.1. Choice of Hardware to Implement SHA

Different factors are taken into consideration to choose hardware for implementation. Table 2.4 provides a summary of different choices to implement cryptography protocols along with their advantages and disadvantages.

Table 2.4. Different choices for hardware implementations of secure hash algorithms

Hardware	Integration	Property	Advantage	Disadvantage
CPU	Desktop or laptop computers	Number of Cores	Testing Using Software	Side Channel Attack General Purpose
GPU	Desktop ,Laptop or Accelerated Supercomputer	Number of Cores	Parallel computing and tested using software	Difficult Programming
FPGA	Stand alone part of ASIC	Application Specific	Easy to optimize , Reconfigurable	Experience in HDL
ASIC	Stand alone	Application Specific	low price, reconfigurable	Pre-planning before Design
HSS	Stand-alone or embedded	combined design	chose the advantage of the best hardware	price and integration.

Central Processing Unit (CPU) is used in desktop and laptop computers. The main property of CPUs is the number of cores inside the processor. As CPUs are general purpose units, cryptography applications share the CPU resources with other applications. Graphics Processing Unit (GPU) is part of general-purpose units that can be found in desktop, laptop, or accelerated supercomputers. The GPUs run algorithms in parallel, thus giving more computation power than CPUs. Field Programmable Gate Arrays (FPGAs) are part of Applications Specific Integrated Circuit (ASIC). The FPGAs are used to implement a predefined function. Moreover, the FPGA is

considered as reconfigurable hardware that is configured at the end-user by changing the layout of implementations. Application Specific Integrated circuit (ASIC) is manufactured to run exactly one application. Because ASIC is small in size it is used when processing speed is important. Hybrid Hardware System (HHS) is the composition of different types of aforesaid hardware, and integrate them to work together to form one powerful cryptographic system. The HHS gives better results in term of performance but increases the price consequently [27, 28, 16, 29].

Different factors need to be considered when choosing any hardware; security, speed, and price. The security represents the resistant of a hardware against any kind of attacks. CPUs and GPUs are vulnerable for channel side attacks, while the FPGAs are more secure because they run for a specified process as ASIC. The second factor is the speed where it measures the time needed to finish the job. CPUs are capable of performing several operations and so can't be too much optimized in one direction. The FPGAs are faster than CPUs and GPUs because they are dedicated to a specific task. ASICs are the best in term of speed as they are flexible for design, however, the small area limits it from being used for bigger tasks. The third factor is the price. CPUs are easy to obtain, cheap to be programmed, and get the programs run quickly. The GPUs are also quite easy to obtain, a bit more expensive to effectively program and capable of efficient execution of programs. The FPGAs are more expensive and require the design of algorithm using hardware description language (VHDL, Verilog), but once programmed they are easily reconfigured. The ASICs have a long design cycle, but once completed they can be manufactured easily and for a low price [30, 14]. The choice of FPGAs arises from the fact that they support different type of design methodologies namely: software design methodology, hardware design methodology, and software with hardware design methodology. The software design methodology is supported using the Environment Development Kit (EDK) tool that comes with the FPGA hardware. The hardware design methodology is the ability of FPGA to support different level of programming languages (e.g., VHDL, C++). While software with hardware methodology is the ability of FPGA to integrate software and hardware. Because FPGA is reconfigurable, the designer can reuse it and makes fast prototyping, which reflected to cost-free mistakes [31]. According to the aforementioned factors, the FPGA is the best choice for implementing and optimizing the SHA.

2.3.2. FPGA Performance Metrics

The FPGAs can be as simple as a logic gate or complicated architecture as a microprocessor. The FPGA architecture is based on one of five components: transistors, logic gates, multiplexers, lookup tables (LUTs), and wide-fanin (AND-OR) structure. To explore more about FPGA the reader advised reading [32].

With the increased usage of FPGA, optimizing the FPGA is crucial and the newly published SHA standards require speed and memory [33]. Therefore, to measure the performance of FPGA, the following metrics need to be taken into consideration:

1. Area: Reflects the total number of Configurable Logic Block (CLB) or Look Up Tables (LUTs) used for any design implementation. However, in some cases, the area comparison might be an unfair factor, because of the differences in placements methodology of different FPGA manufacturers. Therefore, to compare area requirements, the same FPGA's manufacturer is recommended[33].

2. Frequency: The operational clock frequency that a given FPGA can run.

3. Throughput: Used to measure the speed of hardware implementation, which represents the number of bits processed in a given time. Throughput defined by Eq. (2.29),

$$Throughput = \frac{Block\_Size}{T * N_{clk}}, \tag{2.29}$$

   where $Block\_Size$ is the total number of bits processed, T is the time needed to process a block, and N_{clk} is the number of clock cycles.

4. Throughput to Area ratio: The comparison between performance metrics is recommended according to a fair factor. Taking the ratio of throughput to the area requirements can provide a fair comparison between different kinds of FPGAs. [34].

5. Power Consumption: Reflects the total amount of power consumed by the hardware when applying a specific design. Conventionally, this property is defined by the frequency level.

The leading manufacturers of FPGA market are Xilinx and Altera (now Intel). Nowadays, both of them control more than 90% of the FPGA market [35]. Therefore, the majority of hardware implementations of secure hash algorithms are deployed using Xilinx and Altera FPGAs.

SHA hash standards were discussed in different works. Some of these works were dedicated to specific SHA standards [36, 37, 38, 39], while the other compare more than one standard in their works [20]. In the next section, we categorize all optimization techniques of different hash standards on FPGA according to their influence on the optimization metrics. For each hash standard, the effects of optimizations will be addressed comprehensively.

2.4. Optimization Techniques

The equations and structures of the SHA algorithms showed that all operations were based on mathematical and logical functions. Therefore, improving these operations will enhance the computational resources that are used during the hardware deployment. For instance, addition is the core math operation in the SHA-1 and SHA-2, so enhancing the adder units will lead to a significant enhancement to the system [40]. This survey categorizes the FPGA optimization techniques of SHA algorithms according to the type of optimization. The optimization techniques of FPGAs are laid in one of the following categories:

1. Carry Save Adder (CSA): is a small and fast digital adder, used in implementations to compute the sum operation of binary digits. CSA produces an output of size same as the size of input words [41]. CSA enhances the area and maintains throughput in some cases.

2. Pipelining: is a data processing technique, that process a series of connected elements in parallel on a timed fashion [42]. Pipelining combines multiple steps into one step unless they have data dependency between them. Pipelining can be used with all hash standard's optimization methods. It provides significant improvements in term of throughput along with maintaining area requirements in some cases.

3. Unrolling (Unfolding): is a loop transformation method that attempts to optimize the program execution speed. The speed optimization is accomplished by eliminating some loop control statements from programs [43].

4. FPGA Resources: which are predefined blocks that are built inside the FPGA. Moreover, FPGAs support the option of using extra or external resources. These resources include Block RAM (BRAM), Digital Signal Processing (DSP) unit, and Shift Register Look-up table (SRL) [44]. The purpose of BRAM is to store constants and move the combinational logic block burden to BRAM burden. The DSP unit is used to perform part of the logical and mathematical calculations. While the SRL is used to reduce time and area requirement.

5. Iterative Method: is a mathematical operation that tries to get the solution according to a series of improved approximations[45].

6. Mixed Optimization: where two or more optimizations are combined to form an extra optimized version.

All SHA hash standards are bounded by the dependencies between internal rounds. Through several types of optimizations, the most significant techniques were those provided a good enhancement in the performance metrics, particularly, Throughput (Tp), Area (A), and Throughput/Area (Tp/A) ratio. In this section, we will discuss previous works and their contributions in this field.

2.4.1. FPGA Implementation of the SHA-1

This section presents the various FPGA design implementations of the SHA-1 hash standard. SHA-1 has a data dependency between internal steps, so that, any improvement to SHA-1 is relevant to this property. SHA-1 compression function has a critical path equation[7], which relies mainly on add operation. Carry Save Adder (CSA) is used in hardware optimization to enhance the performance of add operation. The CSA is small and fast, because it separates the addition and carry-operations, thereby minimizing the delay time caused by the carry. Optimization through the CSA is frequently used in the implementation of SHA-1 and SHA-2 standards, as both of them rely mainly on addition inside critical path [20]. CSA is used along with Carry Look-ahead Adder (CLA) in some optimization to pre-compute the critical path, which in turn reduces the addition operations and CSA units. The combination of CSA and CLA reduces the computation delay, and save more area resources. This scheme was adopted by Makkad *et al.*, where CSA and CLA were used to optimize the implementations of unfolded, pre-computation and 4 stages pipeline [46]. CSA is used to compute the intermediate values between steps, and reduce the computation delay.

[7]The critical path, is the longest equation of the compression function. Return to SHA-1 and SHA-2 definitions.

The proposed designs were tested and verified using Xilinx Virtex-6 (LX240T) FPGA, and Xilinx ISE 13.2 Design suit. The results showed that the 4-stages pipeline design with CSA gave a good result in term of Throughput with 8607.6 $Mbps$. While, pre-computation, unfold, and CSA-basic architectures gave 2596.2 $Mbps$, 1927.5 $Mbps$, and 1347.2 $Mbps$, respectively. Contrarily, the results showed that CSA-basic design gave the best figures in term of area requirements, as depicted in Table 2.5. Another work that conducted a comparison between different optimization methods including CSA was presented in [46]. The authors used CSA, loop unfolding (two operations), pre-computation, and pipelining technique. The proposed work showed that 4-stages pipeline exceeded the others by throughput but preceded them in term of area requirements. While, CSA optimization gave significant figures in term of area, but the increased operational frequency was obvious which in turn increases the power consumption.

The pipelining method gives an enhanced performance in term of throughput, but influences the area requirements. Authors in [47] proposed a targeted design approach, focusing on the increase of operating frequency (f-operation) and throughput. They focused on maintaining the area requirements without introducing a significant area penalty. The proposed methodology employed hardware reuse and pipelining techniques. The hardware reuse got benefited from the predefined units without creating a new one for different operations. Moreover, the proposed design used multiplexing to control the flow of data and manage the units' usage. The proposed design curtailed the critical path of the SHA-1 algorithm into two stages (addition and multiplexing). The authors tested their design on Xilinx Virtex FPGA. The results showed an improvement in term of speed without significant effect on area requirements. The speed of optimization exceeded 2.5 $Gbps$ throughput with an increased percentage by 37% over the basic structure, and 950 slices of the area. Sometimes, the throughput requirements are much more important than area requirements, thus the designers focus only on optimizing the throughput. Pipelining can also be used jointly with other optimization methods to further improve the overall performance requirements of the system. Suhili *et al.* in [48] proposed high speed and throughput design for the SHA-1 using pipelining with unfolding technique. Five evaluation designs were tested using Xilinx Virtex-5 and Altera Arria-2 FPGAs. The designs used iterative, inner-round pipelining, 4-stage pipelining, 40-stage pipelining and 40-stage pipelining with two times unrolling ($40PPL + 2X$-Unroll). Pipelining is used to maintain the data dependency between internal steps of the compression function, while

the unrolling expand the internal loops of function. Then, the combination will give the pipelining more space to manage the data dependencies. However, this kind of combination will dramatically increases the area requirements. The proposed designs gave the best results for 40-stage pipelining combined with two times unrolling design, with a claimed throughput of 150.269 *Gbps* and 223.618 *Gbps* for Altera and Xilinx FPGAs, respectively.

One of the key enhancement to reduce the critical path of the SHA-1 is the unrolling (unfolding) method. Loop Unrolling increases the number of instructions, but increases the parallelism level. In [49], the authors proposed an architecture to achieve higher parallelism and minimize the critical path. The proposed design employed two unfolded architectures, a pre-computation and hash core. The pre-computation architecture of the SHA-1 used newly defined parameters that are pre-computed before other parameters. The hash-core architecture is used to compute the n^{th} hash operation of the internal hash computation steps. Both designs were tested and verified using Xilinx Virtex-2 (xc2v1000) FPGA with 5.9 *Gbps* throughput and 2894 of area slices. On the other hand, the work presented in [50] tested unfolding architecture up to eight stages. Authors applied their work for the SHA-1 and combined their work with the pipelining method. Their work was tested and verified using Xilinx Virtex-2 FPGA. The results showed a significant improvement in using unfolding along with pipelining better than using unfolding alone. The better results were achieved by using unfolding that reduced the number of required clock cycles to 12 for unpipelined version and 24 cycles for pipelined version. The results gave 3.541 *Gbps* throughput and 4258 area slices using 4-stages unfolding with pipelining. While 893 Mbps throughput and 2394 area slices were achieved without pipelining. As shown in Table 2.5, The enhancements are not guaranteed with the unrolling method, unless they are used with a predefined criterion.

Since the SHA-1 and SHA-2 follow the same construction model. Both can be combined in one system as presented by Michail *et al.* [51]. The proposed work was based on a pipelined design for area-throughput trade-offs for SHA-1 and SHA-256. The work compared between the optimized and non-optimized pipelined designs. The optimized version employed a loop unrolling technique, which unrolled two iterations with one mega step using pre-computation and CSA units. Both of the designs were implemented using four FPGAs: Virtex (xcv1000-6FG680), Virtex-E (xcv3200e-8FG1156), Virtex-II(xc2v6000-6FF1517), and Virtex-4 (xc4vlx100). The speed was used as an optimization goal for both base and optimized versions. The authors claimed that the best results

for base architecture, were those produced by Xilinx Virtex-4 FPGA, with 66.36 *Gbps* and 56.93 *Gbps* for SHA-1 (80PPL) and SHA-256 (64PPL), respectively. While the best throughput of the optimized versions were 88.37 *Gbps* with 40-stages PPL, and 69.27 *Gbps* with 32-stages PPL, on the same FPGA.

As well, the need for low power designs is crucial in case of big data processing. Isobe *et al.* in [52], implemented all processes of TLS/SSL into one FPGA. Low power consumption and high speed were gained by dividing processes into three phases. Phase one was a design of the RSA unit using parallel processing. Phase two included shared key cryptography and hash function unit for sending and receiving. While, phase three employed a protocol processing block, cipher processing block, and data exchange block. The proposed work gave enhanced throughput figures without increasing the power consumption factor. Their work maintained the operating frequency of $65nm$ FPGA to the same levels of non-optimized version, as shown in Table 2.5.

Many designs were used to evaluate and validate SHA standards on Computer-Aided Design (CAD) tools [53, 54]. The work performed by Iyer *et al.* [53] introduced three modules (initial, round and top) to model the SHA-1 algorithm using FPGA. The evaluation and synthesis of the SHA-1 algorithm were applied using the Xilinx Software Development Kit (SDK) toward Virtex-5 FPGA. The CAD tool was used to accelerate the calculation of the SHA-1. Results gave a better speed than the regular processor and comparable figures with high-level synthesis (HLS), in term of throughput. Janik *et al.* in [54], used HLS tools for Altera OpenCL (AOCL) to accelerate the computations of SHA-1. The design was tested and verified using Quartux SDK toward Altera Stratix-5 FPGA. The results showed an enhancement in the system speed over CPU and GPU.

As mentioned before, the SHA-1 and SHA-2 follow the same construction model, therefore, both of them can be combined on the same FPGA implementation. Michail *et al.* [55] proposed a high throughput and area-efficient multi-mode secure hash algorithm using the FPGA. Their work supported the SHA-1 and SHA-2 (256, 512) output hashes. The system was able to produce hash according to the user selection of the SHA-1 or SHA-2. The design was tested and verified using different FPGAs (Virtex-4, 5, 6, and 7). The proposed architecture employed pre-computation and pipelining to optimize the resulting design. The experiments were carried out, for the base and optimized structures, and proved the progression of the optimized architecture.

Table 2.5. Comparison of FPGA optimization methods for the SHA-1 hash function

Work	Opt-Tech[c]	FPGA[b]	Tp/Gbps	Area/slice	Freq/Mhz	Mbps/Slice
[46]	CSA	Virtex-6	1.3472	449	213.13	3.0004
	Unfold		1.9275	518	154.35	3.721
	4PPL		8.6076	1230	172.32	6.998
[47]	PPL	Virtex	2.5267	950	98.7	2.659
[51]	Basic with (4,8,10,20,40)[a] PPL	Virtex	(1.18, 2.35, 2.85, 5.73,-)	(2070, 3920, 4950, 9570, -)	(45.9, 45.9, 44.6, 44.8, -)	(0.57, 0.60, 0.57, 0.59, -)
		Virtex-E	(1.46, 2.93, 3.53, 6.76, 13.08)	(2140, 4030, 5060, 9430, 17690)	(57.0, 57.3, 55.2, 52.8, 51.1)	(0.68, 0.73, 0.69, 0.72, 0.74)
		Virtex-2	(1.82, 3.65, 4.12, 9.09, 17.92)	(2000, 3820, 4800, 9860, 18650)	(71.1, 71.2, 64.3, 71.0, 70.0)	(0.91, 0.95, 0.86, 0.92, 0.96)
		Virtex-4	(3.09, 6.19, 7.01, 14.18, 28.24)	(1990, 3930, 4750, 9000, 17560)	(120.7, 120.9, 109.6, 110.8, 110.3)	(1.55, 1.57, 1.48, 1.57, 1.61)
	Opt (4,8,10,20,40) PPL	Virtex	(3.05, 6.09, 7.46, 14.47,-)	(2490, 4750, 5940, 11060,-)	(59.3, 59.5, 58.3, 56.9, -)	(1.19, 1.28, 1.26, 1.31, -)
		Virtex-E	(3.79, 7.59, 9.23, 17.95, 38.35)	(2560, 4840, 6070, 11310, 20170)	(74.1, 74.1, 72.1, 70.1, 74.9)	(1.48, 1.57, 1.52, 1.59, 1.90)
		Virtex-2	(2.0, 3.97, 7.92, 15.87, 39.07)	(2410, 4590, 5770, 11860, 22000)	(92.4, 92.4, 83.5, 92.3, 95.5)	(0.83, 0.86, 1.37, 1.33, 1.77)
		Virtex-4	(8.05, 16.10, 18.21, 36.89, 88.37)	(2390, 4560, 5740, 10830, 20190)	(157.2, 157.2, 142.3, 144.1, 172.6)	(3.37, 3.53, 3.17, 3.41, 4.37)
[49]	Unfold	Virtex-2	5.9	2894	118	2.038
[50]	4x-Unfold	Virtex 2	0.983	2394	20.9	0.41
	4x-Unfold+4PPL		3.541	4258	41.5	0.83
[48]	(1, 4, 40)-PPL	Arria II	(1.746, 7.911, 80.306)	(402, 976, 6715)	(279.64, 316.76, 321.54)	(4.343, 8.11, 11.96)
		Virtex-5	(1.46, 10.740, 10.67)	(897, 1332, 6465)	(233.899, 430.024, 427.54)	(1.628, 8.06, 16.51)
	2x-Unfold+40PPL	Arria II	150.269	9799	308.17	15.335
		Virtex-5	223.618	11994	458.593	18.644
[52]	Parallel process	65nm FPGA	14.9	3986	236	3.74
[53]	Three module + SDK	Virtex-5	0.786	1351	124.502	0.58
[54]	HLS for AOCL	Stratix-5	3.033	-	-	-
[46]	Pre-Computation	Virtex-6	2.596	546	207.90	4.75
[55]	Basic	Virtex(4,5,6,7)	(4.0, 5.0, 5.1, 5.9)	(7737, 3899, 3787, 3743)	(77.9, 98.3, 99.8, 115.6)	(0.52, 1.28, 1.35, 1.58)
	Opt		(9.2, 11.9, 12.8, 14.3)	(9623, 4275, 4129, 4133)	(90, 116.3, 124.7, 139.6)	(0.96, 2.78, 3.10, 3.46)

[a] Each value in parenthesis refers to its Pipelining level. The (Tp, Area, Freq and Mbps/Slice) for each level represented in parenthesis in the same order at the same raw.

[b] Virtex refers to Xilinx, while Arria refers to Altera.

[c] Basic: Implementation without optimization. CSA: Carry Safe Adder. xPPL: Pipelining to x level. Nx-Unfold: Unfolding method with N unrolled loops.

- Refers to a non reported results.

2.4.2. FPGA Implementations of the SHA-2

The data dependency, between internal steps of SHA-1 and SHA-2, plays a key role in performance improvements. The SHA-2 follows the same construction model as the SHA-1, so any improvement to the SHA-1 is naturally applicable to the SHA-2 [55]. The CSA is used in the SHA-1 to speed up the calculation of the critical path. Similarly, the CSA was adopted by different works to optimize the FPGA implementation of the SHA-2 [56, 57, 58]. Sun *et al.* proposed an architecture to optimize critical path calculation using the CSA [56]. The proposed design supports different SHA-2 flavors (256, 384, and 512) using the control-selection unit. The architecture was tested using $ModelSim6.0a$, and targeted to Xilinx Virtex-2. The results showed improvements in term of throughput and area comparing with the SHA-1, as shown in Table 2.6.

To overcome the overhead of add operations, Mohamed *et al.* in [57] used Carry Propagation Adder (CPA) with CSA. The longest critical data paths contain six and seven add operations, whereby the CPA is used to overcome the overhead of the modular addition computations. The optimization was performed on two levels, the Look-Up Table (LUT) level, and the CSA level. The methodology was tested using Xilinx Virtex-5 FPGA (xc5vlx330t-2ff1738). The results showed that, in term of delay, the optimization on LUT level gave better results than the CSA level. The same throughput of 1359.6 $Mbps$ was observed for both designs. While Algredo *et al.* in [58] proposed two hardware architectures to compute the inner loop of the SHA-2 using the CSA. The first architecture was based on re-arranging the data-flow of the critical path equations. The second architecture was based on two pre-computations instead of one. The proposed approaches relied mainly on the use of the CSA that was used for balancing data paths, and a state buffer. The systems were tested and verified using Xilinx Virtex-2 (XC2VP-7) FPGA. Both designs gave the same figures in all terms, as shown in Table 2.6.

To take advantage of the pipeline method, which is used to optimize the speed and throughput, authors in [37] proposed a pipelined architecture to optimize the SHA-2 in term of throughput and area. To achieve the goal, a set of optimization techniques were applied, systematically, according to two levels. The first one is the algorithmic level, that includes loop unrolling and pre-computation. The second level involves circuit level technique, that includes resource re-arrangement and use of special circuit resources such as CSA. Both levels were combined with

thoroughly applied stage pipelining. The designs were tested and verified using Xilinx Virtex-(2, E, 6, and 7) FPGAs. The proposed techniques showed the best results with the 5-stage pipeline in term of throughput and throughput/area.

Loop Unrolling (Unfolding) method is also adopted to optimize the SHA-2 hash standard. Multiple rounds of the compression function are unrolled and processed in combinational logic components, which reduces the clock cycles required to compute hash function [36]. However, Loop Unrolling method gives no significant results in some cases, unless it is used with other optimization methods. For instance, unroll internal loops of compression function for multiple times will cause an extra area penalty. So the better use of loop unrolling is to combine it with other methods. Authors in [59] proposed a design that combines unrolling optimization with pipelining. a new VLSI architecture was presented, that combines a fast quasi-pipelined design with unrolling. The authors tested their architecture to find the best pipeline-unroll combination. The design was tested and verified using FPGA Xilinx Virtex-2 FPGA. The results showed that SHA-256 pipelined without unrolling gave the best throughput (1.009 Gbps). While the best result for SHA-512 was the unrolled two times with pipelining (1.466 Gbps).

According to the similarity of the internal construction of the SHA-1 and SHA-2, any optimization or implementation toward SHA-1 will work with the SHA-2 [55]. Some designs employ a hardware unit to optimize and validate SHA-2 hash function [60, 61]. In [60], the authors proposed an efficient hardware implementation for the SHA-256 and SHA-512 using Xilinx Virtex-5. The proposed work employs a control unit to manage the flow of data from the padding unit, and passes the padded data to the hash computation unit. The process speeds up the hash production and yields better efficiency. On the other hand, a compact FPGA implementation for the SHA-2 hash function was proposed in [61]. The work designed a customized processor based on FPGA. The design relied on the data reuse to minimize memory access with the help of cache memory. The results showed an improvement in the critical path calculations and good figures in term of throughput.

To support all flavors of the SHA-2 standards along with optimizing them, Rote *et al.* in [62] proposed a performance-enhanced architecture for the SHA-2. The authors introduced a pipelined-round and fully iterative technique for the SHA-2 standard (224, 256, 384 and 512). The design was implemented using Xilinx Virtex-6 FPGA. Results showed that the round pipeline

architecture gives better results in term of throughput and comparable figures in term of area, concerning iterative architecture.

Power consumption is also considered when deploying the SHA-2, Thakur *et al.* in [63]. proposed a low Power and simple implementation of the SHA-2. Their work employed a control system to pass the message blocks without waiting for the synchronization signal. The proposed work decreases the operating frequency of the implementation, hence lowering the power consumption factor.

SHA-2 can be enhanced when careful analyses are made. All aforesaid works showed that SHA-2's figures are versatile depending on the kind of optimization. For instance, throughput gives good results in some cases, and moderate in others.

2.4.3. FPGA Implementations of the SHA-3

Keccak or SHA-3 (officially) is the latest hash function announced by the NIST. Keccak competition has taken three rounds of selection to choose the winning competitor. Particularly, the final round that comprises five algorithms. The selection process relied mainly on the hardware deployment of the competitors. Because of that, we observed many FPGA implementations toward the SHA-3. For instance, a comparison between the final five candidates was investigated by Gaj *et al.*[64]. The work compared design implementations of each candidate. The deployments were tested on both Altera and Xilinx FPGAs. The results showed that Keccak outperformed the other algorithms in both area and speed requirements. Moreover, an experimental bench-marking for different SHA-3 candidates was proposed in [65], where a new Xilinx Zynq board was used with high performance and flexibility. The new board gave a more accurate comparison between the candidates. They are verified using Vivado studio development tool. The results showed that Keccak performed the best among the algorithms in terms of throughput and frequency.

The authors of Keccak claimed that it fully supports hardware implementations [66]. Among all implementations and optimizations of SHA-3[8], we selected the most interesting works in the SHA-3 (Keccak) area that provided the best performance of Keccak FPGA deployment. To follow the same event-listing fashion, we present the optimization methods in the same order they appeared in the previous two sections.

[8]When SHA-3 is mentioned, it explicitly means Keccak.

Keccak follows sponge structure model, which is different from the SHA-1 and SHA-2, as discussed in Section 2.2. However, the CSA can also be used to optimize Keccak standard, jointly with other methods like unfolding, loop unrolling, and pipelining. Authors in [67] proposed a compact FPGA implementation for Keccak function. The design focused on area optimization by merging *rho*, *pi*, and *chi* steps. The area reduction is accomplished by using the same processing units for the three steps, accordingly reduces the number of units used for the FPGA deployments, and consequently reduces the area requirements. The CSA was employed between the three merged steps to speed up the hash computation process. The proposed design was tested and verified using Xilinx Virtex-5 (xc5vlx330t) FPGA. Results showed that the area requirements have decreased to 240 slices with a significant throughput of 7.224 *Gbps*. In essence, the CSA reduces the cycles used to compute the final hash value, besides increasing the throughput of the overall design.

To achieve higher throughput, the pipeline method is used in Keccak FPGA designs. To give a good implication of this property, authors in [68] proposed a pipelined architecture to implement Keccak hash function. The proposed design consists of 4 units (control unit, input/output buffer, padder unit, and Keccak round). The control unit is used for the synchronization of data flow between units. The input/output buffer is used to communicate with external modules. Padder unit is used for padding operation of the input. While the Keccak rounds involve the main data path for the Keccak computations and include the round's constants, the main optimization of the proposed design was on the Keccak rounds by putting the pre-calculated round's constants in registers. Later, all constants are fed to *iota* step one per a round. The system was tested and verified using Xilinx Virtex-5 (XC5VFX70T) FPGA with a throughput of 12.68 Gbps. However, the area requirement was increased Successively, as shown in Table 2.7.

To maintain the area requirements of the pipelined-optimized architecture. Authors in [39]. proposed an area-efficient design for the final five candidates of the SHA-3 competitors (BLAKE, Grøstl, JH, Keccak, and Skein). The candidates were implemented using the pipelining method. All rounds of the proposed designs were kept unchanged, but a 1-stage pipeline is applied. The proposed architectures were tested and verified using Xilinx Virtex-5 and Virtex-6 FPGAs. The results showed that Keccak gave the best figures in term of throughput/area. However, the throughput of the proposed design is significantly low with 864 *Mbps* for Virtex-5 and 145 *Mbps* for Virtex-6. Despite the area requirements were low for both FPGAs. However, to maintain the area and

throughput requirements of FPGA designs. Michail *et al.* in [69] proposed a pipelined architecture for SHA-3 Keccak hash function. Several pipeline stages were tested to determine the most appropriate number of pipeline stages. The design was tested and verified using Xilinx Virtex-4, Virtex-5, and Virtex-6 FPGAs. Experiments showed that the most appropriate number of pipeline stages were four, which produced the best results in term of throughput, area, and throughput/area requirements. The best results of 37.63 *Gbps*, 4117 *slices*, and 9.14 Tp/A were produced by the Virtex-6 FPGA.

Two stages of pipelined design for the SHA-3 hash function was proposed in [70]. The proposed work relies on four components: transformation round, registers, Version Selection initial XORing (VSX) module, zero state register, and control unit. Transformation round combines Keccak round functions (*theta*, *rho*, *pi*, *chi* and *iota*) which in turn cuts the critical path to half. The VSX module responsible for selecting the appropriate Keccak state ([1152, 448],[1088, 832],[832, 768], [576,1024] and [1024,576])[9]. While zero state register is used to initialize the state with 0's. The control unit uses a finite state machine (FSM) to employ five states, S1 for version selection, (S2, S3) for computations, and (S4, S5) for final hash computation. The designs were tested and verified in term of area, frequency and throughput using Xilinx Virtex-5, Virtex-6, and Virtex-7, FPGAs. All results are converged around the same value in all terms, as shown in Table 2.7.

To build a system that supports multi-messages block, the authors in [71] proposed a high-performance FPGA implementation of the SHA-3 hash function. The proposed design used a pipelined multi blocks message architecture that supports all SHA-3 flavors (224, 256, 384 and 512). The architecture employs a software scheduler that performs three functions. The first one process the input message (pad and divide into blocks of 1600 bits). The second function truncates the 512-bits output to the desired hash (224, 256, 284, and 512). The third function updates the state matrix in case of the multi-block message. The design was tested and verified on Xilinx Virtex-4 (XC4VLX200), Virtex-5 (XC5VLX330T), and Virtex-6 (XC6VLX760) FPGAs. The results showed an optimized throughput of two stages pipeline with the increased area used for the three selected FPGAs, as shown in Table 2.7.

The influence of the pipeline method on the unrolled architecture was also studied for Keccak. Suigar *et al.* in [72], presented a low cost and high speed implementation of the SHA-3 hash

[9]Return to Keccak definition in section 3 for details.

Table 2.6. Comparison of FPGA optimization methods for the SHA-2 hash function

Work	Hash	Opt-Tech	FPGA	Tp/Gbps	Area(A)/slice	Freq/Mhz	Tp/A
[60]	256	Conrol-Unit	Virtex-5	1.58 Gbps	387	202.54	4.08
[56]	256	CSA	Virtex	0.291	2207	74	0.132
	384			0.250			0.113
	512			0.467			0.212
[51]	256	Basic with (2,4,8,16,32, 64) PPL	Virtex	(0.75, 1.50, 2.91, 5.70, -, -)	(1330, 2390, 4580, 8990, -, -)	(46.2, 46.9, 45.4, 44.5, -, -)	(0.56, 0.63, 0.64, 0.63, -, -)
			Virtex-E	(0.83, 1.66, 3.25, 6.35, 12.83, 29.85)	(1360, 2400, 4790, 9090, 17810, 29990)	(51.8, 51.8, 50.8, 49.6, 50.1, 58.3)	(0.61, 0.69, 0.68, 0.70, 0.72, 0.99)
			Virtex-2	(1.00, 2.00, 3.97, 7.92, 15.87, 39.07)	(1440, 2600, 5290, 10660, 18980, 27610)	(62.4, 62.4, 62.1, 61.9, 62.0, 76.3)	(0.69, 0.77, 0.75, 0.74, 0.84, 1.42)
			Virtex-4	(1.74, 3.48, 6.93, 13.82, 27.65, 56.93)	(1580, 2970, 6290, 12120, 23520, 31420)	(108.7, 108.8, 108.3, 108.0, 108.0, 111.2)	(1.10, 1.17, 1.10, 1.14, 1.18, 1.81)
		Opt with (2,4,8,16,32) PPL	Virtex	(1.83, 3.67, 6.95, 13.67, -)	(1720, 3100, 5770, 11900, -)	(57.3, 57.3, 57.3, 54.3, 53.4)	(1.06, 1.16, 1.20, 1.15, -)
			Virtex-E	(1.98, 3.97, 7.81, 14.98, 32.36)	(1770, 3160, 5890, 12160, 19410)	(62.0, 62.0, 61.0, 58.5, 63.2)	(1.12, 1.25, 1.33, 1.23, 1.67)
			Virtex-2	(2.38, 4.77, 9.51, 18.97, 40.50)	(1940, 3500, 6620, 13780, 20960)	(74.5, 74.5, 74.3, 74.1, 79.1)	(1.23, 1.36, 1.44, 1.38, 1.93)
			Virtex-4	(4.16, 8.33, 16.64, 33.20, 69.27)	(2060, 3960, 7780, 15260, 26340)	(130.1, 130.1, 130.0, 129.7, 135.3)	(2.02, 2.10, 2.14, 2.18, 2.63)
[57]	256	CSA	Virtex-5	1.3596	1203	170	1.13
[58]	256	CSA	Virtex-2	0.86772	1187	110.10	0.731
		CSA+ Pre-Comp		0.909	1274	115.46	0.713
[37]	512	5PPL	Virtex-2	6.989	7012	54.6	2.659
			Virtex-E	9.126	7151	63.4	1.276
			Virtex-6	9.126	7151	71.3	1.276
			Virtex-7	11.674	7219	91.2	1.617
[73]	256	DSP+BRAM	Stratix-3	1.621	795	205.8	2.03
[59]	256	Basic+PPL	Virtex-2	1.009	1373	133.06	0.735
		2x-unfold+PPL		0.997	2032	73.975	0.491
		4x-unfold+PPL		0.909	2898	40.833	0.314
	512	Basic+PPL		1.329	2726	109.03	0.488
		2x-unfold+PPL		1.466	4107	65.89	0.357
		4x-unfold+PPL		1.364	5807	35.97	0.235
[55]	256 and 512	Basic	Virtex(4,5,6,7)	(4.0, 5.0, 5.1, 5.9)	(7737, 3899, 3787, 3743)	(77.9, 98.3, 99.8, 115.6)	(0.52, 1.28, 1.35, 1.58)
		Opt		(9.2, 11.9, 12.8, 14.3)	(9623, 4275, 4129, 4133)	(90, 116.3, 124.7, 139.6)	(0.96, 2.78, 3.10, 3.46)
[61]	256	Data Reuse	Virtex	0.649	431	35.5	1.5
			Virtex-4	0.915	422	50.06	2.17
			Virtex-5	1.18	139	64.45	8.49
[62]	All Flavors	Iterative	Virtex-6	1.063	766	132.91	1.38
		PPL		2.047	871	271.96	2.35
		Iterative		1.081	736	135.14	1.47
		PPL		2.040	905	271	2.25
		Iterative		1.496	1651	116.9	0.91
		PPL		2.445	1724	200.64	1.42
		Iterative		1.428	1613	111.56	0.89
		PPL		2.348	1811	192.57	1.31

* Please refer to the definitions in the bottom of Table 2.5.

function. Comprehensive implementations of Keccak were employed using loop unrolling with and without pipelining. Seven architectures were tested and verified using Xilinx Spartan-3E FPGA. The architectures were, the basic Keccak, $(1x, 2x, 3x)$-unrolled architecture, and $(1x, 2x, 3x)$ unrolled architecture with (2,3,4)-stage pipelining. The unrolled architectures with pipelining showed the best results in term of throughput and area. The drawn implication after the incorporated experiments showed the defectiveness of unrolling in optimizing the design of Keccak.

FPGA resources are used to make a fair comparison between the SHA algorithms. The FPGA resources include BlockRAM, DSP and, SRL. The authors in [73] proposed an optimization technique for the FPGA implementation of 14 selected hash functions from round two of the selection competition of the SHA-3. The optimization approach was concentrated on the use of the FPGA resources DSP and Block memory (BRAM) units. The proposed system was tested and evaluated for 256-bits output hash. The implementations were tested on Altera Stratix-3 FPGA. The authors divided the functions of the algorithms into categories according to the potential use of the FPGA resources. The testing results for the throughput showed an improvement concerning the throughput /combinational logic block (#CLB) ratio. On the other hand, some works relied on the use of SRL. In [74], the authors designed an FPGA implementation of Keccak using SRL. The SRL register was used in the *rho* step. The results showed a reduced usage of the FPGA resources due to the reduction in LUTs usage. The proposed work outperformed other designs in term of area requirement. The design was tested and verified on Xilinx Virtex-5 FPGA with 156 *Mbps* throughput. Other architectures were benefited from the use of the FPGA resources that are designated for the individual logic operations. Provelengios *et al.* in [75] proposed an optimization technique using the DSP unit ($DSP48E$). The proposed work employed the DSP unit to compute the logic functions (AND, NOT and XOR) that appear in *theta* and *chi* steps. The design was tested and verified using Xilinx Virtex-5 FPGA. The result showed that using DSP unit was inefficient in low complexity demand applications.

However, to test the efficiency of the DSP units with other optimization methods, Ayuzawa *et al.* in [76] proposed an FPGA implementation for Keccak hash function using the DSP unit and pipelining. The authors employed an advanced DSP unit to optimize the overall throughput to area ratio. In their work, pipeline registers were inserted between steps of Keccak operations. All registers were used to support multi-message hashing. The proposed work was tested and verified

using Xilinx Virtex-5 FPGA. Results showed a significant enhancement of 50% in term of throughput to area ratio. In some cases, FPGA resources are used to reduce the power consumption of the FPGA designs. Aziz *et al.* [77] proposed a new approach in designing a low power consumption for Keccak FPGA design. The work was benefited from the full capability of $DSP48$ unit that is widely available in Xilinx FPGAs. The authors presented two designs: one for area distribution, and the other for speed reservation. The DSP48 unit was employed for the calculation of the Keccak hash function. The proposed design was tested using Xilinx Virtex-6 FPGA. The low-frequency figures of 451 MHz were observed, which lower the amount of consumed power to 130 mW.

The flexibility of the SHA-3 Keccak function influenced the researchers to widen their ideas and exploit the FPGA resources. The hardware convenience made the enhancement techniques of Keccak outperform SHA-1 and SHA-2 [78, 79, 80, 81, 82]. Chandran *et al.* in [78] proposed a design using special logic gates. The work supported Keccak calculation in two ways. The first one was step by step algorithm that got benefited from the fact that Keccak runs the five steps 24 times. The second way was to design the algorithm using a multiplexer (MUX) that connects two blocks of Keccak function. The proposed designs were tested and verified using Xilinx Spartan-6 FPGA. The results showed reduced area requirements for the final implementations. Moreover, Rao *et al.* in [79], proposed an architecture to enhance the implementation of the SHA-3 Keccak on FPGA. The work was performed through two phases. In the first phase, the algorithm steps were logically combined to get a total of 25 equations. The second phase includes the hardware design of the algorithm. Manual fulfilling was applied to phase one, then forming the 1600-bit state. The resultant equations were fed to the proposed hardware architecture in phase two, that processed on LUT level. The architecture was tested and verified using Xilinx Virtex-5, Virtex-6 FPGAs. The authors got a throughput of 17.132 *Gbps* for Virtex-5 FPGA and 19.241 *Gbps* for Virtex-6 FPGA. The area requirement optimizations were also studied by kahari *et al.* [83]. A high-speed implementation of SHA-3 Keccak hash function in terms of area and frequency was proposed. The proposed work divided the Keccak hash calculation into two parts namely, the sponge and round functions. The sponge part performs message initialization along with state matrix padding. While the squeeze phase performs the compression calculation and hash production. The proposed design was tested and verified using Xilinx Virtex-5, Virtex-6, and Virtex-7 FPGAs. Virtex-6 FPGA gave

the best results in term of area and efficiency, while Virtex-7 was the best in term of throughput, as shown in Table 2.7.

As Keccak hash function supports variable length hashes (SHAke128, SHAke256). Previous works also investigated the deployments of Keccak toward variable-length output [81, 82]. A compact FPGA implementation is presented in [81]. The authors designed a 1024 hash output of SHA-3 using Xilinx Virtex FPGA. While Sravani *et al.* in [82] supported variable length SHA-3 by combining Rho , Pi, and Chi steps into one step, as discussed before.

In general, optimization methods, either, are dedicated to a specific secure hash algorithm (SHA) or focus on optimizing the architecture toward specific metric(s) (area, speed, throughput, power consumption). The resulting designs help to test any system in a predefined area of interest. Tables 2.5, 2.6 and 2.7 show the three SHA standards along with their FPGA devices that were used. The optimization methods that were used are listed in the tables. Moreover, the results of the area, throughput, frequency, and throughput/area ratio are depicted in the same tables. As a matter of observation, some designers use BlockRAM metric to calculate area requirements. Therefore, we used the method listed in [84] that stated "each BlockRAM equal to 128 slices". Then the overall area will be calculated according to Eq. (2.30).

$$Area = slices + (128 \times BlockRAMs) \tag{2.30}$$

Another important area we need to give insight to is error detection and correction schemes. Any design that relies on hardware may be affected by errors that arise during the computations. The error detection (fault detection) schemes for SHA hardware implementations were studied in different literature [85, 40, 86, 87]. A brief description of this area is discussed in the next text.

2.4.4. Error Detection and Correction

FPGA is used to implement Hash algorithms, but the complexity of the SHA implementations increases the probability to cause faults in the hardware design. Any single error in any round causes an error for the final generated hash, because each round depends on the previous round (inheritance property)[40]. Therefore, any hardware design and implementation for hash functions need to be reliable and authentic. In general, there are three methods to provide a digital design with a fault detection paradigm:

Table 2.7. Comparison of FPGA optimization methods for the SHA-3 (Keccak) hash function

Work	**Opt-Tech**	**FPGA**	**Tp/Gbps**	**Area/slice**	**Freq/Mhz**	**Mbps / Slice**
[68]	PPL	Xilinx -Virtex-5XC5VFX70T	12.68	4793	317.11	2.71
[39]	PPL	Virtex-5	0.864	393	159	2.19
		Virtex-6	0.145	188	285	0.77
[70]	PPL	Virtex-5	18.7	1702	389	10.98
		Virtex-6	19.1	1649	397	11.6
		Virtex-7	20.8	1618	434	12.9
[71]	2PPL	Virtex-4	12.912	5,494	269	2.350
		Virtex-5	16.896	2,652	352	6.371
		Virtex-6	18.768	2,296	391	8.174
[73]	DSP+BRAM	Stratix III	13.913	4277	306.9	3.25
[76]	DSP+PPL	Spartan-6	9.00	4865	-	1.85
[74]	SRL	Virtex-5	0.156	134	248	1.16
[72]	Basic	Spartan-3	4.65	4443	102.6	1.05
	Unfold (x2, x3, x4)		(4.13, 3.42, 2.86)	(6988, 8665, 11173)	(45.6, 25.2, 15.8)	(0.59, 0.39, 0.26)
	(x2-PPL2, x3-PPL3, x4-PPL4)		(8.23, 9.09, 10.11)	(6409, 8463, 10226)	(90.8, 66.8, 55.8)	(1.28, 1.07, 0.99)
[69]	(1, 2, 3, 4)PPL	Virtex 4	(6.55, 12.91, 20.95, 27.07)	(2365, 5494, 8647, 12870)	(273, 269, 291, 282)	(2.77, 2.35, 2.42, 2.10)
		Virtex 5	(9.17, 16.90, 25.34, 34.27)	(1581, 2652, 3197, 4632)	(382, 352, 352, 357)	(5.80, 6.37, 7.93, 7.40)
		Virtex 6	(9.89, 18.77, 28.15, 37.63)	(1115, 2296, 3965, 4117)	(412, 391, 391, 392)	(8.87, 8.17, 7.10, 9.14)
[75]	DSP	Virtex-5	5.70	2573	285	2.215
[67]	Merg (rho, pi , chi)	Virtex-5	7.224	240	301.02	30.1
[77]	DSP	Virtex-6	4.091	208	451.26	19.66
[79]	Merg 5-states	Virtex-5	17.132	1291	377.86	13.27
[64]	Unfold,+PPL and circuit duplication (x1, x1-PPL2, x2-PPL2, x2PPL4)	Virtex-5	(7.18, 7.38, 7.13, 13.55)	(1283, 1774, 1996, 3428)	(-, -, -, -)	(5.60, 4.16, 3.57, 3.95)
		Virtex-6	(7.47, 8.11, -, 13.64)	(1052,1263, -, 2550)	(-, -, -, -)	(7.10, 6.42, -, 5.35)
		Stratix-3	(8.03, 8.55, 13.09, 17.06)	(3734, 4484, 6617, 8934)	(-, -, -, -)	(2.15, 1.91, 1.98, 1.91)
		Stratix-4	(7.61, 8.96, 12.49, 17.33)	(3723, 4481, 6580, 8934)	(-, -, -, -)	(2.04, 2.00, 1.90, 1.94)
[83]	2-Parts	Virtex-5	11.50	1388	278.39	8.48
		Virtex-6	15.76	1167	394.01	13.83
		Virtex-7	16.58	1418	414.54	11.97
[82]	Iterative and merge (ρ, π, χ)	Virtex-5	7.22	240	301.02	30.1

- Hardware Redundancy. Different hardware are used to perform design implementations, separately. Afterward, the final outputs are compared to check and detect errors. However, the process consumes more resources (duplication of all hardware) and consequently an expensive scheme.

- Time Redundancy. A single hardware is used to perform the design at different times. Time redundancy method is less expensive because it uses the same hardware but the task of checking final architecture is expensive in terms of time consumption.

- Information Redundancy. Uses additional information to check the integrity of data (i.e parity bit). For instance, information can be appended to the end of data processed or injected into the system in a predefined time.

The faults that appear during computations are divided into two categories [40]. First one is the permanent category, where the faults appear all the time during the calculation. The second category is the transient faults, that appear in some time slots and disappears for others. In general, faults affect the overall performance of the system.

The ability of the system to detect and recover from any error was taken into consideration by the authors in [40, 86, 85, 88, 87]. Bahram *et al.* in [40]. proposed a time redundancy technique to detect the faults in the design of the SHA-512 algorithm. The final design is free from both permanent and transient faults. The authors in [86] introduced a Totally Self Checking (TSC) design for SHA-256 hash function. The work concentrates on the faults that appear in a harmful environment, such as high computation situations or physical situation as temperature increase. The proposed work relied on Concurrent Error Detection (CED) and uses the TSC design. The results showed that the design can fully recover to 100% error-free system. The Proposed TSC SHA-256 scheme was tested and verified on Xilinx Virtex-5 (XC5VLX330) FPGA, with a throughput of 3.88 Gbps.

The SHA-1 was also studied for faults detection. In [85], a totally self-checking (TSC) architecture for fault detection of SHA-1 and SHA-256 hash functions was proposed. The authors compared TSC design with hardware duplication method. The design can detect and recover with 100% for odd erroneous and appropriately spread even erroneous. Results showed that the TSC

architecture was better than the Duplicate With Checking (DWC) design in terms of area and efficiency.

Informational redundancy is used in reference [88]. The authors used the technique of errors injecting to the system and checked the ability to detect errors and recover. The scheme injected errors in different stages of the SHA-2 hash and investigated the error at the output stage. They were able to detect and recover from the injected error inside the hash computation process. Time and hardware redundancy is also investigated in [87]. The authors introduced two designs for efficient fault detection of the SHA-2. The first design relies on time redundancy block to detect any transient error in the internal calculation of the round operation. The second design used the hardware redundancy scheme with 100% of hardware overhead. The comparison between both designs was obvious, inasmuch, the first one is a time-consuming approach and the second one is expensive.

Regarding the SHA-3 (Keccak), fault detection schemes were also discussed in different publications [89, 90]. Chandran *et al.* in [89] proposed a performance analysis of modified SHA-3. The work employed an error-tolerant unit for SHA-3 calculation to provide a reliable SHA-3 architecture. Moreover, a multiplexer was used in the round function to calculate the output digest of the full Keccak standard. The results showed a reduced area and high throughput architecture. However, the design has a disadvantage of the time delay. The downside came from the operation of error tolerance that was performed in the same unit. The system was tested and verified using Xilinx Spartan FPGA. Error detection and correction scheme was explored in [90] by Mestiri *et al.* The authors focused on predefined method and applied it to the hardware implementation of secure hash algorithms. The authors proposed a fault detection scheme for SHA-3 Keccak using scramble technique. The authors compared between the protected and unprotected version of Keccak. The results were approximately equal in case of frequency and throughput but the area was increased with 63% for the protected Keccak concerning unprotected version.

In essence, to make the FPGA design of the SHA complete, an error detection and correction mechanisms need to be supported because any single bit error will be reflected to the whole output. There are no limits to the ways that might be risen for optimizing any hardware implementation. FPGA is reconfigurable hardware that gives us total control of all resources [21]. The usage of Application Programming Interface (API) tools like FPGA is useful in the process of testing and

evaluating the secure hash algorithms and ease comparison between them [13]. In the next section, a discussion of different FPGA designs of the SHA will be carried out.

2.5. Discussion

In this work, we have presented the hardware implementation and optimization of the popular cryptography hash algorithms SHA-1, SHA-2, and SHA-3. The majority of the FPGA implementations are established on both Xilinx and Altera manufacturers. In this study, Xilinx FPGAs are represented by Virtex, Virtex-2, Virtex-4, Virtex-5, Virtex-6, Virtex-7, Virtex-E, Spartan-3, and Spartan-6 FPGAs. While Altera FPGAs are represented by Altera Arria-2, Stratix-3, Stratix-4, and Stratix-5 FPGAs.

Fig.2.5 shows the count of the usage of different FPGAs to implement the SHA standards. The figure comprises all aforesaid Tables 2.5, 2.6, and 2.7 that were discussed in Section 2.4. The figure shows that some FPGAs were not used for some hash standard, i.e., Xilinx Virtex-2 FPGA was not used for any of SHA-3 designs. However, Xilinx Spartan-3 FPGA was only used for the SHA-3 implementation. Moreover, some FPGAs were suitable for the three hash standards like Xilinx Virtex-4, Virtex-5, and Virtex-6. As depicted in Fig. 2.6.

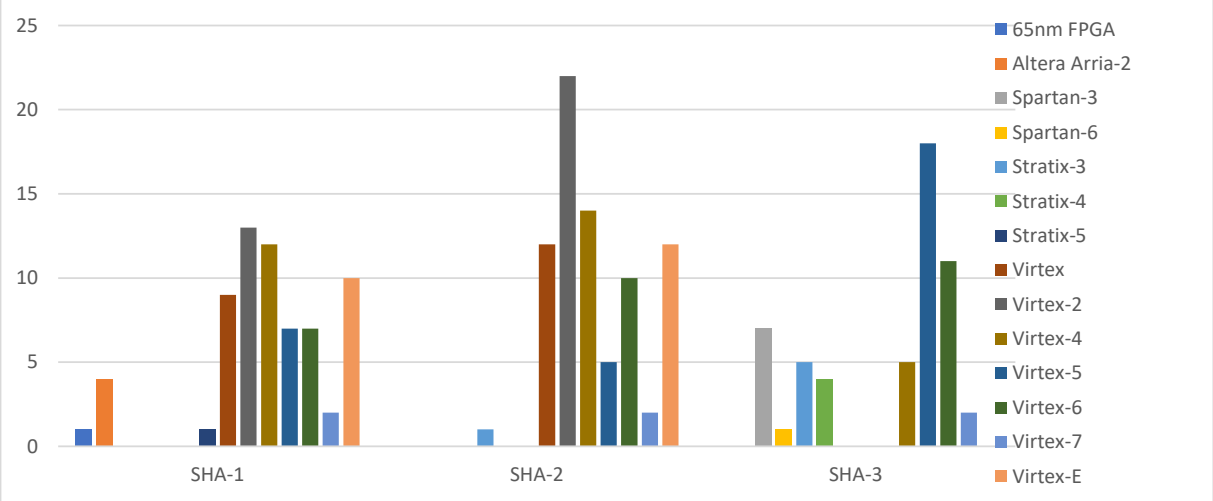

Figure 2.5. Count of different FPGAs' optimizations that were used to implement SHA standards. The figure comprises tables 2.5, 2.6, and 2.7.

Figures 2.7, 2.8, and 2.9 show different graphs for throughput and area with respect to the main optimization methods listed in Tables 2.5, 2.6, and 2.7. The figures depict the influence of

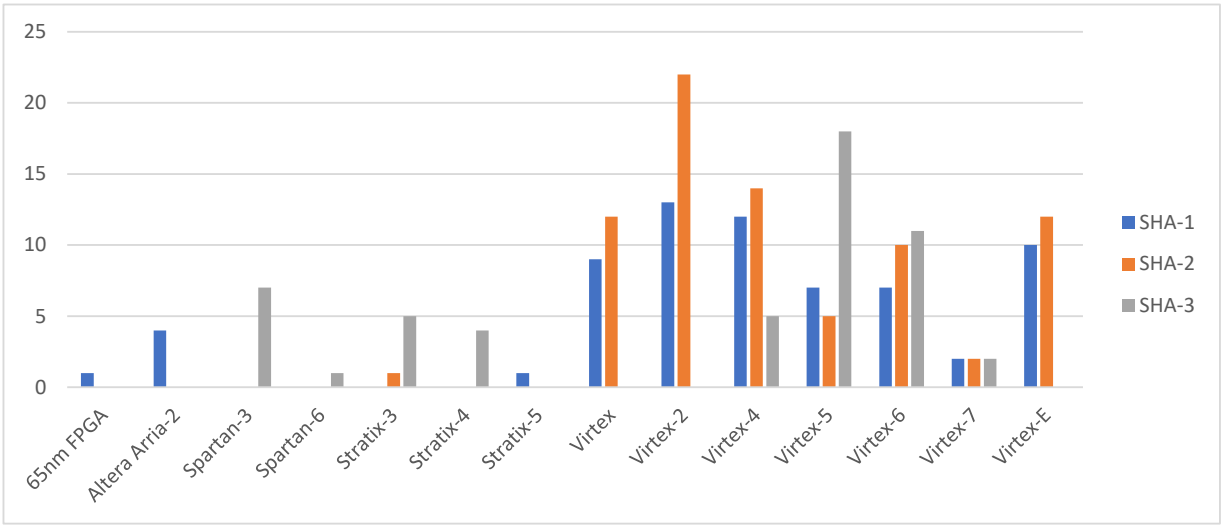

Figure 2.6. Count of optimization regarding each type of FPGAs

optimization techniques on throughput and area requirements for the three hash standards. Each figure contains two graphs, the top graph represents the throughput and area comparison from the FPGAs perspective, while the bottom graph represents the throughput and area comparison from the optimization methods perspective. The x-axis of all figures represents the Area per slices, while the y-axis the throughput per Gbps. To help read the figures, pick any point from the top graph and join it with the same point-position at the bottom graph. Then you will get the Throughput and Area requirements of the point which is implemented using the selected FPGA from the top graph and optimized with the corresponding optimization method from the bottom graph.

Fig.2.7 represents the throughput and area relationship regarding different FPGAs' implementations for the SHA-1 hash standard. Even though the authors in [51] claimed that FPGAs have an upper limit for each optimization methods in which any pipeline optimization higher than eight will give unrealistic results. In spite of that, the designs of the SHA-1 on Xilinx Virtex-5 and Altera Arria-2 proved that pipelining higher than 8-stages can be used for optimization with a high throughput performance. Virtex-5 gives the best throughput among the other when used with two times unfold and 40-stage pipeline (2x-unfold+40PPL) optimization method. Altera Arria-2 comes after for the same design [48]. In the same figure, we can see how the CSA affects area requirements. Moreover, area slices will increase by the use of the CSA along with the pipelining

(PPL) method. Loop-Unrolling shows a better throughput performance when used with pipelining, but the augmented area slices are significant.

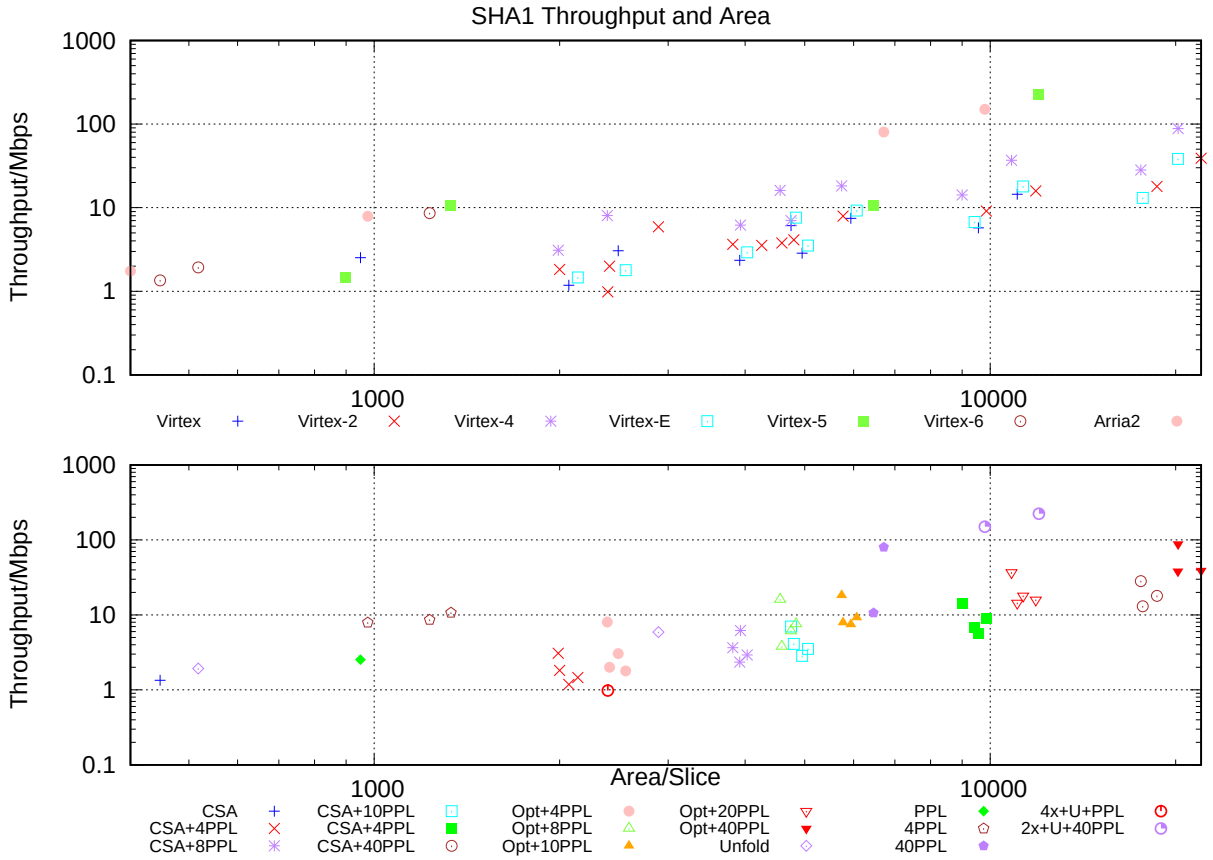

Figure 2.7. Throughput (Gbps) and Area (slice) comparison of the SHA-1 standard using the four main optimization methods

Fig.2.8 shows that the majority of the SHA-2 implementations and optimizations were carried out using Xilinx Virtex-2, Virtex-4, and Virtex-E FPGAs. The figure depicts Throughput and Area comparison of the SHA-2 hash standard regarding different optimization methods. The best results for throughput is the optimized 32-stages pipelined architecture. On another hand, the area requirements for 64-stages pipeline got more area slices than the other. In essence, the area requirements are increasing with the increase of the applied pipeline stage. Carry Save Adder (CSA) optimization shows more enhancements for the FPGA implementations of both SHA-1 and SHA-2 standards. The fact that SHA-1 and SHA-2 rely mainly on addition equations, CSA has a salient effect on their throughput. The SHA-3 benefited from the CSA on designs that relied on

the loop unfolding. Where, after unrolling (unfolding) the addition process appeared number of times, and therefore necessitates the need for the CSA. On the other hand, Loop-Unfold (Unroll) method does not show any enhancement unless it is combined with other methods.

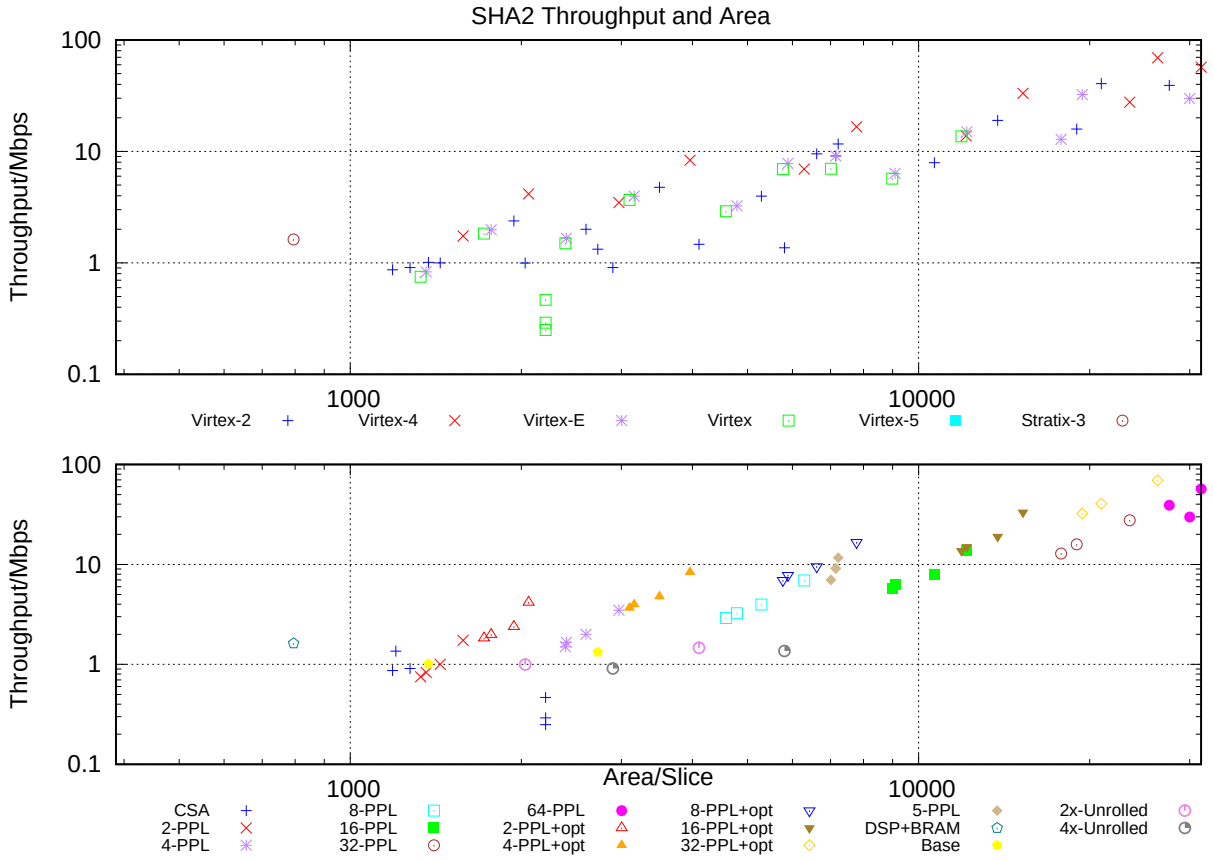

Figure 2.8. Throughput (Gbps) and Area (slice) comparison of the SHA-2 standard regarding the four main optimization methods

The SHA-3 hash standard is compared regarding throughput and area for different optimization methods, as shown in Fig.2.9. The first insight is the majority of the SHA-3 designs relied on the Xilinx Spartan-3, Virtex-4, Virtex-5, and Virtex-6 FPGAs. For throughput optimization, the best results were those of Xilinx Virtex-6 FPGA while regarding area, Virtex-5 got better results than the others. The figure shows the upper hand of the pipelining method over the other optimization techniques, in term of throughput. But, the abundant increase of area requirements comes along with the increase in the number of pipeline stages. On another hand, the FPGA resources give good results in term of throughput and significant enhancements of area requirements.

For instance, DSP+BRAM optimization method provides a moderate throughput value and good area figures but on different FPGA (Altera Stratix-3). The results show that some optimization methods are better to be implemented on specific hardware to achieve the required outcomes. Loop-Unfold optimization reflects the need to be combined with other optimization methods for a better performance.

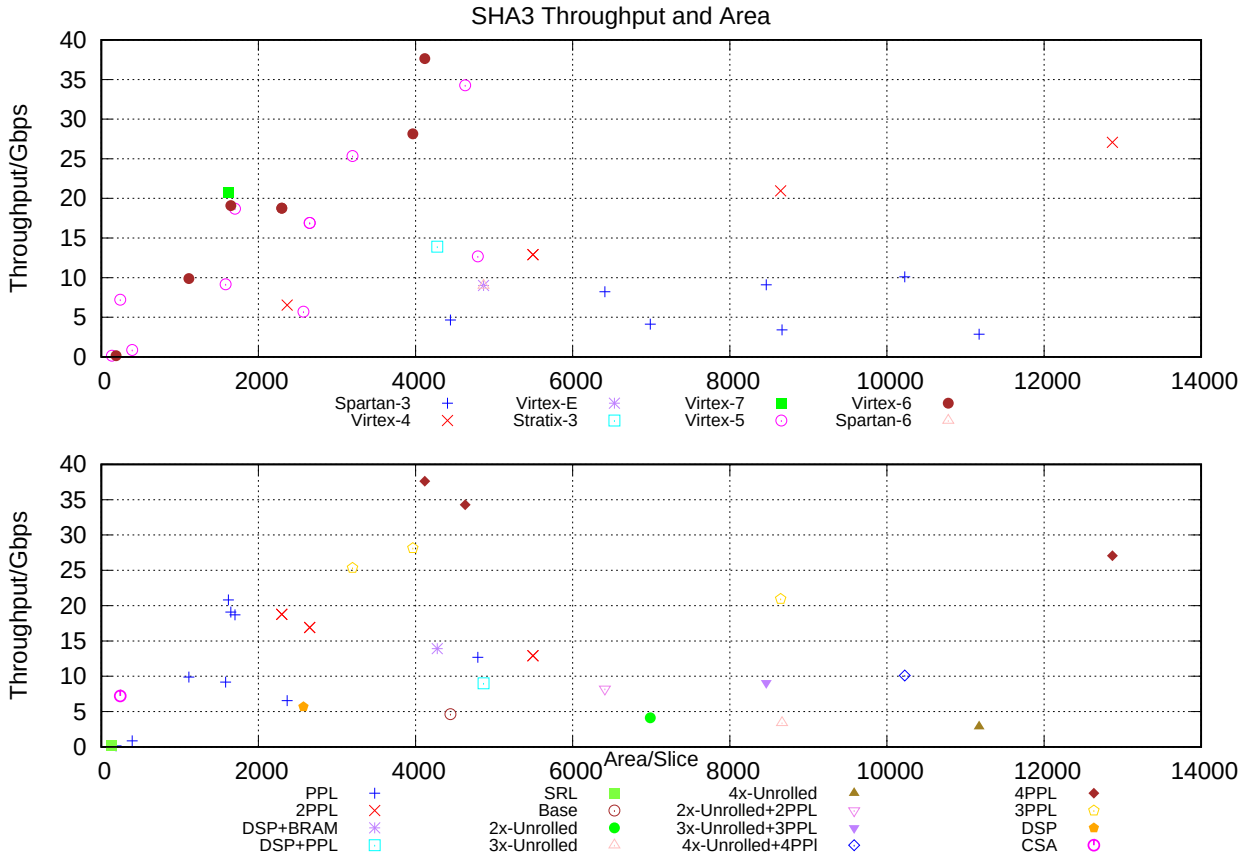

Figure 2.9. Throughput (Gbps) comparison of the SHA-3 standard regarding the four main optimization methods

In the case of other optimization methods, Fig.2.10 shows the optimization methods that are not using the main four components (CSA, PPL, Unfold, and FPGA-Resources). The bottom graph shows that some optimizations combine the SHA-1 and SHA-2 standard, which represented by the numbers 1 and 2. SHA-3 retains the upper hand over SHA-1 and SHA-2 as shown in the figure. The majority of the optimizations were implemented using Virtex-5, Virtex-6, and Stratix-

3 FPGAs. SHA-1 gives a comparable result when optimized using one time unrolling with the hardware duplication.

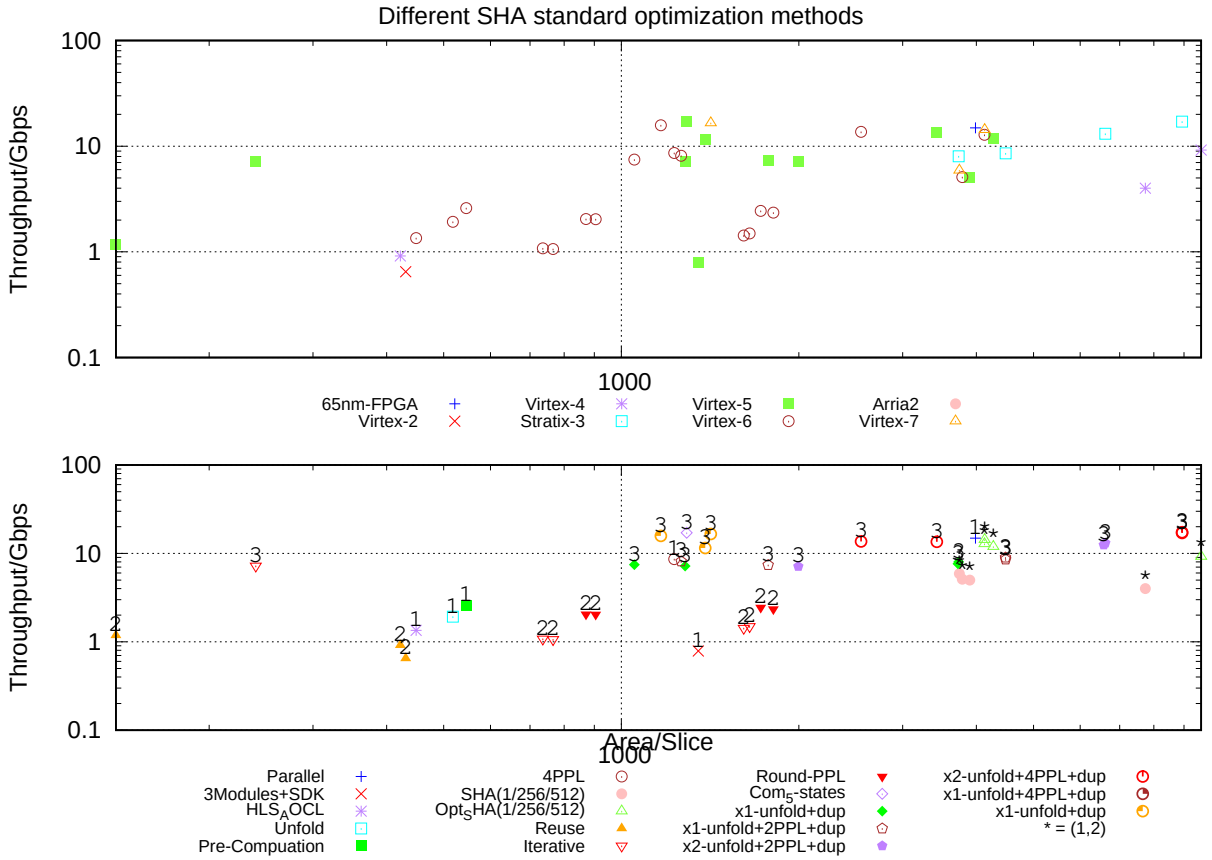

Figure 2.10. Throughput (Gbps) and Area (slice) comparison of All SHA standards regarding different optimization methods

Pipelining is used to optimize SHA-1 and SHA-2. A max of 4-stages pipelined architecture was adopted because of the nonlinear nature of SHA-1 and SHA-2 in the early steps, as discussed earlier. Without any proved optimum pipeline studies, the best of use the pipeline architecture is to increase the operating frequency, hence increasing the throughput but it augmented the area requirements. The authors of Keccak provided a prototype to support hardware implementation so that Keccak outperforms the others in the terms of Throughput and Area. However, the majority of Keccak implementations operate on a higher frequency than the SHA-1 and SHA-2, which in turn reflected on the increase of power consumption.

The hardware implementations of both SHA-1 and SHA-2 were not taken into consideration at the time of their design. The SHA-1 and the SHA-2 standards have non-linear equations on the early steps of calculations. The non-linearity of the equations opposed the hardware deployment methodologies, that affects the overall FPGA designs. The limited hardware support of Merkle-Damgard (MD) construction model diminishes the ability to exploit the FPGA hardware during the deployment of SHA-1 and SHA-2 standards. However, Keccak hash function, that won the selection competition of SHA-3 hash standard fully supports hardware deployment. Moreover, the authors of Keccak provided a prototype for implementing their winning algorithm [91].

There is a trade-off between different FPGA optimization methods. Some designs affect one requirement positively but make a negative impact on others. Therefore, in the case of negative impact, the combination of two or more optimization methods is reflected in the overall performance of a design. For instance, the loop unrolling method decreases the number of clock cycles needed for calculations, but it makes a negative impact on the throughput, as shown in Figures 2.7, 2.8, and 2.9. Combining the loop unrolling with the pipelining method produces a good enhancement in term of throughput with increased area requirements.

Figure 2.11 shows the relationships between the throughput and frequency for the three hash standards (SHA-1, SHA-2, and SHA-3). The x-axis represents the frequency and y-axis represents the throughput. The top part of each sub-figure represents the throughput with respect to the FPGA type, while the bottom part represents the throughput with respect to the optimization technique. The figure shows that the frequency and throughput have a proportional relationship, i.e., for a high throughput there is a higher frequency and vice versa.

The power consumption of an FPGA is determined by the sum of total static and dynamic power. The static power reflects the consumed power by the FPGA design and routing technology, and the dynamic power reflects the consumed power that is driven by the resource utilization of a design [92, 93]. The general power consumption equation is represented by (2.31).

$$P = \sum_{i} C_i.V_i^2.f, \tag{2.31}$$

where, for a resource i, C_i is the capacitance, V_i is the voltage, and f is the frequency. According to Figure 2.11, the frequencies of the SHA-1 hash standard are in the middle, and the frequencies

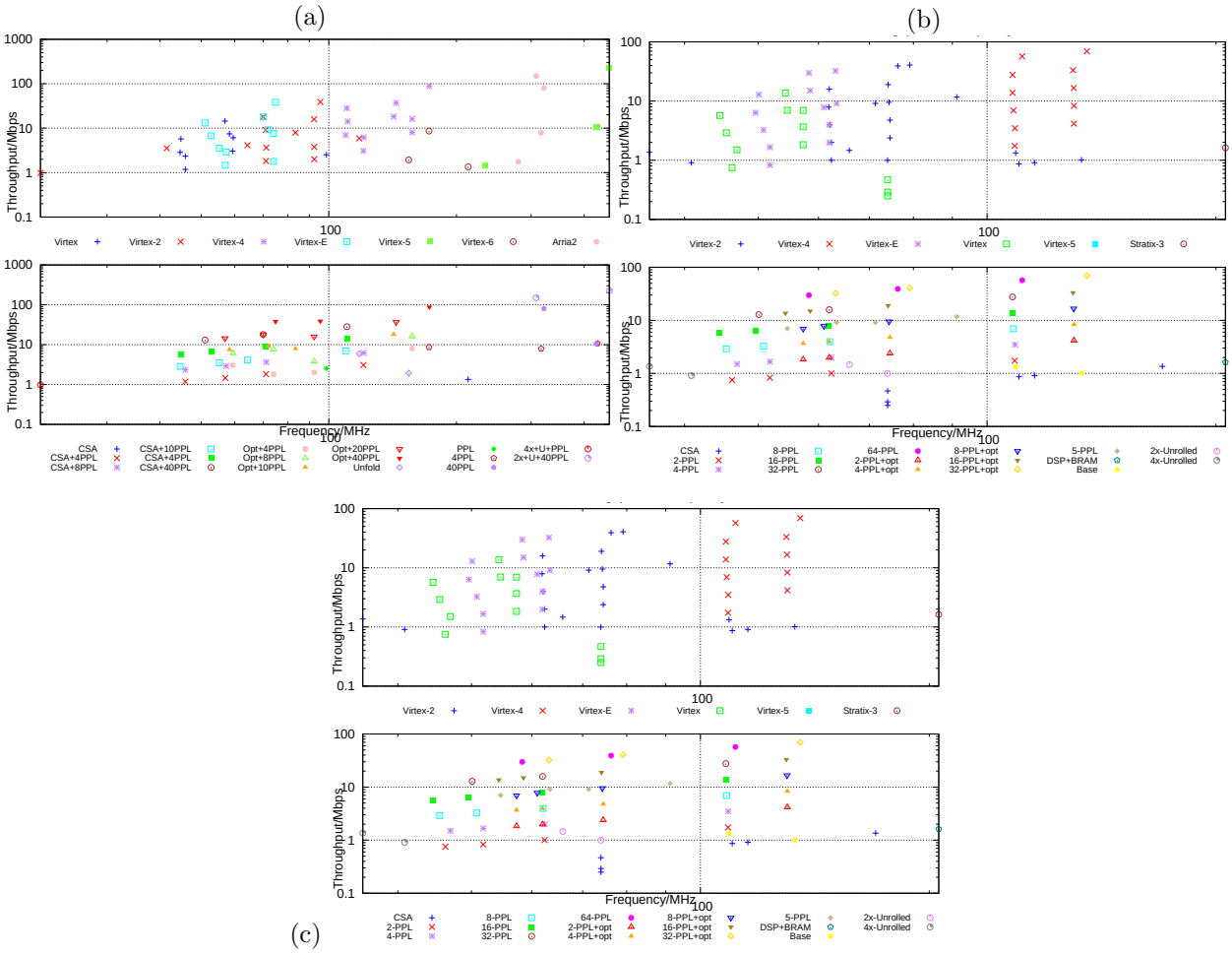

Figure 2.11. Throughput and Frequency relationships for SHA-1, SHA-2, and SHA-3 hash functions.

of the SHA-2 are to the left. This shows the lower frequencies that the SHA-1 and SHA-2 run, which in turn reflected on the total power consumption, as depicted in Equation (2.31). However, the SHA-3 frequencies are biased to the right, which infers that SHA-3 run higher frequencies than SHA-1 and SHA-2, and consequently, reflected to the total consumed power.

2.6. References

[1] Ronald L Rivest. "The MD4 message digest algorithm". In: *Conference on the Theory and Application of Cryptography*. Springer. 1990, pp. 303–311.

[2] Ronald Rivest and S Dusse. *The MD5 message-digest algorithm*. Tech. rep. MIT Laboratory for Computer Science, 1992.

[3] Xiaoyun Wang and Hongbo Yu. "How to break MD5 and other hash functions". In: *Annual international conference on the theory and applications of cryptographic techniques*. Springer. 2005, pp. 19–35.

[4] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. "Finding collisions in the full SHA-1". In: *Crypto*. Vol. 3621. Springer. 2005, pp. 17–36.

[5] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov. "The first collision for full SHA-1". In: *Annual International Cryptology Conference*. Springer. 2017, pp. 570–596.

[6] Secure Hash Standard. "FIPS PUB 180-2". In: *National Institute of Standards and Technology* (2002).

[7] NIST FIPS Pub. *180-3: Federal Information Processing Standards Publication, Secure Hash Standard (SHS)*. Tech. rep. Technical report, National Institute of Standards and Technology, 2008.

[8] Hans Delfs, Helmut Knebl, and Helmut Knebl. *Introduction to cryptography*. Vol. 2. Springer, 2002.

[9] Shu-jen Chang et al. "Third-round report of the SHA-3 cryptographic hash algorithm competition". In: *NIST Interagency Report* 7896 (2012).

[10] National Institute of Standards and Technology. "Secure Hash Standard (SHS) 180-4". In: *FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION* (Mar. 2012).

[11] NIST SHA. *Standard: Permutation-Based Hash and Extendable-Output Functions (DRAFT FIPS PUB 202)*. 2014.

[12] Penny Pritzker and Patrick D Gallagher. "SHA-3 standard: permutation-based hash and extendable-output functions". In: *Information Tech Laboratory National Institute of Standards and Technology* (2014), pp. 1–35.

[13] Brian Baldwin et al. "A hardware wrapper for the SHA-3 hash algorithms". In: *IET Irish Signals and Systems Conference (ISSC 2010)* (2010), pp. 1–6.

[14] Christopher Cullinan, Christopher Wyant, Timothy Frattesi, and Xinming Huang. "Computing performance benchmarks among cpu, gpu, and fpga". In: *Internet: www. wpi. edu/Pubs/E-project/Available/E-project-030212-123508/unrestricted/Benchmarking Final* (2013), p. 55.

[15] Veerpal Kaur and Aman Singh. "Review of Various Algorithms Used in Hybrid Cryptography". In: *International Journal of Computer Science and Network* (Dec. 2013), pp. 157–173.

[16] Miodrag Potkonjak. *Hardware based cryptography*. US Patent 8,379,856. Feb. 2013.

[17] Sundararaman Rajagopalan, Rengarajan Amirtharajan, Har Narayan Upadhyay, and John Bosco Balaguru Rayappan. "Survey and analysis of hardware cryptographic and steganographic systems on FPGA". In: *Journal of Applied Sciences* 12.3 (2012), p. 201.

[18] Kimmo Jarvinen, Matti Tommiska, and Jorma Skytta. "Comparative survey of high-performance cryptographic algorithm implementations on FPGAs". In: *IEE Proceedings-Information Security* 152.1 (2005), pp. 3–12.

[19] K Saravanan and A Senthilkumar. "Theoretical Survey on Secure Hash Functions and issues". In: *International Journal of Engineering Research & Technology (IJERT)* 2.10 (2013).

[20] Ricardo Chaves et al. "Secure Hashing: SHA-1, SHA-2, and SHA-3". In: *Circuits and Systems for Security and Privacy* (2016), pp. 81–107.

[21] Zhijie Shi, Chujiao Ma, Jordan Cote, and Bing Wang. "Hardware implementation of hash functions". In: *Introduction to Hardware Security and Trust*. Springer, 2012, pp. 27–50.

[22] Secure Hash Standard. *Federal information processing standards publication 180-1*. Apr. 1995.

[23] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. "The keccak sha-3 submission". In: *Submission to NIST (Round 3)* 6.7 (2011), p. 16.

[24] Bertoni Guido, Daemen Joan, P Michaël, and VA Gilles. *Cryptographic sponge functions*. Tech. rep. Keccak.Team Official Site, 2011, pp. 1–93.

[25] Christof Paar and Jan Pelzl. "SHA-3 and The Hash Function Keccak". In: *Understanding Cryptography A Textbook for Students and Practitioners, www. crypto-textbook. com* (2010).

[26] Guido Bertoni, Joan Daemen, Michael Peeters, Gilles Van Assche, and Ronny Van Keer. "Keccak implementation overview". In: *URL: http://keccak. neokeon. org/Keccak-implementation-3.2. pdf* (2012).

[27] Francisco Rodriguez-Henriquez, Nazar Abbas Saqib, Arturo Diaz Perez, and Cetin Kaya Koc. *Cryptographic algorithms on reconfigurable hardware*. Springer Science & Business Media, 2007.

[28] Svetlin A Manavski et al. "CUDA compatible GPU as an efficient hardware accelerator for AES cryptography". In: *Signal Processing and Communications* 2007 (2007).

[29] Sikhar Patranabis and Debdeep Mukhopadhyay. *Fault Tolerant Architectures for Cryptography and Hardware Security*. Springer, 2018.

[30] Jason Cong et al. "Understanding Performance Differences of FPGAs and GPUs". In: *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE. 2018, pp. 93–96.

[31] Imene Mhadhbi, Nejla Rejeb, Slim Ben Othman, Nabil Litayem, and Slim Ben Saoud. "Design methodologies impact on the embedded system performances: Case of cryptographic algorithm". In: *Computer Applications & Research (WSCAR), 2014 World Symposium on*. IEEE. 2014, pp. 1–6.

[32] Jonathan Rose, Abbas El Gamal, and Alberto Sangiovanni-Vincentelli. "Architecture of field-programmable gate arrays". In: *Proceedings of the IEEE* 81.7 (1993), pp. 1013–1029.

[33] Kris Gaj, Ekawat Homsirikamol, and Marcin Rogawski. "Fair and comprehensive methodology for comparing hardware performance of fourteen round two SHA-3 candidates using FPGAs". In: *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer. 2010, pp. 264–278.

[34] Ekawat Homsirikamol, Marcin Rogawski, and Kris Gaj. "Throughput vs. area trade-offs in high-speed architectures of five round 3 SHA-3 candidates implemented using Xilinx and Altera FPGAs". In: *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer. 2011, pp. 491–506.

[35] Paul Dillien. "And the Winner of Best FPGA of 2016 is.." In: *Principal, High Tech Marketing* (2017).

[36] Robert P McEvoy, Francis M Crowe, Colin C Murphy, and William P Marnane. "Optimisation of the SHA-2 family of hash functions on FPGAs". In: *Emerging VLSI Technologies and Architectures, 2006. IEEE Computer Society Annual Symposium on*. IEEE. 2006, 6–pp.

[37] George S Athanasiou, Harris E Michail, George Theodoridis, and Costas E Goutis. "Optimising the SHA-512 cryptographic hash function on FPGAs". In: *IET Computers & Digital Techniques* 8.2 (2013), pp. 70–82.

[38] HE Michail, Athanasios P Kakarountas, George N Selimis, and Costas E Goutis. "Optimizing SHA-1 hash function for high throughput with a partial unrolling study". In: *International Workshop on Power and Timing Modeling, Optimization and Simulation*. Springer. 2005, pp. 591–600.

[39] Bernhard Jungk and Jurgen Apfelbeck. "Area-efficient FPGA implementations of the SHA-3 finalists". In: *Reconfigurable Computing and FPGAs (ReConFig), 2011 International Conference on*. IEEE. 2011, pp. 235–241.

[40] Mohsen Bahramali, Jin Jiang, and Arash Reyhani-Masoleh. "A fault detection scheme for the FPGA implementation of SHA-1 and SHA-512 round computations". In: *Journal of Electronic Testing* 27.4 (2011), p. 517.

[41] Julia Drozd, Oleksandr Drozd, Valeria Nikul, and Julian Sulima. "FPGA implementation of vertical addition with a bitwise pipeline of calculations". In: *2018 IEEE 9th International Conference on Dependable Systems, Services and Technologies (DESSERT)*. IEEE. 2018, pp. 239–242.

[42] Kentaro Katayama, Hidetoshi Matsumura, Hiroaki Kameyama, Shinichi Sazawa, and Yasuhiro Watanabe. "An FPGA-accelerated high-throughput data optimization system for high-speed transfer via wide area network". In: *Field Programmable Technology (ICFPT), 2017 International Conference on*. IEEE. 2017, pp. 211–214.

[43] Colin Yu Lin, Zhenghong Jiang, Cheng Fu, Hayden Kwok-Hay So, and Haigang Yang. "FPGA High-level Synthesis versus Overlay: Comparisons on Computation Kernels". In: *ACM SIGARCH Computer Architecture News* 44.4 (2017), pp. 92–97.

[44] Vaughn Betz. "FPGA Architecture for the Challenge". In: *Disponivel no site: http://www. eecg. toronto. edu/˜ vaughn/challenge/fpga_arch. html. Acesso* 15 (2009).

[45] Sergio Lucia, Denis Navarro, Oscar Lucia, Pablo Zometa, and Rolf Findeisen. "Optimized FPGA implementation of model predictive control for embedded systems using high-level synthesis tool". In: *IEEE Transactions on Industrial Informatics* 14.1 (2018), pp. 137–145.

[46] R. K. Makkad and A. K. Sahu. "Novel design of fast and compact SHA-1 algorithm for security applications". In: *2016 IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)*. May 2016, pp. 921–925. DOI: 10.1109/RTEICT.2016.7807963.

[47] A. P. Kakarountas, G. Theodoridis, T. Laopoulos, and C. E. Goutis. "High-Speed FPGA Implementation of the SHA-1 Hash Function". In: *2005 IEEE Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications*. Sept. 2005, pp. 211–215. DOI: 10.1109/IDAACS.2005.282972.

[48] Shamsiah Suhaili, Takahiro Watanabe, and Norhuzaimin Julai. "High Speed and Throughput Evaluation of SHA-1 Hash Function Design with Pipelining and Unfolding Transformation Techniques". In: *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)* 9.3-10 (2017), pp. 19–22.

[49] Eun-Hee Lee, Je-Hoon Lee, Il-Hwan Park, and Kyoung-Rok Cho. "Implementation of high-speed SHA-1 architecture". In: *IEICE Electronics Express* 6.16 (2009), pp. 1174–1179.

[50] Yong Ki Lee, Herwin Chan, and Ingrid Verbauwhede. "Throughput optimized SHA-1 architecture using unfolding transformation". In: *Application-specific Systems, Architectures and Processors, 2006. ASAP'06. International Conference on*. IEEE. 2006, pp. 354–359.

[51] Harris E Michail et al. "Area-Throughput Trade-Offs for SHA-1 and SHA-256 Hash Functions' Pipelined Designs". In: *Journal of Circuits, Systems and Computers* 25.04 (2016), p. 1650032.

[52] T. Isobe, S. Tsutsumi, K. Seto, K. Aoshima, and K. Kariya. "10 Gbps implementation of TLS/SSL accelerator on FPGA". In: *2010 IEEE 18th International Workshop on Quality of Service (IWQoS)*. June 2010, pp. 1–6. DOI: 10.1109/IWQoS.2010.5542723.

[53] Nalini C Iyer and Sagarika Mandal. "Implementation of secure hash algorithm-1 using fpga". In: *Int. J. Inf. Comput. Technol* 3 (2013), pp. 757–764.

[54] Ian Janik and Mohammed AS Khalid. "Synthesis and evaluation of SHA-1 algorithm using altera SDK for OpenCL". In: *Circuits and Systems (MWSCAS), 2016 IEEE 59th International Midwest Symposium on*. IEEE. 2016, pp. 1–4.

[55] Harris E Michail, George S Athanasiou, George Theodoridis, and Costas E Goutis. "On the development of high-throughput and area-efficient multi-mode cryptographic hash designs in FPGAs". In: *Integration, the VLSI Journal* 47.4 (2014), pp. 387–407.

[56] Wanzhong Sun, Hongpeng Guo, Huilei He, and Zibin Dai. "Design and optimized implementation of the SHA-2 (256, 384, 512) hash algorithms". In: *ASIC, 2007. ASICON'07. 7th International Conference on*. IEEE. 2007, pp. 858–861.

[57] Anane Mohamed and Anane Nadjia. "SHA-2 hardware core for virtex-5 FPGA". In: *Systems, Signals & Devices (SSD), 2015 12th International Multi-Conference on*. IEEE. 2015, pp. 1–5.

[58] Ignacio Algredo-Badillo, C Feregrino-Uribe, René Cumplido, and Miguel Morales-Sandoval. "FPGA-based implementation alternatives for the inner loop of the Secure Hash Algorithm SHA-256". In: *Microprocessors and Microsystems* 37.6 (2013), pp. 750–757.

[59] R. P. McEvoy, F. M. Crowe, C. C. Murphy, and W. P. Marnane. "Optimisation of the SHA-2 family of hash functions on FPGAs". In: *IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures (ISVLSI'06)*. Mar. 2006, pp. 1–6. DOI: 10.1109/ISVLSI.2006.70.

[60] Hassen Mestiri, Fatma Kahri, Belgacem Bouallegue, and Mohsen Machhout. "Efficient FPGA Hardware Implementation of Secure Hash Function SHA-2". In: *International Journal of Computer Network and Information Security* 7.1 (2014), p. 9.

[61] Rommel Garcia, Ignacio Algredo-Badillo, Miguel Morales-Sandoval, Claudia Feregrino-Uribe, and Rene Cumplido. "A compact FPGA-based processor for the Secure Hash Algorithm SHA-256". In: *Computers & Electrical Engineering* 40.1 (2014), pp. 194–202.

[62] Manoj D Rote, N Vijendran, and David Selvakumar. "High performance SHA-2 core using the Round Pipelined Technique". In: *Electronics, Computing and Communication Technologies (CONECCT), 2015 IEEE International Conference on*. IEEE. 2015, pp. 1–6.

[63] Dipti Thakur and Utsav Malviya. "Low Power and Simple Implementation of Secure Hashing Algorithm (SHA-2) using VHDL Implemented on FPGA of SHA-224/256 Core". In: *International Journal of Engineering and Management Research (IJEMR)* 8.1 (2018), pp. 1–4.

[64] Kris Gaj, Ekawat Homsirikamol, Marcin Rogawski, Rabia Shahid, and Malik Umar Sharif. "Comprehensive Evaluation of High-Speed and Medium-Speed Implementations of Five SHA-3 Finalists Using Xilinx and Altera FPGAs." In: *IACR Cryptology EPrint Archive* 2012 (2012), p. 368.

[65] Farnoud Farahmand, Ekawat Homsirikamol, and Kris Gaj. "A zynq-based testbed for the experimental benchmarking of algorithms competing in cryptographic contests". In: *ReConFigurable Computing and FPGAs (ReConFig), 2016 International Conference on*. IEEE. 2016, pp. 1–7.

[66] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. "Keccak". In: *Annual international conference on the theory and applications of cryptographic techniques*. Springer. 2013, pp. 313–314.

[67] Alia Arshad, Arshad Aziz, et al. "Compact implementation of SHA3-512 on FPGA". In: *Information Assurance and Cyber Security (CIACS), 2014 Conference on*. IEEE. 2014, pp. 29–33.

[68] Hassen Mestiri, Fatma Kahri, Mouna Bedoui, Belgacem Bouallegue, and Mohsen Machhout. "High throughput pipelined hardware implementation of the KECCAK hash function". In: *Signal, Image, Video and Communications (ISIVC), International Symposium on*. IEEE. 2016, pp. 282–286.

[69] Harris E Michail, Lenos Ioannou, and Artemios G Voyiatzis. "Pipelined SHA-3 implementations on FPGA: Architecture and performance analysis". In: *Proceedings of the Second Workshop on Cryptography and Security in Computing Systems*. ACM. 2015, p. 13.

[70] George S Athanasiou, George-Paris Makkas, and Georgios Theodoridis. "High throughput pipelined FPGA implementation of the new SHA-3 cryptographic hash algorithm". In: *Communications, Control and Signal Processing (ISCCSP), 2014 6th International Symposium on*. IEEE. 2014, pp. 538–541.

[71] Lenos Ioannou, Harris E Michail, and Artemios G Voyiatzis. "High performance pipelined FPGA implementation of the SHA-3 hash algorithm". In: *Embedded Computing (MECO), 2015 4th Mediterranean Conference on*. IEEE. 2015, pp. 68–71.

[72] Jarosław Sugier. "Low cost FPGA devices in high speed implementations of K eccak-f hash algorithm". In: *Proceedings of the Ninth International Conference on Dependability and Complex Systems DepCoS-RELCOMEX. June 30–July 4, 2014, Brunów, Poland*. Springer. 2014, pp. 433–441.

[73] Rabia Shahid, Malik Umar Sharif, Marcin Rogawski, and Kris Gaj. "Use of embedded fpga resources in implementations of 14 round 2 sha-3 candidates". In: *Field-Programmable Technology (FPT), 2011 International Conference on*. IEEE. 2011, pp. 1–9.

[74] Jori Winderickx, Joan Daemen, and Nele Mentens. "Exploring the use of shift register lookup tables for Keccak implementations on Xilinx FPGAs". In: *Field Programmable Logic and Applications (FPL), 2016 26th International Conference on*. IEEE. 2016, pp. 1–4.

[75] George Provelengios, Paris Kitsos, Nicolas Sklavos, and Christos Koulamas. "FPGA-based design approaches of keccak hash function". In: *Digital System Design (DSD), 2012 15th Euromicro Conference on*. IEEE. 2012, pp. 648–653.

[76] Yusuke Ayuzawa, Naoki Fujieda, and Shuichi Ichikawa. "Design trade-offs in SHA-3 multi-message hashing on FPGAs". In: *TENCON 2014-2014 IEEE Region 10 Conference*. IEEE. 2014, pp. 1–5.

[77] Arshad Aziz et al. "A low-power SHA-3 designs using embedded digital signal processing slice on FPGA". In: *Computers & Electrical Engineering* 55 (2016), pp. 138–152.

[78] Nithin R. Chandran and Ebin M. Manuel. "Performance Analysis of Modified SHA-3". In: *Procedia Technology* 24.Supplement C (2016). International Conference on Emerging Trends in Engineering, Science and Technology (ICETEST - 2015), pp. 904–910. ISSN: 2212-0173. DOI: https://doi.org/10.1016/j.protcy.2016.05.168. URL: http://www.sciencedirect.com/science/article/pii/S2212017316302584.

[79] Muzaffar Rao, Thomas Newe, and Ian Grout. "Efficient high speed implementation of secure hash algorithm-3 on Virtex-5 FPGA". In: *Digital System Design (DSD), 2014 17th Euromicro Conference on*. IEEE. 2014, pp. 643–646.

[80] Muzaffar Rao, Thomas Newe, Ian Grout, and Avijit Mathur. "High speed implementation of a SHA-3 core on virtex-5 and virtex-6 FPGAs". In: *Journal of Circuits, Systems and Computers* 25.07 (2016), p. 1650069.

[81] S Bhargav and Drva Sharath Kumar. "Compact Implementation of SHA3-1024 on FPGA". In: *International Journal* 79 (2015).

[82] MM Sravani and CH Pallavi. "Design of Compact Implementation of SHA-3 (512) on FPGA". In: *International Research Journal of Engineering and Technology (IRJET)* 2.02 (2015), pp. 41–46.

[83] Fatma Kahri, Hassen Mestiri, Belgacem Bouallegue, and Mohsen Machhout. "High Speed FPGA Implementation of Cryptographic KECCAK Hash Function Crypto-Processor". In: *Journal of Circuits, Systems and Computers* 25.04 (2016), p. 1650026.

[84] Giacinto Paolo Saggese, Antonino Mazzeo, Nicola Mazzocca, and Antonio GM Strollo. "An FPGA-based performance analysis of the unrolling, tiling, and pipelining of the AES algorithm". In: *International Conference on Field Programmable Logic and Applications*. Springer. 2003, pp. 292–302.

[85] Harris E Michail, George S Athanasiou, George Theodoridis, Andreas Gregoriades, and Costas E Goutis. "Design and implementation of totally-self checking SHA-1 and SHA-256 hash functions' architectures". In: *Microprocessors and Microsystems* 45 (2016), pp. 227–240.

[86] Harris E Michail, Apostolis Kotsiolis, Athanasios Kakarountas, George Athanasiou, and Costas Goutis. "Hardware implementation of the Totally Self-Checking SHA-256 hash core".

In: *EUROCON 2015-International Conference on Computer as a Tool (EUROCON), IEEE.* IEEE. 2015, pp. 1–5.

[87] Fatma Kahri, Hassen Mestiri, Belgacem Bouallegue, and Mohsen Machhout. "An efficient fault detection scheme for the secure hash algorithm SHA-512". In: *Green Energy Conversion Systems (GECS), 2017 International Conference on.* IEEE. 2017, pp. 1–5.

[88] Imtiaz Ahmad and A Shoba Das. "Analysis and detection of errors in implementation of SHA-512 algorithms on FPGAs". In: *The Computer Journal* 50.6 (2007), pp. 728–738.

[89] Nithin R Chandran and Ebin M Manuel. "Performance Analysis of Modified SHA-3". In: *Procedia Technology* 24 (2016), pp. 904–910.

[90] Hassen Mestiri, Fatma Kahri, Belgacem Bouallegue, Mehrez Marzougui, and Mohsen Machhout. "Efficient countermeasure for reliable KECCAK architecture against fault attacks". In: *Anti-Cyber Crimes (ICACC), 2017 2nd International Conference on.* IEEE. 2017, pp. 55–59.

[91] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. "The making of KECCAK". In: *Cryptologia* 38.1 (2014), pp. 26–60.

[92] Peggy Abusaidi, Matt Klein, and Brian Philofsky. "Virtex-5 FPGA system power design considerations". In: *Xilinx WP285 (v1. 0) February* 14 (2008).

[93] Najeem Lawal, Fahad Lateef, and Muhammad Usman. "Power consumption measurement & configuration time of FPGA". In: *2015 Power Generation System and Renewable Energy Technologies (PGSRET).* IEEE. 2015, pp. 1–5.

3. MITIGATION AND IMPROVING SHA-1 STANDARD USING COLLISION DETECTION APPROACH

3.1. Introduction

Secure Hash Algorithms (SHAs) started in 1991 with the first MD4 hash function by Rivest[1]. An optimized version (MD5) was released two years later[2]. An improved version with longer hash length (SHA-0) was developed in 1993, and for security issues, SHA-1 was announced as the official standard in 1995 by the National Institute of Standards and Technology (NIST) [3].

Cryptographic hash functions are widely used in applications, starting from simple password implementation to message authentication over unsecure network. The cryptanalyst tries to verify the strength of hash functions by many tools and techniques. Three challenges exist to verify the completeness of any hash standard: preimage, 2nd preimage and collision resistance properties. Preimage resistance means to easily obtain the hash from a given message, but difficult to extract it back from the computed hash of the message. Second preimage resistance means that it is difficult to find two messages M1 and M2 generating the same digest (Hash). While collision resistance property means that a hash function resists any possibility to generate the same output hash for two different messages or more [4].

SHA-1 is still being used by many entities for data authentication and integrity, e.g., digital signature verification. However, many researches proved that the SHA-1 is exposed to a collision attack through thoroughly efforts to find any hiatus lead to break the hash system[5, 6, 7]. For instance, the collision attack against the SHA-1 hash standard has gradually been obtained, such as 40-steps collision [8], 53-steps [9], 64-steps [10], 76-steps [7], and 80-steps full collision attacks as presented by Wang *et al.* in [11].

However, in the situation of weak messages, the old hash case appears. As some old hashes are not replaceable to new standards. Moreover, the data verifiers continue to accept weak and

The content of this chapter has been published in the IEEE 16th International Conference on Frontiers of Information Technology (FIT2018). The material in this chapter was co-authored by Zeyad Al-Odat, Mazhar Ali and Samee Khan. Zeyad Al-Odat had primary responsibility for conducting experiments and collecting results. Zeyad Al-Odat was the primary developer of the conclusions that are advanced here. Zeyad Al-Odat also drafted and revised all versions of this chapter. Mazhar Ali drafted and revised all versions of this chapter. Samee U. Khan revised the material and served as proofreader.

malicious messages for long time to come [12]. Therefore, the need for an improved version of the SHA-1 or a counter collision method arose to protect primitives that are still using the SHA-1 standard.

In this chapter, we are proposing a counter cryptanalyst technique to help protect the SHA-1 standard against the collision attack. The rest of chapter is organized as follows. Section 2 gives brief descriptions about the SHA-1 and collision attack. In Section 3, the literature review of previous works is discussed. The proposed designs are presented in Section 4. Section 5 concludes the chapter.

3.2. Preliminaries

Some preliminaries need to be addressed to better understand the proposed system. In the subsequent text brief descriptions about the SHA-1 standard, the collision attack, and threat model are presented.

3.2.1. Brief Description about the SHA-1

The SHA-1 standard follows Merkle-Damgård construction [13]. The SHA-1 takes a message of length less than 2^{64}, divides it into equal blocks, and processes it sequentially to produce 160-bit hash output. The SHA-1 produces 160-bit output digest, to do so the message goes through several steps and compression operations before the output hash is produced, as depicted in Fig. 3.1.

The SHA-1 processes a given message according to the following steps:

1. **Message padding**: In this step, "1" is appended at the end of the input message M and followed by least number of 0's until it congruent to 448 Mod 512. The size of the original message M is appended as big-endian 64-bit, as seen in Fig. 3.2. Then the resulting padded message becomes $\widehat{M} = N * 512$, for real $N \geqslant 1$.

2. **Message divide**: In this step, the padded message is divided into equal size blocks ($\widehat{M}$ into N 512-bit blocks $M_0, M_1, ..., M_{N-1}$).

3. **Initial Hash Values**: The SHA-1 maintains five 32 bits IHVs (H0, H1, H2, H3 and H4) initialized with fixed hexadecimal values (67452301_{16}, $EFCDAB89_{16}$, $98BADCFE_{16}$, 10325476_{16}, $C3D2E1F0_{16}$), respectively. Moreover five 32 bits working state variables (A,B,C,D and E) initialized with the values of IHVs accordingly.

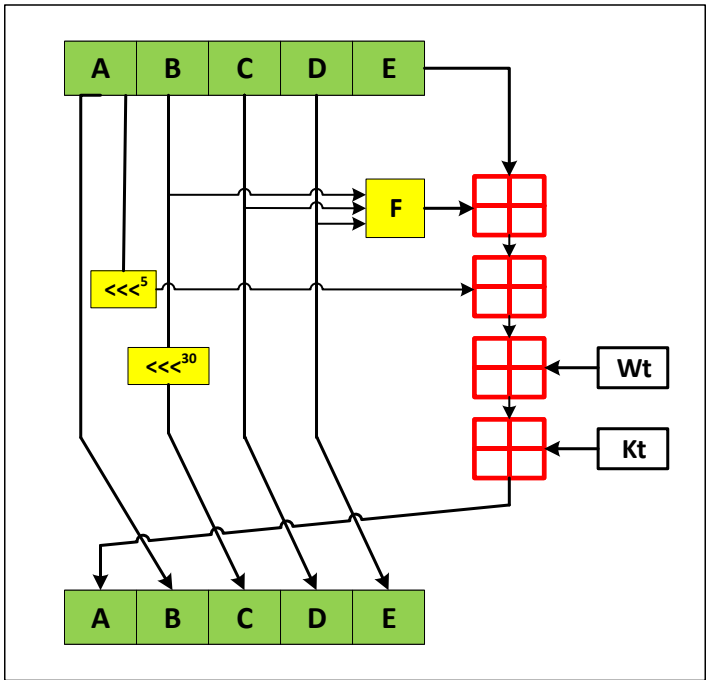

Figure 3.1. SHA-1 general structure, takes message of length $< 2^{64}$, padd it, then divide it into equal size blocks.

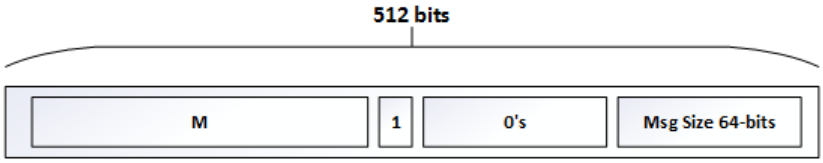

Figure 3.2. Message padding mechanism

4. **Processing**: To compute the hash value of N blocks message, the process goes through the SHA-1 compression function for N+1 states (state for each block plus the initial state), starting with IHVs.

5. **Message Expansion and Working State Variables**: Each block is divided into 16 32-bit words. Then, these words are expanded into 80 32-bit words using the message expansion equation, as shown in (3.1). The 16 message block words are initialized with the messages block, then they are used to calculate the rest of expansion equation to get words W[17] to W[79].

$$W_j = \begin{cases} M_i^{(j)} & ,0 \leq j \leq 15 \\ (W_{j-16} \oplus W_{j-16} \oplus W_{j-16} \oplus W_{j-16} \oplus)_{\lll 1} & ,16 \leq j \leq 79 \end{cases}. \tag{3.1}$$

Where, M_i^j is the j^{th} word of block i, $\oplus$ logical XOR, and $\lll_n$ left rotation by n-bit.

6. **Internal Process**: The compression function consists of 80 steps. Each 20 steps conform a round, and each round has distinct round function and constant K_i, as depicted in Table 3.1. Besides, each group has a distinct round constant (K_i) and manipulator function (f) are used to calculate the output hash. The working state variables change after each step of the 80-steps according to the values below.

$$\begin{aligned}
\mathrm{A_t} &= \mathrm{RL^5(A_{t-1}) + F_t(B_{t-1}, C_{t-1}, D_{t-1}) + E_{t-1} + W_t + K_t} \\
\mathrm{B_t} &= \mathrm{A_{t-1}} \\
\mathrm{C_t} &= \mathrm{RL^{30}(B_{t-1})} \\
\mathrm{D_t} &= \mathrm{C_{t-1}} \\
\mathrm{E_t} &= \mathrm{D_{t-1}}.
\end{aligned}$$

Where, $RL^n(x)$ is a left rotation of word x by n-bits, and $F(B, C, D)$ is the logical function equation.

7. **Hash Output**: The output hash is calculated by adding the output of the last step to the initial hash values (H0, H1, H2, H3 and H4). If the current block is the last block, then the output hash is the concatenation of the five hash values together. Otherwise, the new IHV will be fed as an initial value for the next block calculation. The output hash is represented as:

$$H0 \parallel H1 \parallel H2 \parallel H3 \parallel H4.$$

3.2.2. SHA-1 Differential Attack

The main goal of the SHA-1 collision attack is to find two or more messages that lead to the same output hash. All current researches on finding the collision are basically built upon the methodology described by Wang *et al.* [11]. Wang's finding drew the road to the real collision project that is recently announced by the collaboration of Marc *et. al.* and Google [6]. The aforesaid project found the first real collision attack that broke the SHA-1 hash function [6, 7, 14].

Table 3.1. SHA-1 round functions and constants

Round and Steps	Round Function F (B,C,D)	Round Constant (Ki)
Round1 (0-19)	$(B \wedge C) \vee (\neg B \wedge D)$	*0x5A827999*
Round2 (20-39)	$(B \bigoplus C \bigoplus D)$	*0x6ED9EBA1*
Round3 (40-59)	$(B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$	*0x8F1BBCDC*
Round4 (60-79)	$(B \bigoplus C \bigoplus D)$	*0xCA62C1D6*

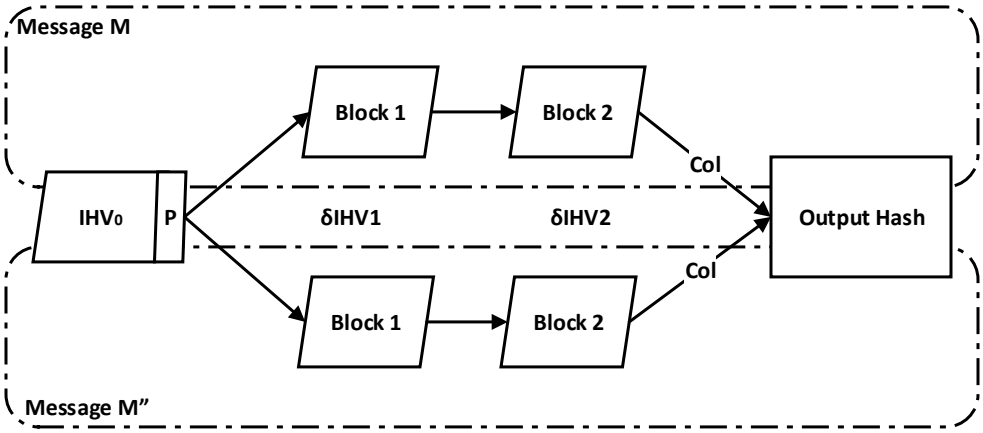

Figure 3.3. Two blocks collision attack.

The procedure depends on finding a disturbance vector which includes a real path among compression operations that lead to a collision. Disturbance vector is an (80×32) vector that contains the modular differences between Message M and $\widehat{M}$ (so that it called differential attack). Fig. 3.3 gives an explanation of two blocks collision attack. For both messages (M and $\widehat{M}$), the differences are calculated after each step. These differences are constructed from the internal block differences after each step computation. Each difference must meet predefined conditions, once met the block difference is added to the disturbance vector. For more details about disturbance vector and differential attack, we refer to [11, 5, 15].

3.2.3. Threat Model

The threat model is the general data flow from attacker perspective. Fig. 3.4 shows a general thread model for a document that contains essential information. A user, through operating system process, digitally signs a document. This signature is the SHA-1 hash of the document. The document is sent through Internet to a third party. The attacker in the middle crafted a

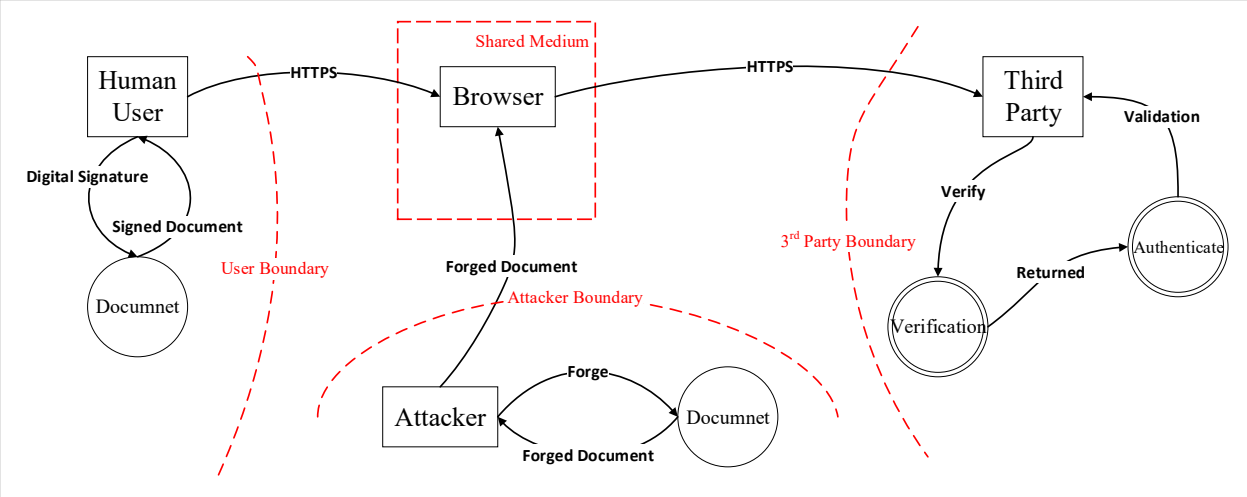

Figure 3.4. Threat model of the proposed system

message to produce the same hash value with different information than the original owner. The third-party user receives the document for verification. The document from the attacker is verified and authorized as it contains a valid hash value.

3.3. Literature Review

The first differential attack was proposed by Chabaud and Joux [16]. They were the first to introduce the idea of differential path taking the XOR differences between messages, as seen in Fig. 3.5. Their work was applied on two message's blocks, and the XOR difference is calculated after each round of computations. The messages were prone to collision by reducing the XOR difference, $\Delta 2 = 0$. Another convenient approach, was the work presented by Wang *et al.* [17]. Wang presented an attack on different hash functions (MD4, MD5, HAVAL, RIPEMD and SHA-0), they were able to break all of them. The attack was basically dependent on modular difference (not the XOR difference previously proposed in [16]). Also, using message modification techniques, these differences were made equal to zero. The result was a feasible attack on all aforesaid hash functions. On the other hand, in [5] Wang *et al.* proposed an improved version of the previously proposed attack on the SHA-0 by building the differential path according to pre-specified conditions.

Manuel *et al.* in [18] proposed an attack on the SHA-0 within one hour. The work relied on the same concept of differential attack from [17]. The author got benefited from the Boomerang Attack by finding optimal differential vector rather than the one used by Wang *et al.* The proposed work found an attack with a claimed complexity of $2^{33.5}$, and one hour of operation to get the final

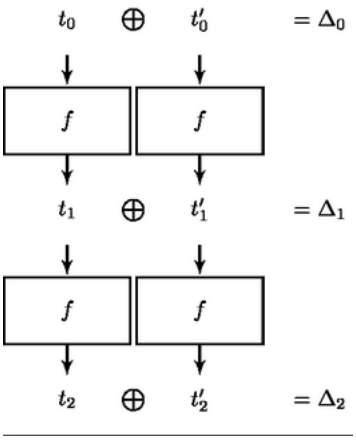

Figure 3.5. The general concept of the differential attack.

hashes, neglecting the time used to compute the optimal differential vector. Following the same approach, Wang *et al.* in [11] announced a new collision attack on full SHA-1 with complexity of 2^{69}. Their approach depends on the previous research on the SHA-0 and MD5, which uses the differential path of modular differences to construct a disturbance vector.

The big contribution on this area was the work done by Marc *et al.* [19]. Their work lead to the real collision attack on the SHA-1. The work was the base for the collision attack on the SHA-1 that has been lately published [6]. The attack generates two different pdf files that have the same hash.

The efforts to counter the collision attack were discussed in few literature [20, 12, 21]. In [20] a method for detecting and preventing collision of hashes during data transmission was presented. The authors registered their findings as a patent with number: $US8,086,860 - B2$. The proposed work comprises four steps: shuffling of bits, compression, T-functioning, and linear feedback shift registration (LFSR). The framework adds more haphazardness to the produced hash information to avoid collision. Notwithstanding, the produced hashes differ from if they were generated from the original SHA-1, regardless of whether the original hash out of collision-suspicious. Stevens *et al.* in [12] proposed an algorithm to detect the occurrence of collision for the Flame attack. Flame attack is a kind of collision that is used to breach windows update patch. Their work depends on previous published disturbance vectors that leads to a collision attack. The authors used the top published disturbance vectors to detect the collision before it takes place and invalidates the output hash once the message is marked as a suspicious one. The proposed work is compatible with the

MD5 and SHA-1 hash standards. lately, Stevens *et al.* in [21] proposed a speedup mechanism to detect the collision attack on the SHA-1 based on unavoidable bit condition. Their work leads to a significant speedup over the previous proposed work [12]. The claimed speed is 1.96 times slower than the original SHA-1 compression function.

The main goal of our work is to protect the entities that still use the SHA-1 hash function in their architectures. For backward compatibility there is a need to detect collisions for the SHA-1.

3.4. Proposed Methodology

The idea of counter collision attack is depicted in Fig. 3.6. The input message is processed and checked using collision detection mechanism, which we will be discuss later in this section. Once the algorithm returns true, then the output hash will be the truncated SHA-512/160, otherwise the regular SHA-1 hash will be passed to the output.

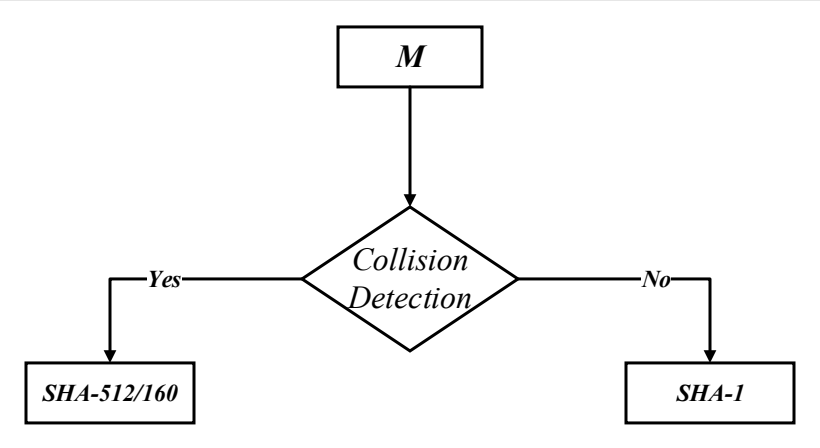

Figure 3.6. General architecture for the improved SHA-1

Collision detection algorithm returns `True` if a collision attack is detected, and false otherwise. Once a message that crafted using a collision attack is detected, then the algorithm will decline the hash output from the regular SHA-1. In our design, we chose the SHA2-512 to replace the hash calculation of the weak messages, because the SHA2-512 is the strongest version of SHA-2 standard. Moreover, the SHA-2 standard follows the same construction model as the SHA-1 [22].

3.4.1. Proposed Work

We are presenting two approaches for counter collision attack. The first one relies on Marc Stevens approach, which presented in [12]. The second design is our proposal. But before we

go through the details of both approaches, an important insights about the backward expansion equation needs to be addressed. The backward expansion equation plays a major rule in both approaches.

3.4.1.1. Backward Expansion Equation

The backward expansion equation was defined by Manuel *et al.* in [23]. The expansion of a message can be calculated by either side (forward or backward), and the initial hash value (IHV) can be extracted if sufficient working state variables are available. Equation (3.2) shows the backward expansion equation, it uses the same number of terms as the forward expansion equation (3.1), as described in Section 2. The backward expansion equation expands in backward direction to obtain $W_{-64,...,W_{-1}}$.

$$W_{i-16} = RR(W_t, 1) \oplus W_{t-3} \oplus W_{t-8} \oplus W_{t-14}. \tag{3.2}$$

Where t takes the values between $16 \leqslant t \leqslant 79$.

When both expansion equations (backward and forward) combined together, 144 words of 32-bit are constructed as follow:

$$W_{-64}, ..., W_{-1}, W_0, ..., W_{15}, W_{16}, ..., W_{79}.$$

According to the aforesaid combination, any sequence, of eighty consecutive 32-bit words, is a valid expansion vector [23].

3.4.1.2. First Approach (Employing Marc Stevens's Mechanism)

The first design, we employ the approach presented by Marc *et al.* in [12]. Fig. 3.7 shows the general structure of the Marc Stevens Mechanism. The process starts with any IHV_k (no need to be the first one). IHV_k is passed to the working state variable[1]. After processing the working state variables of the corresponding round, the IHV_{k+1} is obtained. Processing steps include: message expansion, 80-steps compression function calculation and message differences comparison. After each step, the value WS_i is used alongside with the values of tuple $((\delta B_i, \delta WS_i, i))$ to generate the sibling message $\widehat{M}$. Where δB_i is the block difference in step i, δWS_i is the working state difference , i is the current step. For each possible combination of triples $(\delta B, i, \delta WS_i)$, the sibling message $\widehat{M_i}$ is extracted. From the extracted sibling message $\widehat{M_i}$, the compression function calculations are

[1]Marc *et. al.* called them WS_s, and we pointed to them with letters A, B, C, D, and E

carried out to compute IHV_{k+1}'. Then, both values IHV_{k+1}' and IHV_{k+1} are compared together for equality. Once the two values of IHV_{k+1} and IHV_{k+1}' are equal, then the original message is crafted with a collision attack and the output hash will be declined. After that, for the messages that give a match possibility for their hashes, the SHA-512 truncated to 160 is computed for both messages to get different hash values.

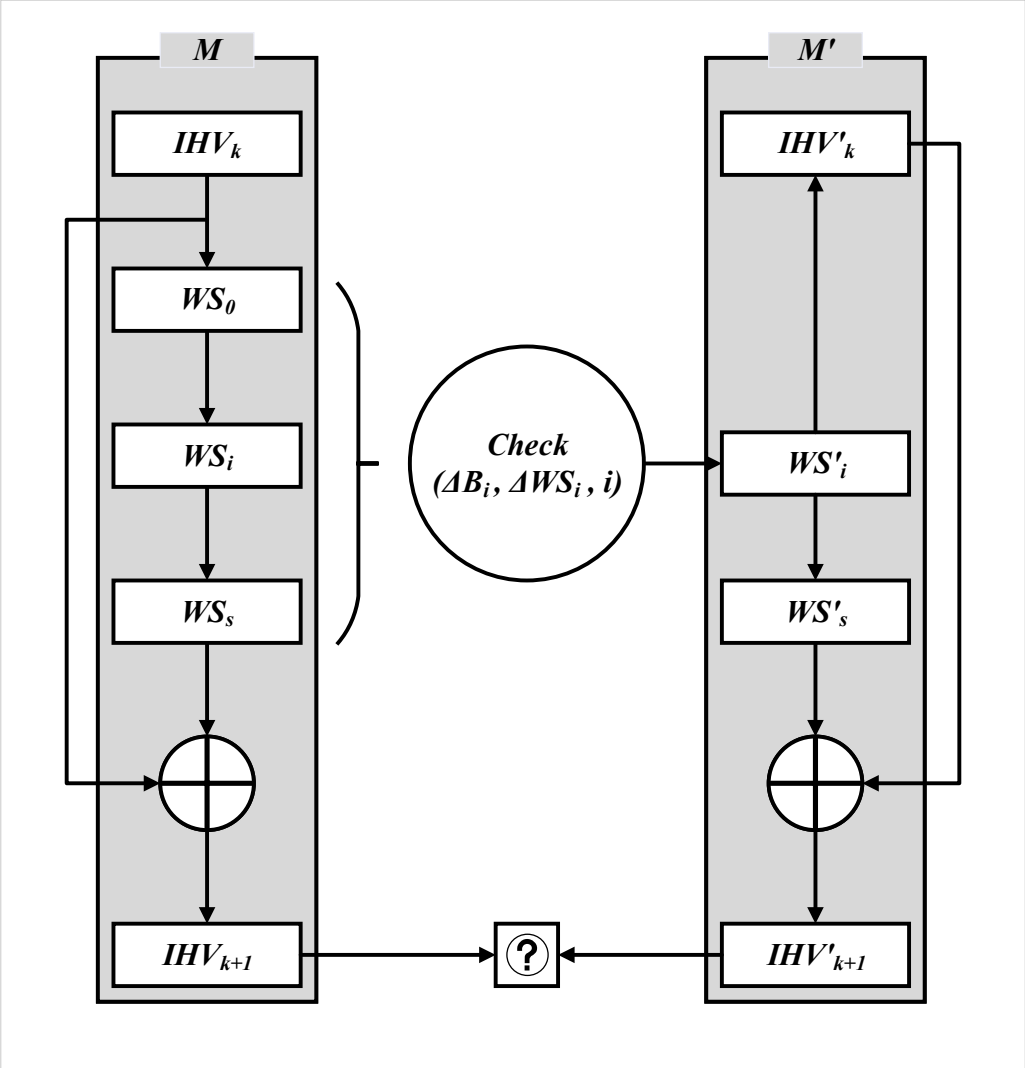

Figure 3.7. First approach: Marc Stevens collision detection mechanism.

3.4.1.3. Second Approach (Our Proposal)

According to Wang *et al.*, all disturbance vectors were built on the two blocks near collision [11]. We can get benefited from this idea and instead of comparing the block differences

after each step in the compression calculation (as stated by Marc Stevens), we propose the idea in Fig. 3.8. The methodology goes through the compression calculations for two blocks. The process starts with IHV_0, through 80-steps calculation compression function, to calculate IHV_1. IHV_1 is used as an input to the second block calculation to get IHV_2. For messages that were crafted with a collision attack, the backward computation of $IHV_2' = \delta IHV + IHV_2$ produces IHV_0' equal to IHV_0. Therefore, for all previously published disturbances, the output values of the second block are added to the values of each disturbance vector (δIHV), one at a time. Then calculate the backward expansion equation as depicted in (3.3). Applying backward expansion equation gives us the new value IHV_0' that is compared with the original IHV_0. If both values (IHV_0 and IHV_0') are equal, then a collision is detected. Instead of declining the output, the truncated SHA-512/160 is calculated to replace the hash value of the weak message.

$$W_{i-16} = RR(W_t, 1) \oplus W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \tag{3.3}$$

In the second approach, instead of reproducing the message M', we only generate the initial hash value of the sibling message without any need to have the message itself or having the IHV_{k+1}. This saves time and can be used to search for message siblings that might collide with the original one.

3.5. Results and Discussions

Both of the above-mentioned approaches were tested and verified for part of published disturbance vectors. Regarding the disturbance vectors that previously published by Wang *et. al.* [11], both algorithms work and produce a hash value SHA-512/160 for the the messages that were crafted using the selected disturbance vectors.

Table 3.2 represents two messages that collide and produce the same output hash after 58-steps of calculations. Our proposed designs can be applied for the shown messages. After applying the collision detection approaches, we were able to detect the collision possibility between the two messages. In spite of the two input messages look close to each other with minor changes. Tables 3.3 and 3.4 represent the hash values for the two PDF files that were reported by Marc *et al.* as an example of real SHA-1 collision. The two files are available for open access in [24], and the result of computing the SHA-1 hash for both of them leads to the same output. After applying

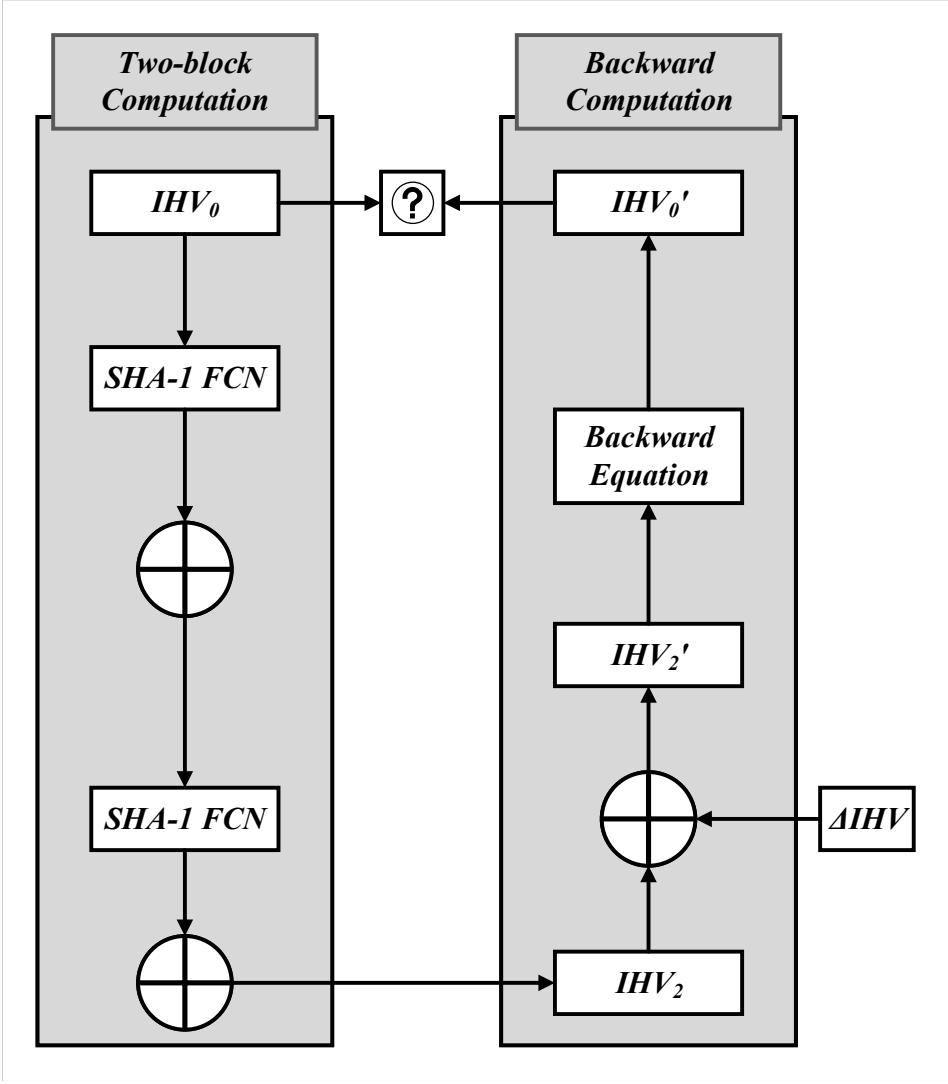

Figure 3.8. The proposed collision detection approach.

the collision detection mechanisms for the two files, a collision warning is generated for the regular hash calculation. Then, the truncated SHA-512 to 160 hash value will be produced instead the SHA-1 value.

For the proposed schemes, it is possible to add more disturbance vectors to the detection algorithms whenever any new published vector leads to a collision. Moreover, each disturbance vector has a probability of order 2^{-70} false positive occurrence [12], where a message is considered a weak while it is not. But, with such probability, the false positive occurrence is negligible. The designs were tested on laptop computer with Intel(R) Core (TM) i7-2640M CPU @ 2.80GHz and 8GB of RAM Under Linux (Ubuntu 18.04 LTS) platform. The testing codes were written under *C*

Table 3.2. Example of two messages that collide at step 58 (all values are in Hexadecimal format)

Message	Hexadecimal representation
M :	132b5ab6 a115775f 5bfddd6b 4dc470eb 0637938a 6cceb733 0c86a386 68080139 534047a4 a42fc29a 06085121 a3131f73 ad5da5cf 13375402 40bdc7c2 d5a839e2
M':	332b5ab6 c115776d 3bfddd28 6dc470ab e63793c8 0cceb731 8c86a387 68080119 534047a7 e42fc2c8 46085161 43131f21 0d5da5cf 93375442 60bdc7c3 f5a83982
Hash :	9768e739 b662af82 a0137d3e 918747cf c8ceb7d4

Table 3.3. SHA-1 values of two pdf files reported by Marc. *et. al.*

File name	Hash value
shattered-1.pdf :	38762cf7 f55934b3 4d179ae6 a4c80cad ccbb7f0a
shattered-2.pdf :	38762cf7 f55934b3 4d179ae6 a4c80cad ccbb7f0a

programming language. To validate our scheme an example with a real collided messages that were built with collision attacks are fed to the designs to test the validity of the system. We were able to detect the collision and produce collision-free hash values for the weak messages, as depicted in Tables 3.2, 3.3, and 3.4.

3.6. Conclusions

In this chapter, we presented two methods to improve the SHA-1 standard against collision attack. The first design relies on Stevens's approach for detecting the SHA-1 collision attack, in which the input message is checked against collision possibility according to three values $(\delta W_s, \delta B_i, i)$. These values belong to the previously published works of disturbance vectors that lead to a collision. After each round, the system is checked for the aforesaid three values that are used to extract the sibling message from a given one. Finally, compares IHV_{k+1} ,IHV_{k+1}' of both messages, if they were equal then the original message is crafted with collision forgery. The second approach is based on two blocks collision and the backward expansion equation. The initial hash value (IHV_0) is processed using 80 steps compression function. Then applying backward expansion equation to get the IHV_0'. For the messages that are crafted with collision attack, both IHV_0

Table 3.4. SHA-1 (SHA-512/160) using the proposed design

File name	Hash value
shattered-1.pdf :	42180282 5d3587fe 185abf70 9669bb96 93f6b416
shattered-2.pdf :	1af8b65d 6a0c1032 e00e48ac e0b4705e edcc1bab

and IHV_0' will be equal. The truncated SHA-512/160 is suggested to replace suspicious message's hashes.

3.7. References

[1] Ronald L. Rivest. "The MD4 Message Digest Algorithm". In: *Proceedings of the 10th Annual International Cryptology Conference on Advances in Cryptology*. CRYPTO '90. Berlin, Heidelberg: Springer-Verlag, 1991, pp. 303–311. ISBN: 3-540-54508-5. URL: http://dl.acm.org/citation.cfm?id=646755.705223.

[2] Ronald Rivest and S Dusse. *The MD5 message-digest algorithm*. Tech. rep. MIT Laboratory for Computer Science, 1992.

[3] Patrick Gallagher and Acting Director. "Secure hash standard (shs)". In: *FIPS PUB* (1995), pp. 180–3.

[4] Muhammad Barham, Orr Dunkelman, Stefan Lucks, and Marc Stevens. "New second preimage attacks on dithered hash functions with low memory complexity". In: *International Conference on Selected Areas in Cryptography*. Springer. 2016, pp. 247–263.

[5] Xiaoyun Wang, Hongbo Yu, and Yiqun Lisa Yin. "Efficient collision search attacks on SHA-0". In: *Annual International Cryptology Conference*. Springer. 2005, pp. 1–16.

[6] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov. "The first collision for full SHA-1". In: *Annual International Cryptology Conference*. Springer. 2017, pp. 570–596.

[7] Pierre Karpman, Thomas Peyrin, and Marc Stevens. "Practical free-start collision attacks on 76-step SHA-1". In: *Annual Cryptology Conference*. Springer. 2015, pp. 623–642.

[8] Eli Biham et al. "Collisions of SHA-0 and Reduced SHA-1". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2005, pp. 36–57.

[9] Vincent Rijmen and Elisabeth Oswald. "Update on SHA-1". In: *Cryptographers' Track at the RSA Conference*. Springer. 2005, pp. 58–71.

[10] Christophe De Canniere and Christian Rechberger. "Finding SHA-1 characteristics: General results and applications". In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2006, pp. 1–20.

[11] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. "Finding collisions in the full SHA-1". In: *Crypto*. Vol. 3621. Springer. 2005, pp. 17–36.

[12] Marc Stevens. "Counter-cryptanalysis". In: *Advances in Cryptology–CRYPTO 2013*. Springer, 2013, pp. 129–146.

[13] Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. "Merkle-Damgård revisited: How to construct a hash function". In: *Annual International Cryptology Conference*. Springer. 2005, pp. 430–448.

[14] Marc Stevens, Pierre Karpman, and Thomas Peyrin. "Freestart collision for full SHA-1". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2016, pp. 459–483.

[15] Nick Merrill. "Better Not to Know?: The SHA1 Collision & the Limits of Polemic Computation". In: *Proceedings of the 2017 Workshop on Computing Within Limits*. ACM. 2017, pp. 37–42.

[16] Florent Chabaud and Antoine Joux. "Differential collisions in SHA-0". In: *Advances in Cryptology—CRYPTO'98*. Springer. 1998, pp. 56–71.

[17] Xiaoyun Wang and Hongbo Yu. "How to break MD5 and other hash functions". In: *Annual international conference on the theory and applications of cryptographic techniques*. Springer. 2005, pp. 19–35.

[18] Stéphane Manuel and Thomas Peyrin. "Collisions on SHA-0 in one hour". In: *Lecture Notes in Computer Science* 5086 (2008), pp. 16–35.

[19] Marc Stevens. "New collision attacks on SHA-1 based on optimal joint local-collision analysis". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2013, pp. 245–261.

[20] Natarajan Vijayrangan. *Method for preventing and detecting hash collisions of data during the data transmission*. US Patent 8,086,860. Dec. 2011.

[21] Marc Stevens and Daniel Shumow. "Speeding up detection of SHA-1 collision attacks using unavoidable attack conditions." In: *IACR Cryptology ePrint Archive* 2017 (2017), p. 173.

[22] Quynh H Dang. *Secure hash standard*. Tech. rep. National Institute of Standards and Technology, 2015.

[23] Stéphane Manuel. "Classification and generation of disturbance vectors for collision attacks against SHA-1". In: *Designs, Codes and Cryptography* 59.1-3 (2011), pp. 247–263.

[24] *www.shattered.io*. [Online; accessed 8. Jul. 2018]. July 2017. URL: `https://shattered.io`.

4. THE SPONGE STRUCTURE MODULATION APPLICATION TO OVERCOME THE SECURITY BREACHES FOR THE MD5 AND SHA-1 HASH FUNCTIONS

4.1. Introduction

Secure Hash Algorithm (SHA) is the most popular cryptography technique for message authentication and verification. The SHA functions were standardized by the National Institute of Standards and Technology (NIST). SHA standards follow different structure models to construct the compression function. The most popular hash standards follow Merckle-Damgard (MD) and Sponge structure models. Where, MD4, MD5, SHA-1, and SHA-2 standards follow the MD structure, While SHA-3 hash standard follows Sponge structure model.

MD4 developed by Rivest in 1990 [1], then it was replaced by MD5 in 1991 [2]. Both MD4 and MD5 maintain 128-bit hash output with 512-bit block size. For security issues and early signs of collision attack, MD5 was replaced by the SHA-1 in 1993 with 160-bit hash and 512-bit block size [3].

In 2001 SHA-2 was developed and standardized by the NIST as the next version of the secure hash algorithm that follows the same structure model (MD). SHA-2 has six different flavours SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, and SHA-512/256 [4]. Then in 2012, NIST announced the next SHA-3 standard Keccak, which was selected by a competition between 63 competitors through three rounds of selection. Keccak was standardized as the SHA-3 hash standard comprises six flavors, four fixed and two extensible size hashes [5].

Three challenges exist to verify the completeness of any hash standard: preimage, 2^{nd} preimage and collision resistance. Preimage resistance property means to easily obtain the hash from a given message, but difficult to extract it back from a given hash. 2^{nd} preimage resistance

The content of this chapter has been published in the IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC2019). The material in this chapter was co-authored by Zeyad Al-Odat and Samee Khan. Zeyad Al-Odat had primary responsibility for conducting experiments and collecting results. Zeyad Al-Odat was the primary developer of the conclusions that are advanced here. Zeyad Al-Odat also drafted and revised all versions of this chapter. Samee Khan drafted, revised all versions of this chapter, and served as proofreader.

means that it is difficult to find two messages M1 and M2 generate the same Hash. While collision resistance property means the resistant of the probability to generate the same output hash for two messages or more, even though they are different or equal [6].

All secure hash algorithms were tested toward security properties of hash standards, especially collision resistance property. MD5 hash standard was fully exposed to collision attack in 2005 by Wang *et al.* [7]. their work was the first published work that provided a collision example of full MD5. In their work, they used the modular difference technique to construct their attack. More details will be presented in Section 4.3.

The security analysis of the SHA-1 hash standard, against collision attack, was also explored by different publications [8, 9]. Using the concept of modular difference to construct collision path, Wang *et al.* in [8], theoretically, succeeded to find collision attack on full SHA-1. Recently, in 2017, Stevens *et al.* found the first real example of messages that collided when processed using SHA-1 compression function.

However, the secure hash algorithms (MD5 and SHA-1) are still be used by different entities, particularly the SHA-1. Therefore, the efforts of researchers and developers were employed to overcome the collision dilemma which prone systems and applications into a serious security breach.

This chapter analyzes the collision and length extension attacks of the secure hash algorithms, MD5 and SHA-1. The analysis is carried out by testing several examples of collided messages that were generated by the help of ChameleonCloud which is a configurable experimental environment for large-scale cloud research [10]. This chapter presents a versatile modification to the compression functions of MD5 and SHA-1 to counter the collision and length extension attacks. The modification employs the internal round functions of the Keccak hash standard. Yet, Keccak is the most secure hash standard against security breaches. The strength of Keccak comes from the sturdiness of the compression function of Keccak standard [11].

The rest of chapter is organized as follows: Section 2 presents background information related to the subject of the chapter. Section 3 presents the literature review of the previous works. Section 4 shows the proposed work. results and discussions are presented in section 5. Section 6 concludes the chapter.

4.2. Background

Before going through the details of our proposal, a brief description of secure hash algorithm standards and collision attack will be presented. In addition, we collected all notations that are used in this chapter in Table. 4.1.

Table 4.1. Notations of the modified MD5 and SHA-1 structure

Symbols	Meaning
IHV	Initial Hash Value
WS	Working State Variables
$\oplus, \lll$	XOR, Shift left
$[x, y]$	state matrix parameters. Take values 0,1,2,3,4
ROT	Rotate left operation
f	Permutation function
θ	Theta step of the SHA-3 compression function
ρ	Rho step of the SHA-3 compression function
π	Pi step of the SHA-3 compression function
χ	Chi step of the SHA-3 compression function
ι	Iota step of the SHA-3 compression function
b	State matrix size
r	Bit rate which equal to the message block size
c	Capacity, where $b = r + c$.

4.2.1. Brief Description of Secure Hash Algorithms

Secure hash algorithms accept a message of predefined size, then through several steps of compression function calculations, the output hash is produced. Table 4.2 shows different hash standards and their corresponding parameters.

For MD hash standards, the maximum message size that each algorithm accepts is depending on the block size, where the 512-bit block size accepts messages of size less than 2^{64}, while the others accept a message size of 2^{128}-bit. However, the SHA-3 (Keccak) hash standard accepts messages of unlimited (UL) size. The Table shows that SHA-3 supports the same hash lengths that SHA-2 supports beside to two extensible output hashes SHAKE-128 and SHAKE-256. But,

Table 4.2. The MD5, SHA-1, SHA-2, and SHA-3 corresponding parameters

SHA Family				
Algorithm	**Hash Size/bit**	**Block Size/bit**	**Msg Size/bit**	**#Round**
MD5	128	512	$< 2^{64}$	64
SHA-1	160	512	$< 2^{64}$	80
SHA2				
224	224	512	$< 2^{64}$	64
256	256	512	$< 2^{64}$	64
384	384	1024	$< 2^{128}$	80
512	512	1024	$< 2^{128}$	80
512/224	224	1024	$< 2^{128}$	80
512/256	256	1024	$< 2^{128}$	80
SHA3				
224	224	1152	UL	24
256	256	1088	UL	24
384	384	832	UL	24
512	512	576	UL	24
SHAKE-128	Arbitrary	1344	UL	24
SHAKE-256	Arbitrary	1088	UL	24

the block sizes that SHA-3 incorporate are different from the block sizes that SHA-2 incorporated. Moreover, the number of internal rounds of SHA-3 is different from the MD hash standards rounds, this is because of the difference in the nature between the MD and Sponge structures. The Only hash standard that provides a variable hash output is the SHA-3[1].

All hash standards perform pre-processing steps before starting the compression function calculations. These pre-processing steps summarized as follows:

- Message padding. In this phase, the message is padded with a sufficient number of zeros to make the message size divisible by the message's block-size. However, in the case of Sponge structure, the padding process is carried out according to the state size. where the state size is equal 1600-bit for the 24 rounds.

[1]Whenever we mention SHA-3 in this chapter, it implicitly means Keccak.

- Message divide. After the message padding phase, the message is divided into equal size blocks, each of size equal to block size. However, in the case of SHA-3, the message is divided according to the state size (b).

- Compression function calculation. The message's blocks are processed sequentially, one at a time, using the round compression functions according to the used hash. Each block is processed a number of times equal to the number of rounds according to the desired hash function. The output of each block is fed as an input to the second block, after finishing all blocks.

- Output hash generation. After processing all message's blocks, the output hash is taken the concatenation of part or all of the output of the last block calculation round. However, the SHA-3 provides diversity to select the output hash, where the output of each block can be squeezed more time during the same round.

For more details about secure hash algorithms and their compression functions, the reader is referred to [5].

4.2.2. Sponge Structure Model

In Sponge structure model, the data are absorbed in and squeezed out using permutation function. Each block is processed as state matrix and divided into two parts, the bit rate part which denoted by r, and the capacity part which denoted by c. The sum of the two values $r + c = b$, where r is equal to the block size. The state matrix is initialized with zeros then the message blocks (p_i) are processed sequentially as shown in Fig. 4.1. After processing all steps the output hash is taken from any output z_i, according to the desired hash.

4.3. Related Work

Because the MD structure model was prone to security breaches, the researches try to protect hash standards that follow MD structure, particularly MD5 and SHA-1. For instance, collision attack and length extension attack are considered as main security issues that threaten the MD hash standards (MD4, MD5, SHA-1, and SHA-2).

MD5 and SHA-1 hash standards were exposed to collision attack by Wang *et al.* [12, 8]. They presented a novel technique to construct the first collision attack of MD5 and SHA-1. They used a modular difference technique to build a disturbance vector that leads to a path for collision.

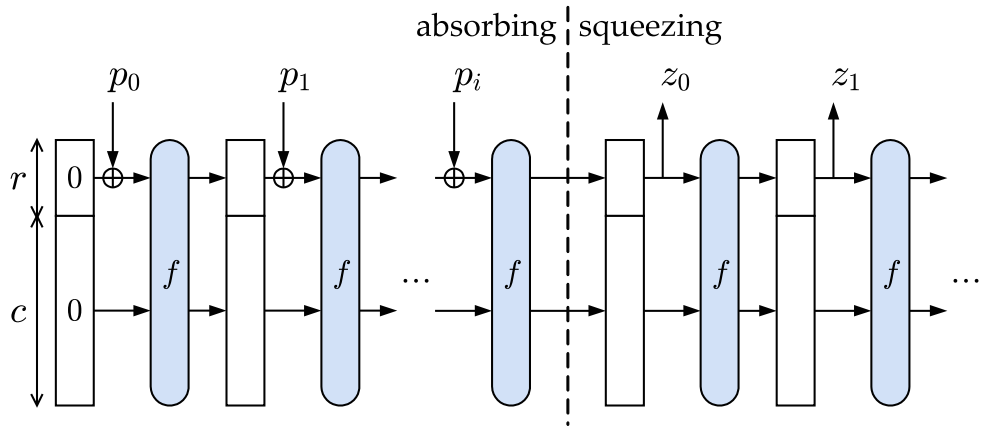

Figure 4.1. Sponge structure

Their work was supported with examples of two blocks messages that have the same hash value. Then in 2017, Stevens *et al.* presented the first real example of collided messages that have the same SHA-1 value [9]. Their work built upon the idea of the differential attack that Wang *et al.* came up with. After the announcement of the first real example of collision attack, many entities dropped SHA-1 from their systems, and they tried to find the best replacement to the SHA-1. However, The SHA-1 and MD5 are still be used by some entities.

To support the entities that still using the weak hash standards, many researchers suggest different replacements and modifications to overcome the collision issue. Two approaches were adopted by the researches. The first approach detects the possibility of a collision before it takes place. While the second approach improves the weak hash standards. The first approach was presented by different publications [13, 14, 6]. Stevens *et al.* developed a novel technique to detect the possibility of a collision attack toward the SHA-1. They got benefited from different disturbance vectors that are used to build a collision attack. The proposed technique was able to detect the occurrence of collision among the used disturbance vectors. The authors used the top 32 vectors that have a high possibility to establish a collision when they are employed [13]. The authors in [14] used the technique of unavoidable bit conditions to speed up the detection of collision attack. They built upon the work of Stevens *et al.*, and they were able to get better speed than the previous technique. Both of the aforesaid techniques build a sibling message from a given message and carry out the calculations for both massages for all possibilities. However Alodat *et al.* in [6] proposed a new technique to detect the collision attack using the idea of two block collision.

Authors claimed that any collision attack built upon two block collision attack, and if we are able to detect the collision of the first two blocks then we will save time and efforts. Then to speed up the detection process the authors constructed their design without any need to build the sibling of a given message.

On another hand, the second approach was adopted by many publications [15, 16, 17, 18]. Instead of detecting the collision attack the researchers of the second approach worked on improving the weak hash functions. Hakim *et al.* in [15] proposed improvement for the MD hash standards. They worked on combining the MD5 and SHA-256 hash functions in one design. The authors changed the expansion equation of the SHA-256 to add more randomness to the proposed scheme. In their work, the structure of MD5 standards was used twice and fed back as an input to the SHA-256 function. Rogel *et al.* proposed a modified version of the SHA-1 by changing adding multiplexer box in the internal calculation of the intermediate hash value [17]. Moreover, Authors in [18] presented a new architecture that used the Sponge structure model to construct a secure hash standard. The proposed design (Titanium) was able to produce a 576-bit hash, which considered as a new hash length that the other hash standards do not have.

The other threat that makes the MD structure model vulnerable is the length extension attacks. This attack exposed many hash standards including MD5, SHA-1, SHA-256, and SHA-512. This kind of attack threatens the Message Authentication Code (MAC) with the hash standards by generating a valid hash value without any need to know the secret code [19]. Some published and online publications proved the inability of the MD structure model against length extension attack [20, 21]. Where the attacker intercepts the sender message and makes the proper modifications without the receiver notice of that modifications even though using the hash function.

In this chapter, we used the Sponge structure model to construct the MD5 and SHA-1 hash functions. Moreover, the internal structure of the Keccak hash function will be employed in our scheme.

4.4. Proposed Methodology

We propose a Sponge structure implementation for the MD5 and SHA-1 hash functions. A pre-processing is accomplished before starting with the Sponge operations (absorb and squeeze). The message is padded first using $10 * 1$ technique, where 1 is added at the end of the message and followed by a sufficient number of zeros then ended with 1. The final message size needs to

be multiple of block size P_i. Afterward, the message is divided into equal size blocks (P_i) each of r bits [5]. Notice that, the padding technique here is different from the technique used in the MD structure.

According to the Sponge structure, the message goes into two phases, the absorb and squeeze phases. In the absorbing phase, the state of b-bit is initialized with 0's, where each state is represented by two values, bit-rate (r) and capacity (c), where, r is the block size P_i, and c complete the state size to 1600-bit. The XOR operations are carried out for the message blocks P_i with r bits of the state. A permutation function f is applied to get next state value [11]. For our proposal, we set $r = 576$ and $c = 1024$. In the squeezing phase, the output hash (z) is taken from the least significant bits of z_0 according to the corresponding hash size (128, 160). Where 128 is the MD5 digest and 160 is the SHA-1 digest.

The permutation function consists of five steps represented by Greek symbols, Theta (θ), Rho (ρ), Pi (π), Chi (χ), and Iota (ι).

Theta **(θ) step.**

This step operates on a 3D-Array ($5 \times 5 \times 64$) as shown in Fig. 4.2. A single 5×5 array is a slice, where the 1D-array, toward z axis, of 64 bits is a lane. Theta step manipulates the state array according to 4.1, 4.2, and 4.3. Where $C[x]$ and $D[x]$ represent lanes and $A[x, y]$ represents slice. Theta computes the parity of each column, then combines them with the XOR operator.

$$C[x] = A[x, 0] \oplus A[x, 1] \oplus A[x, 2] \oplus A[x, 3] \oplus A[x, 4], \tag{4.1}$$

$$D[x] = C[x - 1] \oplus ROT(C[X + 1], 1), \tag{4.2}$$

$$A[x, y] = A[x, y] \oplus D[x], \tag{4.3}$$

where $x = 0, 1, 2, 3, 4$ for all cases.

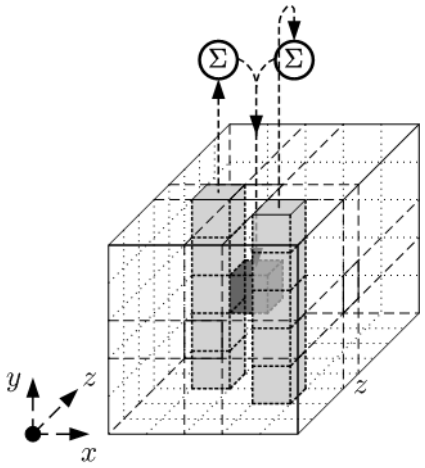

Figure 4.2. Theta step

Rho (ρ) **step**

In this step, one element (lane) of the state matrix $A[x, y]$ is rotated by i-bit, as seen in Eq. 4.4. The rotation offset value denoted by $r[x, y]$ is a constant value assigned according to Table 4.3.

$$step(\rho) = ROT(A[x, y], r[x, y]) \tag{4.4}$$

Table 4.3. Values of constants r[x,y] in the Rho step

	$x = 3$	$x = 4$	$x = 0$	$x = 1$	$x = 2$
$y = 2$	25	39	3	10	13
$y = 1$	55	20	36	44	6
$y = 0$	28	27	0	1	62
$y = 4$	56	14	18	2	61
$y = 3$	21	8	41	54	15

Pi (π) **Step**

Pi step is a complement to *rho* step, where it takes the rotated lanes from *rho* step and put them in different positions in the array matrix ($B[x, y]$), without modifying any value. It only permutes the matrix according to 4.5,

$$B[y, 2x + 3y] = ROT(A[x, y], r[x, y]), \tag{4.5}$$

where $x, y = 0, 1, 2, 3, 4$.

Chi (χ) **Step**

In this step the B matrix, that was generated from the previous step, is manipulated and put the result back in array matrix A according to Eq. 4.6.

$$A[x, y] = B[x, y] \oplus ((\bar{B}[x + 1, y]) \wedge B[x + 2, y]), \tag{4.6}$$

where $x, y = 0, 1, 2, 3, 4$, $\wedge$ bit-wise AND operation, and $\bar{B}[]$ is the bit-wise NEGATION of lane.

Iota (ι) **Step**

Iota step adds the round constant $RC[i]$ to the state matrix A at location $A[0, 0]$ according to 4.7, where each round has a distinct 64-bit round constant.

$$A[0, 0] = A[0, 0] \oplus RC[i], \tag{4.7}$$

where $RC[i]$ is a round constant.

The above permutation steps (θ, ρ, π, χ, and ι) are carried out for 24 rounds. Each round has a distinct round constant $RC[i]$ and their values are assigned according to Table 4.4.

Table 4.4. Iota step round constants

RC[i] starting from RC[0] to RC[23] from left to right top to down.	
$0x0000000000000001$	$0x000000008000808B$
$0x0000000000008082$	$0x800000000000008B$
$0x800000000000808A$	$0x8000000000008089$
$0x8000000080008000$	$0x8000000000008003$
$0x000000000000808B$	$0x8000000000008002$
$0x0000000080000001$	$0x8000000000000080$
$0x8000000080008081$	$0x000000000000800A$
$0x8000000000008009$	$0x800000008000000A$
$0x000000000000008A$	$0x8000000080008081$
$0x0000000000000088$	$0x8000000000008080$
$0x0000000080008009$	$0x0000000080000001$
$0x000000008000000A$	$0x8000000080008008$

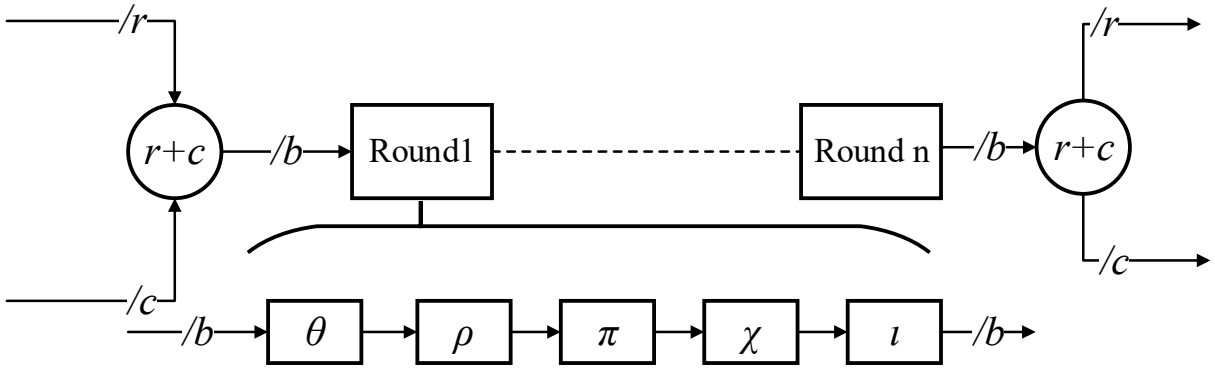

Figure 4.3. Internal round of the compression function.

Fig 4.3 shows the integration between the five permutation steps and the compression components. The input state consists of r and c values which together conform the $b = 1600$ value. The input message's block p_i is processed using the permutation functions for 24 times. Afterward, the output b value is fed as IHV to the next state calculations.

4.5. Results and Discussions

The Sponge structure modulation helps to protect the MD5 and SHA-1 hash standards. Two main security breaches are considered for analysis, collision and length-extension attacks. The experiments were tested using a configurable experimental environment for large-scale cloud research, Chameleon [22]. Where all message samples and testing results were generated on the Chameleon environment. Readers can access all codes, datasets, and other artifacts by referring to [23].

4.5.1. Collision Attack

To test the validity of the proposed scheme, we prepared real examples of collided messages that produce the same hash value. Fig. 4.4 shows two examples of collided messages. Fig. 4.4-a and Fig. 4.4-b are two bank checks that have different information and produce the same SHA-1 value. While, Fig. 4.4-c and Fig. 4.4-d are two different text with the same MD5 hash. Table 4.5 shows the corresponding hash values for each figure. The SHA-1 value for Fig. 4.4-a and Fig. 4.4-b are equal, but when we applied our proposal we were able to produce different hash values. Moreover, the MD5 hash values for Fig. 4.4-c and Fig. 4.4-d are the same, but with our proposal the MD5 values are different.

Table 4.5. Two examples of PDF files that have the same SHA-1 value.

SHA-1		
Fig.4.4-a	SHA-1:	`eda6de91-e04bcf93-342c63ee-c7de45b4-720e4a9e`
	proposed:	`e4667d12-70318267-b8612f69-1100285e-f5fbbd29`
Fig.4.4-b	SHA-1:	`eda6de91-e04bcf93-342c63ee-c7de45b4-720e4a9e`
	proposed:	`a6f02b7a-b36964ae-664d5676-70eb1492-f93fcaa4`
MD5		
Fig.4.4-c	MD5:	`39885429-fe2761b1-0b263b34-8dfdd1b2`
	Proposed:	`75a40711-19b15b99-6eaca068-4d57341b`
Fig.4.4-d	MD5:	`39885429-fe2761b1-0b263b34-8dfdd1b2`
	Proposed:	`288eb391-cb79b77e-c97a0ea9-f3c78274`

4.5.2. Length Extension Attack

Length extension attack not only targeted MD5 and SHA-1 hash functions but all MD structure hashes. One of the main contributions of the proposed scheme is to protect the MD5 and SHA-1 against length extension attack by modulates both functions using Sponge structure.

The attacker uses the padding technique of MD structure models to produce a successful length extension attack. Where the MD-structure padding ends with the message length, the attacker starts from where the padding has ended and continues to add more zeros and the modified information then append the new hash value. Because the receiver deals with payload and hash, he is unable to detect the alteration, because the received information matches the received hash.

Table 4.6 shows example of money transactions between client and server. The hash value has three phases: the client hash phase which represented by MD5_1 and SHA1_1, the after-attack hash phase which represented by MD5_2 and SHA1_2, and the server hash phase which represented by MD5_3 and SHA1_3. The hash value in the client side is computed by hashing the concatenation of `account_from` $\|$ `account_to` $\|$ `amount` values. The attacker intercepts the transaction and performs the length extension attack by appending 5 to the transaction to make the amount equal to 105. The new hash is appended to the modified message and resent to the server. At the server side, the received message is hashed and compared with the new hash value. Then the transaction is approved by the server and the process completed.

(a) (b)

(c)

```
4b48 414e f37e 2764 2643 5c93 fb5c 0341  KHAN.~'d&C\..\.A
c362 cdc0 849c 7796 0753 5858 ebc2 e4ba  .b....w..SXX....
1852 e7e5 42b2 c0f5 84ec 7890 bd58 f540  .R..B.....x..X.@
6095 2f99 9a52 bbe1 a9fa 1c5a d678 eaa0  `./..R.....Z.x..
4c20 f77f d070 dacd 0f42 e71f 3dc1 7232  L ...p...B..=.r2
e618 51fa 2247 2c56 a1ec 2e3f 4a80 c914  ..Q."G,V...?J...
416b 8cf0 a78c bb56 8823 024d d577 59cc  Ak.....V.#.M.wY.
01c4 8d10 f142 1387 c26e c4f4 d673 4c02  .....B...n...sL.
```

(d)

```
4b48 41ce f37e 2764 2643 5c93 fb5c 0341  KHA..~'d&C\..\.A
c362 cdc0 849c 7796 0753 58d8 ebc2 e4ba  .b....w..SX.....
1852 e7e5 42b2 c0f5 84ec 7890 bd58 f540  .R..B.....x..X.@
6095 2f99 9a52 bbe9 a9fa 1c5a d678 eaa0  `./..R.....Z.x..
4c20 f7ff d070 dacd 0f42 e71f 3dc1 7232  L ...p...B..=.r2
e618 51fa 2247 2c56 a1ec 2ebf 4a80 c914  ..Q."G,V....J...
416b 8cf0 a78c bb56 8823 024d d577 59cc  Ak.....V.#.M.wY.
01c4 8d10 f142 137f c26e c4f4 d673 4c02  .....B...n...sL.
```

Figure 4.4. Two examples of collided real examples. (a) and (b) belong to SHA-1 collision. (c) and (d) belong to MD5 collision, where to the left of each text is the corresponding Hexadecimal equivalent.

Table 4.6. Length extension attack bank transaction example

	account_from	account_to	amount	append
Digest	123456	112233	100	5
MD5_1	6036708eba0d11f6ef52ad44e8b74d5b			
MD5_2	8c792805248678cfd72c2a99298748d1			
MD5_3	8c792805248678cfd72c2a99298748d1			
SHA1_1	a65181853e5a1b94b4c7a84b7fc561bdcb9b75e8			
SHA1_2	281385ea5e434561b439dbe4e2417d83fac16833			
SHA1_3	281385ea5e434561b439dbe4e2417d83fac16833			

However, this kind of attack is weak against of Sponge structure model. The message size is not appended to the end of the message in the padding phase, so that, the attacker has no clue about the message size or the amount of extension needed. Our design is strong against length extension attack because we used the Sponge structure. Moreover, different padding technique in the prepossessing phase was carried out.

4.6. Conclusions

In this chapter, a Sponge structure modulation for the MD5 and SHA-1 is presented. The proposed design helps to solve the weaknesses of the MD5 and SHA-1 against security breaches. We investigated our proposal toward collision and length extension attacks. The results showed that the proposed design is resistant to the aforesaid attacks. The strength of our design comes from the strength of the Sponge structure, where the internal permutation function manipulates the data many times with five different steps $(\theta, \rho, \pi, \chi, \iota)$.

4.7. References

[1] Ronald L. Rivest. "The MD4 Message Digest Algorithm". In: *Advances in Cryptology-CRYPTO' 90*. Ed. by Alfred J. Menezes and Scott A. Vanstone. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, pp. 303–311. ISBN: 978-3-540-38424-3.

[2] Ronald Rivest and S Dusse. *The MD5 message-digest algorithm*. Tech. rep. MIT Laboratory for Computer Science, 1992.

[3] PUB FIPS. "180-1. secure hash standard". In: *National Institute of Standards and Technology* 17 (1995), p. 45.

[4] FIPS PUB. "Secure hash standard (shs)". In: *FIPS PUB 180* 4 (2012), pp. 1–27.

[5] NIST DRAFT FIPS PUB and PUB FIPS. "202". In: *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions* (2014).

[6] Zeyad Al-Odat, Mazhar Ali, and Samee U Khan. "Mitigation and Improving SHA-1 Standard Using Collision Detection Approach". In: *2018 International Conference on Frontiers of Information Technology (FIT)*. IEEE. 2018, pp. 333–338.

[7] Xiaoyun Wang and Hongbo Yu. "How to Break MD5 and Other Hash Functions". In: *Advances in Cryptology – EUROCRYPT 2005*. Ed. by Ronald Cramer. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 19–35. ISBN: 978-3-540-32055-5.

[8] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. "Finding collisions in the full SHA-1". In: *Crypto*. Vol. 3621. Springer. 2005, pp. 17–36.

[9] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov. "The first collision for full SHA-1". In: *Annual International Cryptology Conference*. Springer. 2017, pp. 570–596.

[10] *ChameleonCloud*. [Online; accessed Feb. 2019]. Feb. 2019. URL: `https://www.chameleoncloud.org`.

[11] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. "The keccak sha-3 submission". In: *Submission to NIST (Round 3)* 6.7 (2011), p. 16.

[12] Xiaoyun Wang and Hongbo Yu. "How to break MD5 and other hash functions". In: *Annual international conference on the theory and applications of cryptographic techniques*. Springer. 2005, pp. 19–35.

[13] Marc Stevens. "Counter-cryptanalysis". In: *Advances in Cryptology–CRYPTO 2013*. Springer, 2013, pp. 129–146.

[14] Marc Stevens and Daniel Shumow. "Speeding up detection of SHA-1 collision attacks using unavoidable attack conditions." In: *IACR Cryptology ePrint Archive* 2017 (2017), p. 173.

[15] S Hakim and M Fouad. "Improving Data Integrity in Communication Systems by Designing a New Security Hash Algorithm". In: *Journal of Information Sciences and Computing Technologies* 6.2 (2017), pp. 638–647.

[16] Mihir Bellare, Joseph Jaeger, and Julia Len. "Better Than Advertised: Improved Collision-Resistance Guarantees for MD-Based Hash Functions". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2017, pp. 891–906.

[17] Ruji P. Medina Rogel L. Quilala Ariel M.Sison. "Modified SHA-1 Algorithm". In: *Indonesian Journal of Electrical Engineering and Computer Science*. ijeecs.v11. 2018, pp. 1027–1034.

[18] Mohammad A AlAhmad. "Design of a New Cryptographic Hash Function–Titanium". In: *Indonesian Journal of Electrical Engineering and Computer Science* 10.2 (2018), pp. 827–832.

[19] Yu Sasaki and Lei Wang. "Message extension attack against authenticated encryptions: Application to PANDA". In: *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 99.1 (2016), pp. 49–57.

[20] David Nuñez, Isaac Agudo, and Javier Lopez. "Attacks to a proxy-mediated key agreement protocol based on symmetric encryption." In: *IACR Cryptology ePrint Archive* 2016 (2016), p. 1081.

[21] *MD5 Length Extension Attack Revisited | Vũ's Inner Peace*. [accessed 10. Feb. 2019]. Feb. 2019. URL: https://web.archive.org/web/20141029080820/http://vudang.com/2012/03/md5-length-extension-attack.

[22] Kate Keahey et al. "Chameleon: a Scalable Production Testbed for Computer Science Research". In: *Contemporary High Performance Computing: From Petascale toward Exascale*. Ed. by Jeffrey Vetter. 1st ed. Vol. 3. Chapman & Hall/CRC Computational Science. Boca Raton, FL: CRC Press, 2018. Chap. 5.

[23] *zeyadodat/sponge-stucture-modulation*. [accessed 12. Feb. 2019]. Feb. 2019. URL: https://github.com/zeyadodat/sponge-stucture-modulation.

5. A MODIFIED SECURE HASH ALGORITHM ARCHITECTURE TO CIRCUMVENT COLLISION AND LENGTH EXTENSION ATTACKS

5.1. Introduction

Secure Hash Algorithm (SHA) is one of the most used cryptography primitives. The SHA is used for the digital signature authentication. SHA was incorporated into National Institute of Standard and Technology (NIST) since 1991. Thereafter, to give the hash standards a formal form, all SHAs were standardized by the NIST [1, 2, 3]. Hash standards began their construction by following Merckle-Damgard (MD) structure-model. In the MD model, the message is divided into blocks, compressed with compression function, and represented by a fixed size output digest [4]. MD4, MD5, SHA-1, and SHA-2 hash standards follow the MD model[4]. However, each one of them has its compression functions and variables. Due to the collision attack threat, NIST decided to hold a competition to find a successor hash standard. After three stages of the competition, Keccak won the competition and was announced as the new SHA-3 standard [5]. Keccak follows sponge (absorb-squeeze) structure model, where the data are absorbed in and squeezed out. The authors of the Keccak added two variable-length hash flavors, which are not available before, SHAKE-128 and SHAKE-256. Also, Keccak supports fixed-size hashes (224, 256, 384 and 512) as SHA-2 [6].

Different publications proved that MD4, MD5, and SHA-1 hash standards were exposed to collision attack [7, 8, 9, 10, 11]. Wang *et al.* in [8]. proved that SHA-1 is mathematically exposed to a collision attack. Recently, Stevens *et al.* have built upon the arguments of Wang *et al.*, and found the first real example of two PDF files that collide to the same SHA-1 value [12]. Despite that, SHA-1 is still used by different entities and applications (software and hardware). Therefore, the need for an improved and mitigated version of the SHA-1 hash function is crucial, because,

The content of this chapter has been submitted to the IEEE Transactions on Information Forensics and Security. The material in this chapter was co-authored by Zeyad Al-Odat and Samee Khan. Zeyad Al-Odat had primary responsibility for conducting experiments and collecting results. Zeyad Al-Odat was the primary developer of the conclusions that are advanced here. Zeyad Al-Odat also drafted and revised all versions of this chapter. Samee Khan drafted, revised all versions of this chapter, and served as proofreader.

replacing SHA-1 cost money and time, in addition to the hardware limitations to conform with new hashes. So that we present this work to support entities that still using the SHA-1 in their system.

Because the SHA-1 and the SHA-2 standards are following the same construction model (MD), and SHA-2 is strong against collision attack (no collision attack has announced for it). Then modifying the SHA-1 with the principles of the SHA-2 will consolidate the SHA-1 against collision attack either. Furthermore, the consolidation of the SHA-1 using the SHA-2 hash function will strengthen the SHA-1 and preserve the general properties of the SHA-1, e.g., preserve the length of the SHA-1 output to 160-bit. However, the length extension attack threats all MD structure hash standards, including SHA-2. Therefore, a strong replacement that preserves the general structure of the MD hash standards needs to be available.

This chapter presents an improvement for the SHA-1 and SHA-2 hash algorithms. This improvement concentrates on the fusion between the SHA-1 and SHA-2. The main contributions of this work are: outperform the SHA-1 and SHA-2 in term of performance metrics and produce a strong version hash function that is strong against collision and length extension attacks. The rest of chapter is organized as follows: Section 2 gives basic preliminaries related to our proposal. Section 3 presents the literature review of the previous works. Section 4 shows the proposed work. results and discussion in section 5. Section 6 concludes the chapter.

5.2. Preliminaries

Before going through the details of our proposal. The SHA-1, SHA-2, and testing methods need to be addressed first. Brief descriptions about SHA-1, SHA-2, and testing metrics are carried out in the subsequent text.

5.2.1. Brief Description of SHA-1

SHA-1 accepts a message of size less than 2^{64}. Before the message is processed, it needs to be prepared first with padding. This includes adding "1" at the end of the message, then appends the least number of zeros until message size is congruent to 448/512. Then the message size is appended at the end of the message as 64-bits. The input message is divided into equal size blocks, each block size is 512-bit. Five working state variables (A, B, C, D, E) are initialized with fixed Intermediate Hash Values $(IHV0)$. Each block is divided into 16 32-bit words, $(m_0, m_{1,\ldots,m_{15}})$.

Then the message words are expanded using 5.1: for $i = 16$ to 79,

$$m_i = (m_{i-3} \oplus m_{i-8} \oplus m_{i-14} \oplus m_{i-16}) \lll 1, \tag{5.1}$$

where $\oplus$ is the logical XOR operation, and $\lll 1$ is the cyclic shift rotation to the left by 1-bit.

The expanded message words are processed as four groups. Each group consists of 20-steps, round constant k_i, and compression function f, as illustrated in Table 5.1.

Table 5.1. SHA-1 round functions and constants

Round and Steps	**Round Function f(B,C,D)**	**Round Constant (k_i)**
Round1 (0-19)	$(B \wedge C) \vee (\neg B \wedge D)$	`0x5A827999`
Round2 (20-39)	$(B \bigoplus C \bigoplus D)$	`0x6ED9EBA1`
Round3 (40-59)	$(B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$	`0x8F1BBCDC`
Round4 (60-79)	$(B \bigoplus C \bigoplus D)$	`0xCA62C1D6`

Figure 5.2 shows the block diagram of the SHA-1, where the working state variables change after every step as follows:
For $i = 1$ to 80,

$$A_i = RL^5(A_{i-1}) \boxplus F_i(B_{i-1}, C_{i-1}, D_{i-1}) \boxplus E_{i-1} \boxplus W_i \boxplus K_i$$

$$B_i = A_{i-1}$$

$$C_i = RL^{30}(B_{i-1})$$

$$D_i = C_{i-1}$$

$$E_i = D_{i-1},$$

where $A_{80}||B_{80}||C_{80}||D_{80}||E_{80}$, formulate the final hash value.

5.2.2. Brief Description of SHA-2

SHA-2 follows the same construction model as SHA-1. However, the message scheduler and compression functions are different. Furthermore, SHA-2 supports four different flavors (224, 256, 384 and 512) in addition to two truncated versions of SHA-2/512. The SHA-2 employs 8 working

state variables (A, B, C, D, E, F, G, H) each of size of 512-bit for 224 and 256 flavors, and 1024-bits for 384 and 512 flavors. SHA-2 maintains 64 steps for 224 and 256 versions and 80 steps for the others. Each step of the SHA-2 has a distinct constant value k_i. After the message is divided into equal size blocks, the expansion process takes place according to 5.2, 5.3, and 5.4.

$$\sigma_0 = RR^{r1}(W_{t-15}) \oplus RR^{r2}(W_{t-15}) \oplus SR^{r3}(W_{t-15}), \tag{5.2}$$

$$\sigma_1 = RR^{q1}(W_{t-2}) \oplus RR^{q2}(W_{t-2}) \oplus SR^{q3}(W_{t-2}), \tag{5.3}$$

$$W_t = W_{t-16} + \sigma_0 + W_{t-7} + \sigma_1 \qquad 16 \leq t \leq n, \tag{5.4}$$

where $RR^n(X)$ rotates word X to the right by n bits, and $SR^n(X)$ shift right word X by n bits. For SHA-224 and SHA-256 $n = 63$, $r1 = 7$, $r2 = 18$, $r3 = 3$, $q1 = 7$, $q2 = 19$, and $q3 = 10$, while $n = 79$, $r1 = 1$, $r2 = 8$, $r3 = 7$, $q1 = 19$, $q2 = 61$, and $q3 = 6$ for SHA-384 and SHA-512.

The compression function of the SHA-2 is computed according to 5.5, 5.6, 5.7, and 5.8. These equations are applied for n times, according to the desired SHA-2 flavor.

$$Ch(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z), \tag{5.5}$$

$$Maj(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z), \tag{5.6}$$

$$\sum\nolimits_0(x) = RR^{r1}(x) \oplus RR^{r2}(x) \oplus RR^{r3}(x), \tag{5.7}$$

$$\sum\nolimits_1(x) = RR^{q1}(x) \oplus RR^{q2}(x) \oplus RR^{q3}(x), \tag{5.8}$$

where σ_0 and σ_1 are the message expansion equations, $\sum_0$ and $\sum_1$ are the message manipulators that are used inside the compression function. The values $r1 = 2$, $r2 = 13$, $r3 = 22$, $q1 = 6$, $q2 = 11$, and $q3 = 25$ are for SHA-224 and SHA-256, while $r1 = 28$, $r2 = 34$, $r3 = 39$, $q1 = 14$, $q2 = 18$, and $q3 = 41$ are for SHA-384 and SHA-512.

The working state variables (A, B, C, D, E, F, G, H) change after each step accordingly until all steps have completed, as depicted in Figure 5.2. The figure shows the block diagram of the SHA-2 hash standard and the working state variables chain-modification. After processing all blocks,

the final hash is extracted from the concatenation of part or all working state variables, according to the desired hash size [3].

5.2.3. Threat Model

Two main threats are related to our work, the collision attack and length extension attack. The collision attack broke the MD5 and the SHA-1 hash functions, and the researches are going on for the SHA-2 hash function and its corresponding flavors. The length extension attack exposes the MD structures hash functions, e.g., the SHA-1 and SHA-2. This attack is directed to the Hashed Message Authentication Code (HMAC) that is built upon the hash functions.

5.2.3.1. Collision Attack

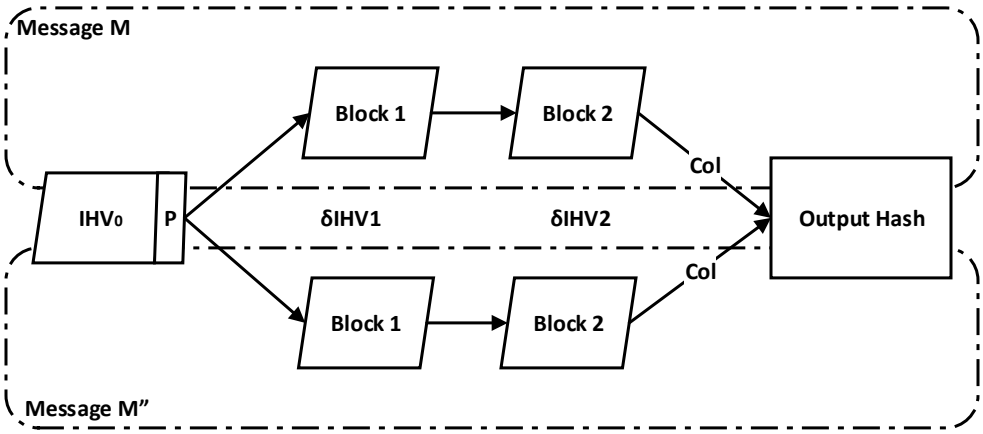

Figure 5.1. Collision attack of two different messages (M and $M^{"}$)

.

Collision resistance is one of the secure hash algorithms properties [13]. If this property is breached, then the corresponding hash function is exposed to a collision attack. This attack is implemented to find two different messages that lead to the same output hash when they are processed using the same hash function [14]. The first successful collision attack is the one that was proposed by Wang et al. [8].

The implemented design was built upon the idea of modular differences of the internal blocks of hash functions. Through these differences, the collision path is constructed in one vector called the disturbance vector [8]. Figure 5.1 shows the construction of a disturbance vector from the modular differences between two messages (M,$M^{"}$) where each one of them consists of two

blocks (Block1, Block2). Both messages started with the same prefix (p) and Initial Hash Value (IHV_0). Then, the differences between the first block and the second block lead to construct the differential path. Once the two messages lead to the same output hash, then the corresponding differential path is considered as a successful disturbance vector.

5.2.3.2. Length Extension Attack

The other threat that makes the MD structure model vulnerable is the length extension attack. This attack exposed many hash standards including MD5, SHA-1, SHA-256, and SHA-512. This kind of attack threatens the Message Authentication Code (MAC) with the hash standards by generating a valid hash value without any need to know the secret code [15]. Some published and online publications proved the inability of the MD structure model against length extension attack [16, 17]. Where the attacker intercepts the sender message and makes the proper modifications without the receiver notice of that modifications.

5.2.4. Metrics

The secure hash algorithms need to pass the performance and accuracy tests to be validated [18]. These tests include the following:

- Test Vectors: which are predefined test samples that were standardized by the NIST to test the validity of hash standards [19].

- Avalanche Effect: Avalanche is an enviable property of the cryptography algorithms. This property measures the strength of the cryptography algorithms against bit change. It imposes that any small change (at least 1-bit) of the plain text produces a radical change to the output hash of the same text before the change. For cryptography algorithms, the output hash of the changed text needs to be at least 50% different from the hash value of the same text before change [20].

- Hamming Distance: which is the measure of the number of bit differences between two different bit strings, of the same size. In our case, Hamming distance is used to measure the number of bit difference between the hash value of one message, and the hash value of the same message with a small bit(s) change [21].

- Bit-Hit: which counts the number of bits, between two hashes, that have the same state (0, 1) and bit-position.
- Test Hashed Message Authentication Code (HMAC).

In our design, we use the aforesaid metrics to test the validity and efficiency of the proposed scheme. In the subsequent text, a literature review of the state of art in this field will be presented.

5.3. Related Work

Because the SHA-1 hash standard showed weakness against collision attack, developers and applications are migrating to the SHA-2 hash standard. However, SHA-1 is still being used by some entities and applications. The researchers have worked to improve the SHA-1 to more strengthened version [22, 23, 24, 13, 25, 26].

The improvement of the SHA-1 focused on the randomness diversity of the hash computation process. This is achieved by modifying the compression function f, as represented by Rogel *et al.* in [24]. They proposed a modified version for the SHA-1 hash function. The proposed modification maintains 192-bit hash, rather than 160-bit of the regular SHA-1. The authors claimed that increasing the size of the SHA-1 hash will strengthen the SHA-1 and add more hash choices. Their design employed a mixer box to mix the working state variables a, b, c, d, e after each step, before forwarding the IHV to the next step. This operation adds more randomness to the generated hash. Furthermore, the bits organization were changed in the modified version. The proposed design was tested using different samples of messages with small bit change. The results showed that the proposed SHA-1 gave good figures in term of avalanches effect better than the SHA-1.

The change of the SHA-1 hash length also inspired Rao *et al.* in [23]. They proposed an advanced version of the SHA-1 with 320-bit hash length. The proposed work changes the SHA-1 block size from 32-bit to 64-bit, which in turn changes the block size to 1024-bit. The extended length SHA-1 resist the collision probability of k randomly generated values by a factor equal to $k/2 * 2^{320}$. The authors succeeded to produce a 320-bit hash value, but they diverge from the concept of the SHA-1 that need to be 160-bits.

Some other works focus on enhancing the MD hash functions by changing the MD concepts [26, 22]. Tiwari *et al.* proposed an enhanced version of SHA-1 hash function by changing the MD paradigm. The proposed work used the dither structure model instead of MD by dividing

the structure into two stages: preprocessing, and computation stages. The preprocessing stage includes message padding, dividing, and IHV initialization. While the computation stage involves the expansion and compression function calculations. An extra dither input is fed to the SHA-1 compression function, which generated using a pseudo-random number generator. The proposed architecture was tested against the birthday attack, differential attack, side attack, and meet in the middle attack. The results showed that the proposed scheme outperforms the SHA-1 in term of security analysis and comparable results in term of speed. However, Alahmad proposed MD model enhancement without restricting to specific standard [22]. The proposed design introduces an enhancement to the hash algorithms using the sponge structure paradigm. The author employed the squeeze and absorb concepts from the SHA-3.

On another hand, other works concentrated on detecting the collision attack before it takes place [27, 28, 13]. To detect the collision attack using the disturbance vectors of the SHA-1 standard, Stevens in [27] proposed a counter collision attack methodology based on the disturbance vectors, that are used to create the collision. These vectors were employed to discover any possibility of a collision before it happens. Then in [28], Stevens *et al.* enhanced the speed of collision detection mechanism using an unavoidable bit condition mechanism. The enhancement involves neglecting the disturbance vectors of less collision probability and consequently speed the detection process. Accordingly Alodat *et al.* in [13] proposed a collision detection approach along with a replacement to the hashes that are weak against collision attack. The proposed design relied on backward expansion equation, and two blocks collision to detect any colliding possibility. The proposed design stated that the messages that are weak against collision are replaced with SHA-512 truncated to 160-bit.

However, in the aforesaid works, there is no focus on the length extension attack. The length extension attack was studied by different researchers using prefix-free computing techniques [29, 30]. But, the prefix-free computing methodology showed weakness against length extension attack, as presented by Ammour *et al.* in [31]. They proposed a method to check the randomness of the MD hash standards using Pseudo Random Oracle (PRO). The proposed design investigates the resistance of MD hash standard against the length extension attack in case of prefix-free computing. They were able to find a counter-example that shows no relationship between the prefix-computing and the length extension attack.

In our work, we propose a consolidated version of the SHA-1 hash function. Our proposal fuses between SHA-1 and SHA-2 standards to produce stronger SHA-1 version against security attacks.

5.4. Proposed Methodology

To maintain the general structures of the original SHA-1 and SHA-2 standards, we used the compression function of the SHA-1 hash function and the round constants of the SHA-2 hash function, as mentioned in Section 2. Eight working state variables A, B, C, D, E, F, G, and H each of size 64-bit are initialized with the IHV values of the SHA-512, as shown in Table 5.2.

Table 5.2. The initial hash values of the proposed design

H_0=6a09e667f3bcc908	H_1=bb67ae8584caa73b
H_2=3c6ef372fe94f82b	H_3=a54ff53a5f1d36f1
H_4=510e527fade682d1	H_5=9b05688c2b3e6c1f
H_6=1f83d9abfb41bd6b	H_7=5be0cd19137e2179

Figure 5.2 shows the block diagram of the proposed scheme, the SHA-1, and the SHA-2. The figure shows how the fusion between the SHA-1 and the SHA-2 is carried out. The Figure 5.2 shows the function block of the proposed scheme and how the working state variables change inside the compression function. The proposed design accepts 32-bit and 64-bit block sizes, where the 32-bit block sizes are processed by padding a 32-bit block with 32 zeros. The proposed design comprises 80 steps, where every 20 steps have a distinct compression function (F_n). This function is adopted from the definition of the SHA-1 hash function, as shown in Table 5.1.

5.4.1. Padding Method

In our design, we employed the 10^*1 technique for padding. The size of the input message must be divisible by the block size. In this technique, a "1" is added to the end of the input message, then least number of zeros are added to make the message size multiple of block size minus 1. Then "1" bit is added at the end, as depicted in Algorithm 2. The number of bits (P), which are needed to make the message size divisible by block (B) size, is determined first. Then, two bits are reserved for the two 1's that will be added at the beginning and end of the padded message. Finally, the remaining bits are filled with 0's and produce the padded message M^*.

Algorithm 2: Padding technique

Input: Message (M)
Block Size (B)
Output: Padded Message (M^*), s.t. $length(M^*)$ is multiple of B

1 $P = M \mod B$
2 $* = P - 2$
3 $M^* = M \| 1 \| 0^* \| 1$
4 **Return** M^*

5.4.2. Fused Compression Function

The function manipulators $\sum_0$ and $\sum_1$ are adopted from the SHA-2 definition, where they add more randomness to the intermediate hash values. Moreover, the message expansion equation of the SHA-2 is employed to expand the working state variables of the proposed scheme.

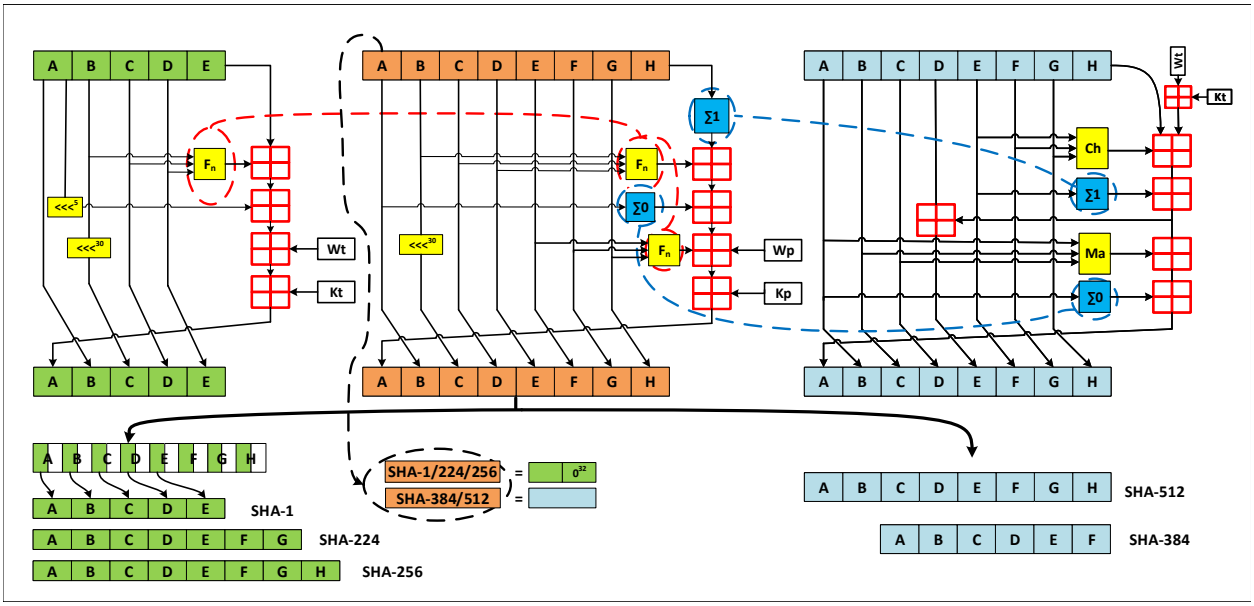

Figure 5.2. The fusion between SHA-1 and SHA-2 hash standards to produce the proposed design

Algorithm 3 shows the pseudo-code of the proposed scheme. The message is padded first then divided into equal size blocks (1024-bit each). Then, the Initial Hash Values (IHVs) are assigned to the working state variables. Afterward, each message block M_i is expanded using the message expansion equation (5.9).

$$W_t = \sigma_0(W_{t-3}) \boxplus Wt - 8 \boxplus \sigma_1(Wt - 14) \boxplus Wt - 16, \tag{5.9}$$

Algorithm 3: PROPOSED SHA

Input: Padded Message
$M = \{M_0, M_1 \ldots, M_n\}$ Blocks
Output: Output Hash

Initialize_IHV :=
$A = H_0^{(i-1)}$, $B = H_1^{(i-1)}$, $C = H_2^{(i-1)}$,
$D = H_3^{(i-1)}$, $E = H_4^{(i-1)}$, $F = H_5^{(i-1)}$,
$G = H_6^{(i-1)}$, $H = H_7^{(i-1)}$
for $t \leftarrow 0$ ***to*** n **do**
  if $t < 16$ **then**
    $W_t \leftarrow M_t$
  else
    $W_t \leftarrow \sigma_0(W_{t-2}) + W_{t-7}) + \sigma_1(W_{t-15}) + W_{t-16}$
for $t \leftarrow 0$ ***to*** 79 **do**
  $H = G$
  $G = F$
  $F = E$
  $E = D$
  $D = C$
  $C = ROL^{32}(B)$
  $B = A$
  $A = \sum_0(A) + F_n(B, C, D) + Kp_t + Wp_t + \sum_1(H) + F_n(E, F, G)$
for $i \leftarrow 0$ ***to*** n **do**
  $H_0^{(i)} = a + H_0^{(i-1)}$
  $H_1^{(i)} = b + H_1^{(i-1)}$
  $H_2^{(i)} = c + H_2^{(i-1)}$
  $H_3^{(i)} = d + H_3^{(i-1)}$
  $H_4^{(i)} = e + H_4^{(i-1)}$
  $H_5^{(i)} = f + H_5^{(i-1)}$
  $H_6^{(i)} = g + H_6^{(i-1)}$
  $H_7^{(i)} = h + H_7^{(i-1)}$
return *Hash*
SHA-1 $\leftarrow H_0^{(N)} \| H_1^{(N)} \| H_2^{(N)} \| H_3^{(N)} \| H_4^{(N)}$
SHA-224 $\leftarrow H_0^{(N)} \| H_1^{(N)} \| H_2^{(N)} \| H_3^{(N)} \| H_4^{(N)} \| H_5^{(N)} \| H_6^{(N)}$
SHA-256 $\leftarrow H_0^{(N)} \| H_1^{(N)} \| H_2^{(N)} \| H_3^{(N)} \| H_4^{(N)} \| H_5^{(N)} \| H_6^{(N)} \| H_7^{(N)}$
SHA-384* $\leftarrow H_0^{(N)} \| H_1^{(N)} \| H_2^{(N)} \| H_3^{(N)} \| H_4^{(N)} \| H_5^{(N)}$
SHA-512* $\leftarrow H_0^{(N)} \| H_1^{(N)} \| H_2^{(N)} \| H_3^{(N)} \| H_4^{(N)} \| H_5^{(N)} \| H_6^{(N)} \| H_7^{(N)}$

where σ_0, σ_1 are Sigma functions adopted from the SHA-2 definition, and $\boxplus$ is the modular addition operation. The σ_0 and σ_1 equations are defined by 5.10 and 5.11.

$$\sigma_0 = ROR(X, 7) \oplus ROR(X, 18) \oplus SHR(X, 3), \tag{5.10}$$

$$\sigma_1 = ROR(X, 17) \oplus ROR(X, 19) \oplus SHR(X, 10). \tag{5.11}$$

The eight working state variables change after each step according to 5.12, 5.13, 5.14, 5.15, and 5.16, for $t = 0$ to 79,

$$\begin{aligned} A = F(B, C, D) + K_t + W_t + \sum\nolimits_0(A) \\ + \sum\nolimits_1(H) + F(E, F, G) \end{aligned}, \tag{5.12}$$

$$B = C, \tag{5.13}$$

$$C = D, \tag{5.14}$$

$$D = E, \tag{5.15}$$

$$E = F, \tag{5.16}$$

$$F = G, \tag{5.17}$$

$$G = H, \tag{5.18}$$

$$H = ROL^{30}(A), \tag{5.19}$$

where $ROL^n(x)$ is the left rotation of word X by n-bit, and $\sum_0$ and $\sum_1$ are the Sum function manipulators and represented by 5.20 and 5.21.

$$\sum\nolimits_0 = ROR(X, 2) \oplus ROR(X, 13) \oplus ROR(X, 22), \tag{5.20}$$

$$\sum\nolimits_1 = ROR(X, 6) \oplus ROR(X, 11) \oplus ROR(X, 25). \tag{5.21}$$

The compression function of the proposed scheme continues until all steps and blocks are processed. The output hash is produced by the concatenating of all or parts of the working state variables (A, B, C, D, E, F, G, H). For the 512-block hash functions (SHA-1, SHA-224, and SHA-

256), the hash output is taken from the concatenation of the least significant 32-bit of working state variables, as seen in Figure 5.2. On the other hand, the SHA-384 and SHA-512 hash functions are produced using the full block size of working state variables.

In the subsequent section, different test criterion will be used to test our proposal. Then we will conclude the chapter.

5.5. Verification of the Proposed Design

The formal verification of the functional correctness of the proposed design is presented in this section. The functional correctness is verified using Coq theorem prover. The Coq is a tool that is used to write a formal proof. The Coq supports interactive proof style, thus it is called interactive proof assistant [32]. The functional specifications of the proposed design are verified using the functional program in Coq. To the best of our knowledge this is the first work that conducts a formal verification for the secure hash algorithms.

5.5.1. Specifications of the Proposed Design

The proposed design defines several functions (F_n, ROL, ROR, $\sum_0$, $\sum_1$, σ_0, σ_1) and the constant values (IHVs, K_p). These functions are transformed into Conjunctive Normal Form (CNF) and fed to Coq for verification.

The transformation of the round function F_n is represented in Listing 1. As mentioned before, we have four functions that are used in the proposed design. Each function operates for 20 steps of the compression function. For all verification codes in this section, z refers to binary-encoded integers and *nat* refers to natural numbers.

```
1 Definition Fn^1−20 (x y z : int) : int := Int.or (Int.and x y)(Int.and
  ↪  (Int.not x) z).
2 Definition Fn^21−40 (x y z : int) : int := Int.xor ((Int.xor y z) x).
3 Definition Fn^41−60 (x y z : int) : int := Int.or (Int.or ((Int.and x
  ↪  y)(Int.and y z)) Int.and y z).
4 Definition Fn^61−80 (x y z : int) : int := Int.xor ((Int.xor y z) x).
```

Listing 1. The transformation of the function F_n into Coq

The function manipulators from equations (5.20), (5.21), (5.10), and (5.11) are transformed and defined to be used in the verification, as shown in Listing 2.

```
Definition ROR b x : int := Int.ror x (Int.repr b).
Definition ROL b x : int := Int.rol x (Int.repr b).
Definition SHR b x : int := Int.shru x (Int.repr b).
Definition Sum_0 (x : int) : int := Int.xor (Int.xor (ROR 2 x) (ROR 13 x))
  ↪  (ROR 22 x).
Definition Sum_1 (x : int) : int := Int.xor (Int.xor (ROR 6 x) (ROR 11 x))
  ↪  (ROR 25 x).
Definition Sig_0 (x : int) : int := Int.xor (Int.xor (ROR 7 x) (ROR 18 x))
  ↪  (SHR 3 x).
Definition Sig_1 (x : int) : int := Int.xor (Int.xor (ROR 17 x) (ROR 19 x))
  ↪  (SHR 10 x).
```

Listing 2. Transformation of the function manipulators into Coq

The constant values of (K_p) and the Initial Hash Values (IHVs) are processed as a vector of a series of 64-bit hexadecimal values. These values are written in Coq in the decimal form and injected to the program using the `Int.repr` command. Listing 3 shows the code written in Coq for the K_p constant values and the IHVs of the proposed design. They are mapped into two series of vectors and defined to be used in the verification process.

```
Definition K_p := map Int.repr [4794697086780616226, 6480981068601479193,
  ↪  ..., 2966304930631591287643 5].
Definition IHV := Map Int.repr [7640891576956012808, 13503953896175478587,
  ↪  ..., 6620516959819538809].
```

Listing 3. The constant values in Coq

From the definition of the proposed design, the input message (M) needs to be padded before processing. The padding specification is written in Coq according to Algorithm 2. Listing 4 shows the padding process in Coq. The number 2 comes from 1+1: 1 terminator byte (128) and 1 ending byte (1). These bytes are used to put the padding boarders. Then, the value $(-(n+2)$ mod 64) gives the number of required zero bytes between the boarders that makes the message M multiple of block size. After padding, the message expansion is applied using Equation (5.9). It is translated into Coq as depicted in Listing 5.

```
Definition padding M := let n := Zlength M in Zlist_to_intlist (M++[128%Z]
  ↪  ++ list-repeat(Z.nat(-(n+2) mod 64)) 0) ++[1%Z].
```

Listing 4. The message padding in Coq

```
Function Wp (M: Z-> int) (t: Z){measure Z.nat t}: int :=
if zlt t 16 then M t
else (Int.add (Int.add (sig_1 (Wp M (t-2))) (Wp M (t-7)))(Int.add (sig_0 (Wp
  ↪  M (t-15))) (Wp M (t-16)))).
```

Listing 5. The message expansion equation in Coq

The proofs of the working state variables are conducted in four recursive calls for the message expansion equation, as shown in Listing 6. The `omega` function is a tactic in Coq for solving goals in Presburger arithmetic, and the `intros` tactic is used to change the variables into hypothesis in Coq.

```
Proof.
intros;
apply Z2Nat.inj_lt;omega.(* t-2<t *)
intros;
apply Z2Nat.inj_lt;omega.(* t-7<t *)
intros;
apply Z2Nat.inj_lt;omega.(* t-15<t *)
intros;
apply Z2Nat.inj_lt;omega.(* t-16<t *)
Qed.
```

Listing 6. The message expansion equation in Coq

For each round of the internal compression function, Listing (7) is applied to produce the intermediate hash value after each step. The function returns the n^{th} element of the Kp list using the `nthi` function call. For each function call of the internal rounds, the `proof` function is applied to check the validity after each round. The `Qed` command is used to extract the proof terms from the proof scripts.

```
Definition registers := list int
Function Round (Wreg: registers) (M: Z→int)(t: Z.nat(t+1)) t} {measure(fun
  ↪  t⇒)} : registers :=
if zlt t 0 then Wreg
else match Round Wreg M (t-1) with [a,b,c,d,e,f,g,h] ⇒
let T1 := Int.add(Int.add(Int.add(Int.add h (Sum_1 h))(Fn e f g))(nthi Kp
  ↪  t))(Wp M t) in
let T2 := Int.add (Sum_0 a) (Fn b c d) in [Int.add T1 T2, a, b, c, Int.add d
  ↪  T1, e, f, g]
⇒ nil
end.
Proof.
intros; apply Z2Nat.inj-lt; omega.
Qed.
```

Listing 7. The message expansion equation in Coq

5.5.2. Functional Specification

The functional specifications are proved after the program satisfies the list of specifications that were listed previously. The expected behaviour of the program represents the functional specifications that the program intends to produce. Therefore, we wrote the specifications in an executable form and run it using Coq Integrated Development Environment (Coqide 8.9.1). This property allows to run and validate the list of specifications and functionalities at the same time, which represents a powerful property of Coq.

5.6. Results and Discussions

To test the validity of our scheme, we used four different ways. The first one, apply the official NIST test vectors. Secondly, the Avalanche effect, which measures the effect of bit change on the produced hash. Thirdly, The hamming distance, which measures the number of bit differences between two messages, with a predefined bit difference between them. Lastly, the bit-hit property, which measures the count of bits that have the same values at the same position between two hashes.

The experiments were tested using a configurable experimental environment for large-scale cloud research, Chameleon [33]. Where all message samples and testing results were generated on the Chameleon environment. Readers can access all codes, datasets, and other artifacts used to produce results from https://github.com/zeyadodat/collision-files.

5.6.1. Test Vectors

The test vectors, that were approved by the NIST, are used to test the validity of the proposed scheme. Table 5.3 shows a comparison between the hash values produced by the SHA-1 and the proposed scheme. In our proposal, we preserved the hash length of the SHA-1 and got different hash values. The table contains six testing vectors, these vectors differ in length to cover different test cases. Through all cases, our proposal was able to generate a unique hash value. To save space, we present the hash results of the SHA-1 hash function and our replacement.

Table 5.3. The official NIST test vectors for secure hash algorithms

case1: "abc"	
Proposed SHA-1:	f17447eb 851879a7 23a0185c a691062f 5092e404
SHA-1:	a9993e36 4706816a ba3e2571 7850c26c 9cd0d89d
case2: " "	
Proposed SHA-1:	63a7110d f5ce8be1 7d1662c8 6d7e95b1 7ecf15e6
SHA-1:	da39a3ee 5e6b4b0d 3255bfef 95601890 afd80709
case3: "abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq"	
Proposed SHA-1:	787dcd87 d9fbc0ed 71de8cf6 39b780a3 2b16b546
SHA-1:	84983e44 1c3bd26e baae4aa1 f95129e5 e54670f1
case4: "abcdefghbcdefghicdefghijdefghijkefghijklfghijklmghijklmnhijklmnoijklmnopjklmnopqklmnopqrlmnopqrsmnopqrstnopqrstu"	
Proposed SHA-1:	3882860f b6d50182 3457e4e6 9d78ec2d 81a8524d
SHA-1:	a49b2446 a02c645b f419f995 b6709125 3a04a259
case5: one million (1,000,000) repetitions of the character "a"	
Proposed SHA-1:	fc2006d2 e9f5cf5a dae2cf05 4a1b2c07 93b02edd
SHA-1:	34aa973c d4c4daa4 f61eeb2b dbad2731 6534016f
case6: "abcdefghbcdefghicdefghijdefghijkefghijklfghijklmghijklmnhijklmno" repeated 16,777,216 times	
Proposed SHA-1:	5a9458d1 b0e2fa7e 21b8d5b3 df562d13 06e8c1f8
SHA-1:	7789f0c9 ef7bfc40 d9331114 3dfbe69e 2017f592

5.6.2. Avalanche Effect

For the cryptography algorithms, the avalanche effect property holds if the output hash of a message is 50% different from the hash value of the same message with 1-bit change. Two approaches were used to test the avalanche effect. The first one compares the proposed scheme with existing works using the same set of messages. The second approach relies on generating random messages with random lengths.

5.6.2.1. Predefined Messages Set

In the first test, we used predefined messages and compared them with the SHA-1, SHA-256, SHA-512 and the latest existing work of modifying the SHA-1. Table 5.4 shows the avalanche test results compared to our proposal.

Table 5.4. Comparison of avalanche test

Test Sample	Avalanche (%)				
	SHA-1	SHA-256	SHA-512	Work[24]	Proposed
Test1	46.25	49.03	51.2	56.77	57.5
Test2	50.09	51.02	50.8	48.37	56.875
Test3	50.00	49.35	49.75	38.75	50.2
Test4	51.13	52.1	52.5	49.76	55

The algorithms were tested on the same message samples. Where "Test1" examines two messages with 1-bit change, "Test2" examines two messages with difference in only few bits, "Test3" examines two words with values: "abc123_owlstead_1255" and "abc123_owlstead_59131", and "Test4" examines two messages with different length where the first message is "b b", and the second message is "b b b". For the fourth test cases, our proposal outperforms the others in term of avalanche effect. Which supports the validity and distinction of the proposed scheme.

5.6.2.2. Random Messages Set

For proof of concept, we extended the test to include different cases with different messages' lengths. Ten categories were used, where each category comprises 1000 randomly generated messages with random lengths. The first category contains 1-bit change messages, where each message is generated randomly and regenerated with 1-bit change. The hash value is computed for both generated and regenerated messages of each category. Then we compute the avalanche effect between the hashes of the generated and regenerated versions of each message. The same procedure is carried out for all categories. Where, the categories are (2, 3,..., 10)-bit changes. After processing all categories we computed the average avalanche effects as shown in Figure 5.3.

The $x-$axis represents the number of samples while the $y-$axis is the average avalanche effects of all samples. The figure shows that our proposal gives an avalanche effect above 50%

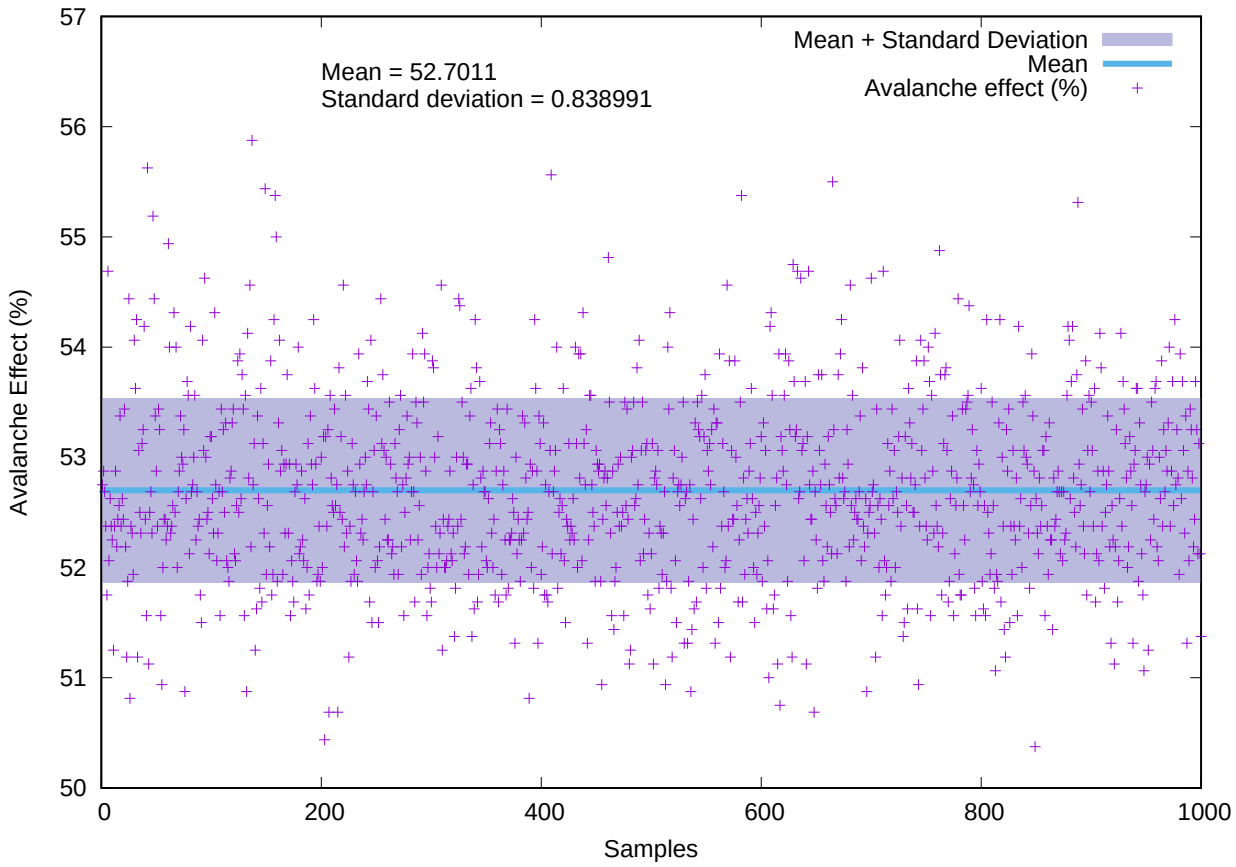

Figure 5.3. Average of the avalanche effect of all testing samples for the ten categories.

for all samples. The avalanche effect has a Mean value above 52% for all 1-Million samples of all categories. Mean value proves the validity of our proposal. Where the avalanche effect needs to be above 50% for the cryptography algorithms.

5.6.3. Hamming Distance

To test the hamming distance, we take the same ten categories and samples that were used to test the avalanche effect. The Average Hamming distance between the hash values of the generated and regenerated messages of the same category can be depicted in Figure 5.4. The figure shows the Mean value of hamming distances for all samples of all categories.

The average Hamming distance value is above 84, which is greater than half of the total bits of the produced hash. These values (avalanche effect and hamming distance) prove that our design is a good choice to replace the SHA-1 and SHA-2 functions.

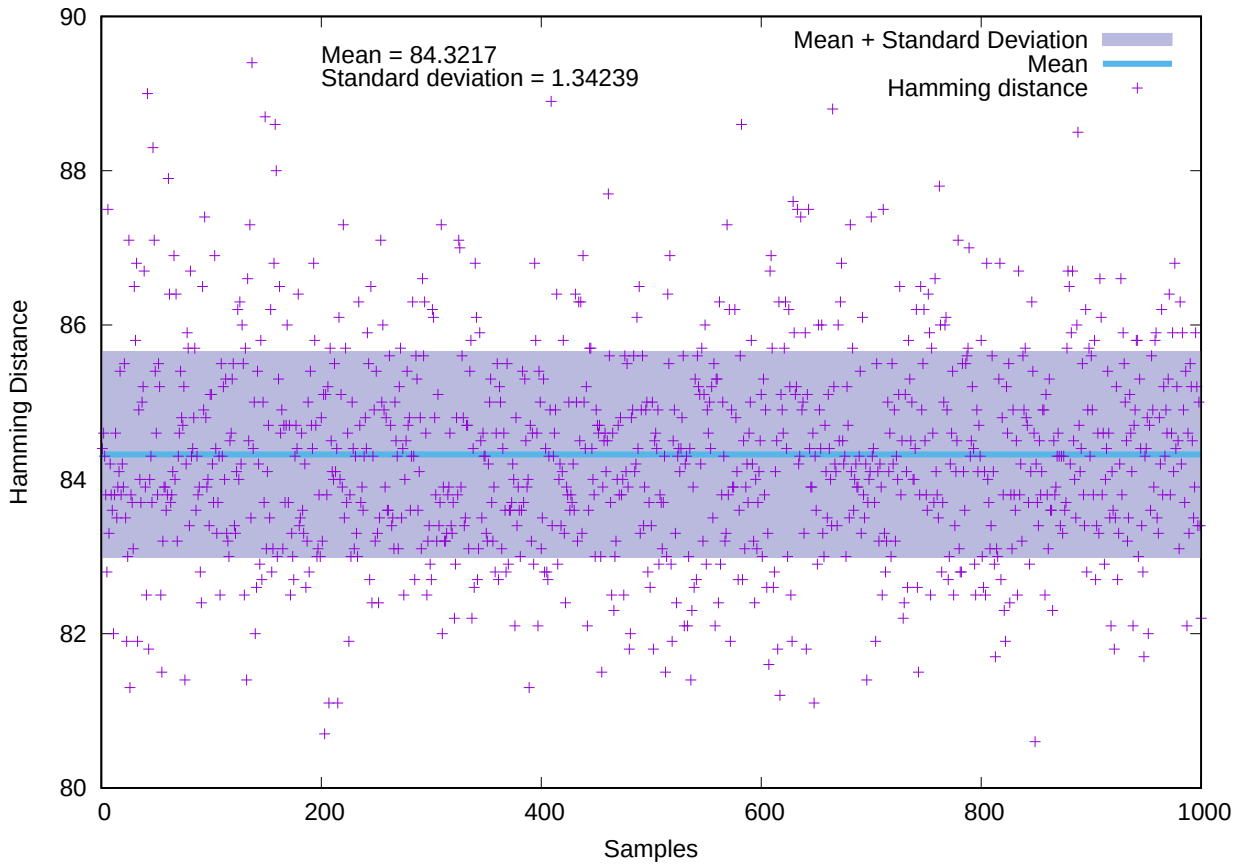

Figure 5.4. Average of the hamming distance of all testing samples for the ten categories.

5.6.4. Bit-Hit

To further test our design, we check the bit-hit property, which counts the total number of bits at the same position that have the same bit state. The same categories and samples that were used for avalanche effect and hamming distance were used. Table 5.5 shows the average number of bits that are at the same position between generated and regenerated message when applying our design on SHA-1, SHA-256, and SHA-512. All values give a constant average for the three algorithms SHA-1, SHA-256, and SHA-512 with approximately 2.5, 4, and 7.5 bit hit averages, respectively. This property reflects the ability of our design to resist the possibility of a collision attack at the bit level.

5.6.5. Counter-Collision Attack

The collision attack is one of the main security issues that threaten the secure hash algorithms. Wherefore, the new hash standards or the modified versions of existing SHAs need

Table 5.5. Average number of bit change

Bit change Category	# of samples	Average of different bits		
		SHA-1	SHA-256	SHA-512
1-bit change	1000	2.514	3.9	6.5
2-bit change	1000	2.444	4.1	6.85
3-bit change	1000	2.455	3.75	7.25
4-bit change	1000	2.513	4.13	6.95
5-bit change	1000	2.521	4.025	6.75
6-bit change	1000	2.463	3.725	7.33
7-bit change	1000	2.542	4.25	7.5
8-bit change	1000	2.426	4.125	7.65
9-bit change	1000	2.543	3.125	8.125
10-bit change	1000	2.548	3.55	7.513

to circumvent the collision attack. We test the collision resistance on the SHA-1 hash standard because it is the only hash standard (among our scope) with real colliding examples.

To test the efficiency of our proposal against collision attack, we used real examples of PDF files. The examples contain four files file1, file2, file3, and file4. The SHA-1 values of file1 and file2 are equal, as well as the SHA-1 values of file3 and file4. Table 5.6 shows the computed hash values of the four PDF files. Our proposal produces a unique hash value for each file. However, when using the SHA-1 to compute the hash, the resultant hashes of file1 and file2 are the same. The PDF files that were used in the experiments can be found in [34].

The above-mentioned tests were used to examine our proposal. The test vector was used to validate our proposal. The Avalanche effect, hamming distance, and Bit-Hit were used to test the efficiency and strength of our proposal. While real collided PDF files were tested on our scheme, and a unique hash value for each file was produced.

5.6.6. Length Extension Attack

Length extension attack not only targeted the MD structure hashes. One of the main contributions of the proposed scheme is to protect the MD hash functions against length extension attack by modulates the internal compression functions and using different padding mechanism, which hides the actual message size from the attacker.

Table 5.6. Two examples of PDF files that have the same SHA-1 value.

Example1		
file1	SHA-1:	38762cf7-f55934b3-4d179ae6-a4c80cad-ccbb7f0a
	Proposed:	f757b8cf-8eece51a-809b902d-d2d7909f-14b413ff
file2	SHA-1:	38762cf7-f55934b3-4d179ae6-a4c80cad-ccbb7f0a
	Poposed:	044be3f9-1f97096a-1c11dd05-89a9f0e9-d59bf674
Example 2		
file3	SHA-1:	d00bbe65-d80f6d53-d5c15da7-c6b4f0a6-55c5a86a
	Proposed:	26d52b64-706ff75c-cb0a2365-de8cfd8d-8c759dd6
file4	SHA-1:	d00bbe65-d80f6d53-d5c15da7-c6b4f0a6-55c5a86a
	Proposed:	94f3ec17-ccdf9650-ad95d196-c4ceafc4-93bf17de

The attacker uses the padding technique of MD structure models to produce a successful length extension attack. Where the MD-structure padding ends with the message length, the attacker starts from where the padding has ended and continues to add more zeros and the modified information then appends the new hash value. Because the receiver deals with a payload and hash, there will be an inability to detect the alteration because the received information matches the received hash.

Table 5.7 shows an example of money transactions between client and server. The hash value has three phases: the client hash phase which represented by SHA1_1, the after-attack hash phase which represented by SHA1_2, and the server hash phase which represented by SHA1_3. The hash value in the client side is computed by hashing the concatenation of `account_from` ‖ `account_to` ‖ `amount` values. The attacker intercepts the transaction and performs the length extension attack by appending 5 to the transaction to make the amount equal to 105. The new hash is appended to the modified message and resent to the server. At the server-side, the received message is hashed and compared with the new hash value. Then the transaction is approved by the server and the process completed.

In our proposal, the message size is not appended to the end of the message in the padding phase, so that, the attacker has no clue about the message size or the amount of extension needed.

Table 5.7. Length extension attack bank transaction example

	account_from	**account_to**	**amount**	**append**
Digest	123456	112233	100	5
SHA1_1	a65181853e5a1b94b4c7a84b7fc561bdcb9b75e8			
SHA1_2	281385ea5e434561b439dbe4e2417d83fac16833			
SHA1_3	281385ea5e434561b439dbe4e2417d83fac16833			

Moreover, the padding technique that we employed is different from the padding methods of the SHA-1 and SHA-2 hash functions.

5.7. Conclusions

In this chapter, an improved version of the SHA-1 and SHA-2 hash function is presented. The proposed design consolidates the SHA-1 and SHA-2 against collision and length extension attacks. The proposed design preserves the general properties of the SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512 hash functions., e.g., the hash length. The design employs the function manipulators of the SHA-1 and SHA-2 hash standards, which add more randomness to the proposed design and strengthen it against collision and length extension attacks. The testing results proved the efficiency of our proposal through different factors, which are avalanche effect, testing vectors, hamming distance, and bit-hit. Moreover, our proposal is effective against collision and length extension attacks as shown in the provided examples.

The scope still open for the future researches that aim to consolidate the hash standards against security breaches of the cryptographic hash functions. The security analyses of the SHA-3 hash function will be conducted and compared with the performance of other designs.

5.8. References

[1] Ronald L. Rivest. "The MD4 Message Digest Algorithm". In: *Advances in Cryptology-CRYPTO' 90*. Ed. by Alfred J. Menezes and Scott A. Vanstone. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, pp. 303–311. ISBN: 978-3-540-38424-3.

[2] Ronald Rivest and S Dusse. *The MD5 message-digest algorithm*. Tech. rep. MIT Laboratory for Computer Science, 1992.

[3] FIPS PUB. "Secure hash standard (shs)". In: *FIPS PUB 180* 4 (2012), pp. 1–27.

[4] Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. "Merkle-Damgård revisited: How to construct a hash function". In: *Annual International Cryptology Conference*. Springer. 2005, pp. 430–448.

[5] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. "The keccak sha-3 submission". In: *Submission to NIST (Round 3)* 6.7 (2011), p. 16.

[6] NIST DRAFT FIPS PUB and PUB FIPS. "202". In: *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions* (2014).

[7] Xiaoyun Wang and Hongbo Yu. "How to break MD5 and other hash functions". In: *Annual international conference on the theory and applications of cryptographic techniques*. Springer. 2005, pp. 19–35.

[8] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. "Finding collisions in the full SHA-1". In: *Crypto*. Vol. 3621. Springer. 2005, pp. 17–36.

[9] Christophe De Canniere and Christian Rechberger. "Finding SHA-1 characteristics: General results and applications". In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2006, pp. 1–20.

[10] Stevens Marc. "Fast Collision Attack on MD5." In: *IACR Cryptology ePrint Archive* 2006 (2006), p. 104.

[11] Marc Stevens. "New collision attacks on SHA-1 based on optimal joint local-collision analysis". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2013, pp. 245–261.

[12] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov. "The first collision for full SHA-1". In: *Annual International Cryptology Conference*. Springer. 2017, pp. 570–596.

[13] Zeyad Al-Odat, Mazhar Ali, and Samee U Khan. "Mitigation and Improving SHA-1 Standard Using Collision Detection Approach". In: *2018 International Conference on Frontiers of Information Technology (FIT)*. IEEE. 2018, pp. 333–338.

[14] Zulfany Erlisa Rasjid, Benfano Soewito, Gunawan Witjaksono, and Edi Abdurachman. "A review of collisions in cryptographic hash function used in digital forensic tools". In: *Procedia Computer Science* 116 (2017). Discovery and innovation of computer science technology in artificial intelligence era: The 2nd International Conference on Computer Science and Computational Intelligence (ICCSCI 2017), pp. 381–392. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2017.10.072. URL: http://www.sciencedirect.com/science/article/pii/S1877050917321221.

[15] Yu Sasaki and Lei Wang. "Message extension attack against authenticated encryptions: Application to PANDA". In: *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 99.1 (2016), pp. 49–57.

[16] David Nuñez, Isaac Agudo, and Javier Lopez. "Attacks to a proxy-mediated key agreement protocol based on symmetric encryption." In: *IACR Cryptology ePrint Archive* 2016 (2016), p. 1081.

[17] *MD5 Length Extension Attack Revisited | Vũ's Inner Peace*. [accessed 10. Feb. 2019]. Feb. 2019. URL: https://web.archive.org/web/20141029080820/http://vudang.com/2012/03/md5-length-extension-attack.

[18] Saif Al-Kuwari, James H Davenport, and Russell J Bradford. "Cryptographic hash functions: recent design trends and security notions". In: *Cryptology ePrint Archive: Report 2011/565* (2010).

[19] DI Management Services Pty Limited David Ireland. *Test vectors for SHA-1, SHA-2 and SHA-3*. [accessed 15. Nov. 2018]. July 2018. URL: https://www.di-mgt.com.au/sha%5C_testvectors.html.

[20] Parvez khan Pathan and Basant Verma. "Hyper Secure Cryptographic Algorithm to Improve Avalanche Effect for Data Security". In: *International Journal of Computer Technology and Electronics Engineering (IJCTEE)* 1.2 (2015).

[21] Mohammad Norouzi, David J Fleet, and Ruslan R Salakhutdinov. "Hamming distance metric learning". In: *Advances in neural information processing systems*. 2012, pp. 1061–1069.

[22] Mohammad A AlAhmad. "Design of a New Cryptographic Hash Function–Titanium". In: *Indonesian Journal of Electrical Engineering and Computer Science* 10.2 (2018), pp. 827–832.

[23] Siddhartha Rao. "Advanced SHA-1 Algorithm Ensuring Stronger Data Integrity". In: *International Journal of Computer Applications* 130.8 (2015), pp. 25–27.

[24] Ruji P. Medina Rogel L. Quilala Ariel M.Sison. "Modified SHA-1 Algorithm". In: *Indonesian Journal of Electrical Engineering and Computer Science*. ijeecs.v11. 2018, pp. 1027–1034.

[25] Raaed K Ibraheem, Roula A J Kadhim, and Ali S H Alkhalid. "Anti-collision enhancement of a SHA-1 digest using AES encryption by LABVIEW". In: *Information Technology and Computer Applications Congress (WCITCA), 2015 World Congress on*. IEEE. 2015, pp. 1–6.

[26] Harshvardhan Tiwari and Krishna Asawa. "Enhancing the Security Level of SHA-1 by Replacing the MD Paradigm". In: *Journal of computing and information technology* 21.4 (2014), pp. 223–233.

[27] Marc Stevens. "Counter-cryptanalysis". In: *Advances in Cryptology–CRYPTO 2013*. Springer, 2013, pp. 129–146.

[28] Marc Stevens and Daniel Shumow. "Speeding up detection of SHA-1 collision attacks using unavoidable attack conditions." In: *IACR Cryptology ePrint Archive* 2017 (2017), p. 173.

[29] Nasour Bagheri, Praveen Gauravaram, Lars R Knudsen, and Erik Zenner. "The suffix-free-prefix-free hash function construction and its indifferentiability security analysis". In: *International Journal of Information Security* 11.6 (2012), pp. 419–434.

[30] Shoichi Hirose, Je Hong Park, and Aaram Yun. "A simple variant of the Merkle-Damgård scheme with a permutation". In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2007, pp. 113–129.

[31] Kamel Ammour, Lei Wang, and Dawu Gu. "Pseudo random oracle of Merkle-Damgård hash functions revisited". In: *Science China Information Sciences* 62.3 (2019), p. 32112.

[32] Yves Bertot and Pierre Castéran. *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions*. Springer Science & Business Media, 2013.

[33] Kate Keahey et al. “Chameleon: a Scalable Production Testbed for Computer Science Research”. In: *Contemporary High Performance Computing: From Petascale toward Exascale*. Ed. by Jeffrey Vetter. 1st ed. Vol. 3. Chapman & Hall/CRC Computational Science. Boca Raton, FL: CRC Press, 2018. Chap. 5.

[34] *zeyadodat/collison-files*. [accessed 16. Jan. 2019]. Jan. 2019. URL: `https://github.com/zeyadodat/collision-files`.

6. RANDOMNESS ANALYSES OF THE SECURE HASH ALGORITHMS, SHA-1, SHA-2 AND MODIFIED SHA

6.1. Introduction

A powerful tool is needed for message authentication and verification. Secure Hash Algorithm (SHA) is the most popular tool to ensure integrity properties. SHA was developed and standardized by the National Institute of Standards and Technology (NIST) [1]. SHA was designed for compression purpose, where a message of any size is compressed to a fixed-length hash. To consider any cryptography hash function as a secure one, three challenges exist preimage, 2^{nd}preimage, and collision resistance. The preimage means that the SHA function is a one-way function. The 2^{nd}preimage means that there are no messages that have the same hash value. The collision resistance means that there is no possibility to generate the same hash from two or more different messages [2].

The primary hash algorithms follow two types of structures, Merkle-Damgård and Sponge structures. The hash functions MD5, SHA-1, and SHA-2 follow the Merkle-Damgård , while the official SHA-3 (Keccak) follows the Sponge structure. The MD5, and SHA-1 produce 128-bit, and 160-bit output hash, respectively. However, these three hash functions were exposed to collision attack by Wang *et al.* since 2005 [3]. Since then, the SHA-2 emerged to be the official hash standard with four different flavors (SHA-224, SHA-256, SHA-384, SHA-512), and extra truncated versions (SHA-512/224, and SHA-512/256). Even with the existence of the SHA-2, some entities still using the SHA-1 and MD5 hash functions [4]. Lately, the NIST released the latest hash standard SHA-3 (Keccak) on 2015 after a competition that was held to select the winner among 64 candidates [5].

The designers of the hash functions try to generate a unique output hash for any input and make it random when compared to another output. To test the randomness property, the Pseudo-Random Number Generator (PRNG) is used as a statistical test. the NIST uses the statistical

The content of this chapter has been published in the IEEE 17th International Conference on Frontiers of Information Technology (FIT2019). The material in this chapter was co-authored by Zeyad Al-Odat, Assad Abbas and Samee Khan. Zeyad Al-Odat had primary responsibility for conducting experiments and collecting results. Zeyad Al-Odat was the primary developer of the conclusions that are advanced here. Zeyad Al-Odat also drafted and revised all versions of this chapter. Assad Abbas drafted and revised all versions of this chapter. Samee U. Khan revised the material and served as proofreader.

tests to verify the cryptography applications [6, 7]. However, these tests are general and they are only used to test the randomness of binary sequences from any source. Moreover, the SHAs are designed to have fixed input parameters, which are nonrandom, and fixed output length. Therefore, the NIST test suites may not give a fair result for the randomness analysis.

In this chapter, randomness analyses are conducted to examine the secure hash algorithms (SAH-1 and SHA-2). Moreover, a modified design is proposed to improve the randomness of the internal rounds of the hash functions. The analyses are based on the *Bayes factor* and *odd ratio* tests. We utilized a large-scale experimental environment testbed and Compute Unified Device Architecture (CUDA) platform to test the design and compare it with existed hash standards.

The rest of chapter is organized as follows: A background information about the SHA-1, SHA-2, and *Bayesian odd ratio test* will be presented in Section 2; Section 3 presents the state of art in the related field; Section 4 elaborates the randomness test and all related analyses; Sections 5 presents the discussions and analyses; Conclusion will be presented in Section 6.

6.2. Preliminaries

To better understand the proposed randomness analyses scheme, brief descriptions about SHA-1 and collision attack needs to be addressed. The threat model is also included to describe the possible security issues that are related to our work.

6.2.1. SHA-1 Standard

SHA-1 was published by the NIST in 1995 after undisclosed security concerns with the SHA-0 [8]. The SHA-1 maintains 160-bit hash output through 80 steps of compression function evaluations. The SHA-1 is considered as a part of the Merkle-Damgård functions, where the input message is divided into number of blocks and processed sequentially.

The SHA-1 takes a message of a predefined size and process it using the SHA-1 round function F. This includes the preprocessing phase where the message is padded to make its size divisible by 512 (block size). The message is padded by adding "1" at the end of the message M, least number of 0's, and the message size as 64-bit integer, as shown in Figure6.1.

After padding, the message is divided into equal size blocks (B_i) each of 512-bit. Each block is divided into 16 32-bit words and expanded into 80 32-bit words using Equation (6.1), which produces 2560-bit. These 80 words are represented by the W_t in functional block diagram of the SHA-1, as shown in Figure6.2. The block words are assigned to the first 16 words of

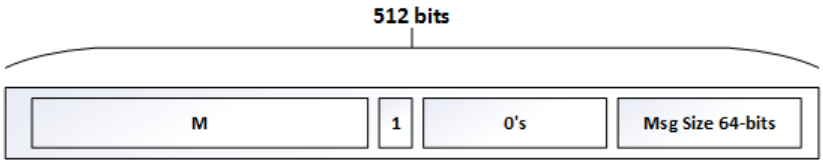

Figure 6.1. Padding the message before processing

the W_t and the rest of the expanded words are calculated using these values and the message expansion equation. The figure shows that five working state variables (A, B, C, D, and E) are used to evaluate the intermediate hash value after each step. These variables are initialized with fixed 32-bit hexadecimal values (H_0=`67452301`, H_1=`EFCDAB89`, H_2=`98BADCFE`, H_3=`10325476`, and H_4=`C3D2E1F0`).

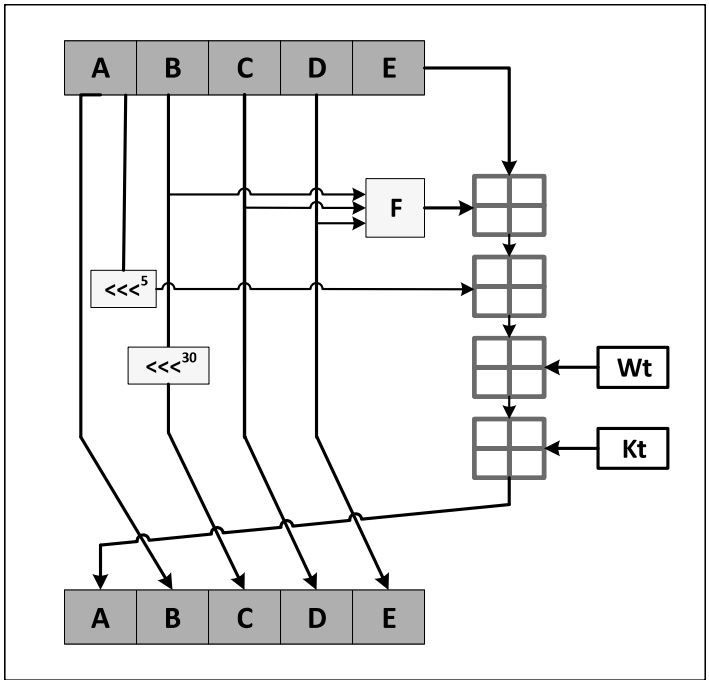

Figure 6.2. Function block of the SHA-1

$$W_t = \begin{cases} B_i^t, & 0 \le t \le 15 \\ (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \lll 1, & 16 \le t \le 79, \end{cases} \tag{6.1}$$

where $\oplus$ is the logical XOR operation, and $\lll 1$ is the cyclic shift rotation to the left by 1-bit. Noting the block B_i is divided into 16 words and these words are assigned to the first 16 values of the W_t.

The function block of the SHA-1 is processed 80 times (steps), where each 20 steps represent a group. Each group has a distinct round function f and constant K_t, as represented in Table 6.1.

The expanded message words are processed as four groups. Each group consists of 20-steps, round constant k_t, and compression function f, as illustrated in Table 6.1.

Table 6.1. The groups functions (f) and constants (K_t) of the SHA-1

Round(steps)	**Function f(B,C,D)**	**Constant (K_t)**
1 (0-19)	$(B \wedge C) \vee (\neg B \wedge D)$	`0x5A827999`
2 (20-39)	$(B \bigoplus C \bigoplus D)$	`0x6ED9EBA1`
3 (40-59)	$(B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$	`0x8F1BBCDC`
4 (60-79)	$(B \bigoplus C \bigoplus D)$	`0xCA62C1D6`

The intermediate hash values (A,B, C, D, and E) change cyclically every step using (6.2), (6.3), (6.4), (6.5), (6.6).

For $i = 1$ to 80,

$$A_i = ROTL^5(A_{i-1}) + F_i(B_{i-1}, C_{i-1}, D_{i-1}) + E_{i-1} + W_i + K_i, \tag{6.2}$$

$$B_i = A_{i-1}, \tag{6.3}$$

$$C_i = RL^{30}(B_{i-1}), \tag{6.4}$$

$$D_i = C_{i-1}, \tag{6.5}$$

$$E_i = D_{i-1}. \tag{6.6}$$

After processing all blocks, the final hash is represented by the concatenation of the five intermediate hash values (A, B, C, D, and E).

6.2.2. SHA-2 Standard

SHA-2 follows Merkle-Damgård structure model. Unlike SHA-1, SHA-2 maintains four different flavors (SHA-224, SHA-256, SHA-384, and SHA-512) that produce different outputs. Each

flavor of the SHA-2 has a unique compression function and round constants. The SHA-2 uses eight variables to process the intermediate hash value (A, B, C, D, E, F, G, and H). The size of the variables depends on the desired flavour. For SHA-224 and SHA-256 flavors, the variable size is equal to 32-bit and 64-bit for the SHA-384 and SHA-512 flavors. SHA-2 maintains 64 steps for 224 and 256 versions and 80 steps for the others.

Like the SHA-1, the SHA-2 compression function evaluation requires a preprocessing phase. A message is padded and divided into blocks, as described in the SHA-1 definition. Each block is expanded using (6.7), then, the expanded values are assigned to W_t.

$$W_t = W_{t-16} + \sigma_0 + W_{t-7} + \sigma_1 \qquad 16 \leq t \leq n, \tag{6.7}$$

where σ_0 and σ_1 are represented by (6.8) and (6.9), respectively.

$$\sigma_0 = RR^{r1}(W_{t-15}) \oplus RR^{r2}(W_{t-15}) \oplus SR^{r3}(W_{t-15}), \tag{6.8}$$

$$\sigma_1 = RR^{q1}(W_{t-2}) \oplus RR^{q2}(W_{t-2}) \oplus SR^{q3}(W_{t-2}), \tag{6.9}$$

where $RR^n(X)$ rotates word X to the right by n bits, and $SR^n(X)$ shift right word X by n bits. For SHA-224 and SHA-256 $n = 63$, $r1 = 7$, $r2 = 18$, $r3 = 3$, $q1 = 7$, $q2 = 19$, and $q3 = 10$, while $n = 79$, $r1 = 1$, $r2 = 8$, $r3 = 7$, $q1 = 19$, $q2 = 61$, and $q3 = 6$ for SHA-384 and SHA-512.

Figure6.3 shows the functional block of the SHA-2 hash function. The intermediate hash values change, cyclically, after each step according to (6.10), (6.11), (6.12), (6.13), (6.14), (6.15), (6.16), (6.17), (6.18), (6.19)

$$T1 = H_{t-1} + W_t + K_t + Ch(E, F, G) + \sum\nolimits_1 \tag{6.10}$$

$$T2 = Maj(A, B, C) \sum\nolimits_0 \tag{6.11}$$

$$H = G, \tag{6.12}$$

$$G = F, \tag{6.13}$$

$$F = E, \tag{6.14}$$

$$E = D + T1, \tag{6.15}$$

$$D = C, \tag{6.16}$$

$$C = B, \tag{6.17}$$

$$B = A, \tag{6.18}$$

$$A = T1 + T2, \tag{6.19}$$

where Ch, Maj, $\sum_0$, and $\sum_1$ are represented by (6.20), (6.21), (6.22), and (6.23), respectively.

$$Ch(E, F, G) = (E \wedge F) \oplus (\neg E \wedge G), \tag{6.20}$$

$$Maj(A, B, C) = (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C), \tag{6.21}$$

$$\sum\nolimits_0(V) = RR^{r1}(V) \oplus RR^{r2}(V) \oplus RR^{r3}(V), \tag{6.22}$$

$$\sum\nolimits_1(V) = RR^{q1}(V) \oplus RR^{q2}(V) \oplus RR^{q3}(V), \tag{6.23}$$

where $\sum_0$ and $\sum_1$ are the message manipulators that are used inside the compression function. The values $r1 = 2$, $r2 = 13$, $r3 = 22$, $q1 = 6$, $q2 = 11$, and $q3 = 25$ are for the SHA-224 and SHA-256, while $r1 = 28$, $r2 = 34$, $r3 = 39$, $q1 = 14$, $q2 = 18$, and $q3 = 41$ are for the SHA-384 and SHA-512.

After processing all blocks and the corresponding working state variables, the final hash is produced by the concatenation of part or all of the working state variables (A, B, C, D, E, F, G, and H), as depicted by (6.24), (6.25), (6.26), and (6.27).

$$SHA_{224} \rightarrow A \| B \| C \| D \| E \| F \| G, \tag{6.24}$$

$$SHA_{256} \rightarrow A \| B \| C \| D \| E \| F \| G \| H, \tag{6.25}$$

$$SHA_{384} \rightarrow A \| B \| C \| D \| E \| F, \tag{6.26}$$

$$SHA_{512} \rightarrow A \| B \| C \| D \| E \| F \| G \| H, \tag{6.27}$$

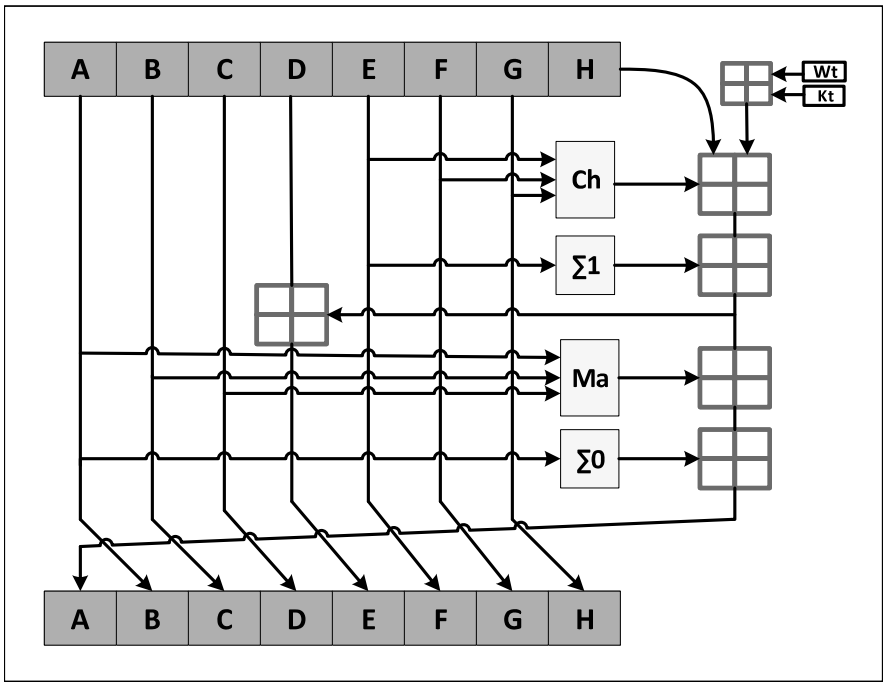

Figure 6.3. Function block of the SHA-2

noting that, the size of each working state variable is 32-bit for the SHA-224 and SHA-256 and 64-bit for the SHA-384 and SHA-512 hash functions.

6.2.3. Bayesian Odd Ratio Test

The purpose of the Bayesian theory is to verify the evidence of the scientific theory that is applied to binomial distribution [9]. Let Θ denotes a *hypothesis*, D denotes the observed sample after running the model, $Pr(\Theta)$ the probability of the model, and $Pr(D)$ the probability of the sample. Then, the Bayesian test states that the conditional probability of the model given the sample ($Pr(\Theta|D)$) is represented by (6.28)

$$Pr(\Theta|D) = \frac{Pr(D|\Theta)Pr(\Theta)}{Pr(D)}, \tag{6.28}$$

where it represents the conditional probability of $\Theta|D$.

The odd ratio test assumes that there are two hypotheses Θ_1 and Θ_2, where Θ_1 represents the model of random population and Θ_2 represents the model of nonrandom population [10]. Then

the *odd ratio* test is calculated using Equation (6.29).

$$\frac{Pr(\Theta_1|D)}{Pr(\Theta_2|D)} = \frac{Pr(D|\Theta_1)}{Pr(D|\Theta_2)}.\frac{Pr(\Theta_1)}{Pr(\Theta_2)}, \tag{6.29}$$

where $Pr(\Theta_1|D)/Pr(\Theta_2|D)$ is the posterior odd ratio, $Pr(D|\Theta_1)/Pr(D|\Theta_2)$ is the *Bayes factor*, and $Pr(\Theta_1)/Pr(\Theta_2)$ is the *prior odd ratio*.

6.3. Related Work

The randomness of cryptography functions was studied by different researchers. The studies used the NIST test suites to analyze the randomness of the bit sequences of certain cryptography functions, e.g., Advanced Encryption Standard (AES) and SHA. Doganaksoy *et al.* proposed a cryptographic randomness testing of block ciphers and SHA [11]. The proposed work, employed Strict Avalanche Criterion (SAC), Linear Span, and Coverage tests. These tests were used to evaluate the internal blocks of the block ciphers and hash functions. Other researchers employed the NIST test suite to test the randomness of block cipher functions.

Hellekalk and Wegenkittl proposed a model to test the block cipher applications. They use the NIST test suite by applying the PRNG [12]. The block ciphers were turned into PRNG and tested using the NIST suite. The empirical results showed that the AES gave interesting results regarding nonlinear RNG.

Kaminsky in [13] proposed a Bayesian statistical test for block cipher and message authentication code (MAC). The proposed design employed the Bayesian test factor and odd ratios to measure the randomness level of block cipher functions. The proposed design was applied to the PRESENT and IDEA block cipher functions, and SipHash and SQUASH MAC functions. The randomness test of the functions showed the number of nonrandom rounds of each function and the ration of nonrandom to the random rounds.

The NIST test suite also employed to test the randomness of the reduced round SHA-3 [14]. The proposed work turned the internal blocks of the SHA-3 reduced round into PRNG then applied the NIST test suite on them. For the reduced round SHA-3, the randomness ratio was 21.3% for 3 rounds of the Keccak hash function. However, this result implies no significance since it was applied on 3 rounds instead of full rounds.

The ALgebric Normal Form (ANF) of random boolean functions to test symmetric ciphers and hash functions was presented in [15]. The proposed work presents new statistical testing for AES, SHA-0, MD4, MD5, Ripmid, and SHA-1 functions. The work employed Pseudo-Random Bit Generator (PRBG) to turn the internal blocks of the functions.

In our work, we analyze the randomness of the SHA functions directly, without turning the blocks into PRNG. Our work is performed on different SHA standards and a suggested modification is proposed. The proposed modification produces more randomness to the output hash than any other flavors of the SHAs.

In the subsequent section, the randomness test, logarithmic *Baeyian factor* test, and *odd ratio* test will be elaborated in details.

6.4. The Randomness Test

6.4.1. Definition of the Randomness Test

The randomness is calculated after each round of the SHA-1 and SHA-2 algorithms. Figure 6.4 shows the block diagram of the randomness test. The input message (M) is fed to the compression function of the SHA-1 or SHA-2. After each round of the compression function calculations, the intermediate hash value is divided into small bit groups (G). Then the randomness test is applied by XORing a certain bit group of the current output (H_i) with the same bit group of the previous round (H_{i-1}). Then, the odds ratio is computed by checking whether the output of the XORing is uniformly distributed. The results of all rounds and bit group categories are stored to apply the odd ratio equation on them. This will be applied for different bit group sizes (1-bit, 2-bit, 4-bit, and 8-bit groups), where each bit group category has its calculations.

6.4.2. Logarithmic Bayes Factor

The *Bayes factor* test is performed using the Bernoulli model, where n Bernoulli trials are performed. The success probability is denoted by χ, which represents the successes number of trials that follows the binomial distribution.

In our model, the *Bayes factor test* is performed on two models (Θ_1 and Θ_2). The probability of success for the Θ_1 and Θ_2 models are χ_1 and χ_2, respectively. Considering that the number of success of n trials is m. Then the *Bayes factor* equations for Θ_1 and Θ_2 are represented by a

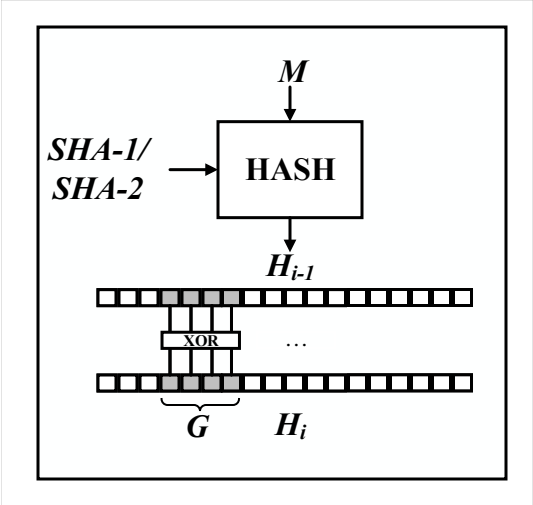

Figure 6.4. The randomness test on the SHA-1 and SHA-2

binomial distribution model, as seen in Equation (6.30).

$$\frac{Pr(D|\Theta_1)}{Pr(D|\Theta_2)} = \frac{n!}{m!\,(n-m)!}\,p^m(1-p)^{n-m}(n+1), \tag{6.30}$$

where p represents χ_1 or χ_2.

To avoid the problem of floating point overflow, the logarithm of the *Bayes factor* is computed instead of the *Bayes factor* [16]. Moreover, the factorial ($n!$) is replaced by a gamma function. According to [17] the gamma function (Υ) of the factorial ($n!$) is represented by (6.31).

$$n! = \Upsilon(n+1)! \tag{6.31}$$

After substituting the Υ function the *Bayes factor* equation becomes:

$$\frac{Pr(D|\Theta_1)}{Pr(D|\Theta_2)} = \frac{\Upsilon(n+1)}{\Upsilon(m+1)\,\Upsilon(n-m+1)}\,p^m(1-p)^{n-m}(n+1). \tag{6.32}$$

Then after applying the logarithm on Equation 6.32, the logarithmic *Bayes factor* becomes:

$$\begin{aligned}\log\frac{Pr(D|\Theta_1)}{Pr(D|\Theta_2)} &= \log\Upsilon(n+1) - \log\Upsilon(m+1)\\ -\log\Upsilon(n-m+1) + m\log p + (n-m)\log(1-p)&\\ +\log(n+1).&\end{aligned} \tag{6.33}$$

The efficient computing of the *Logarithmic Bayes test* using computer programming is the reason for using the logarithmic form [16].

6.4.3. Odd Ratio

The *odd ratio* represents the ratio between the two hypotheses (Θ_1 and Θ_2) in the presence and absence of each one of them toward the other [18]. The *odd ratio test* is performed according to the procedure described in Algorithm 4. The algorithm accepts the input message (M) and the set of bit group sizes (G); in our work, we set G to have four different bit groups (1, 2, 4, and 8-bit groups). The algorithm works for both Secure Hash Algorithms (SHA-1 and SHA-2), which must be determined before the beginning of the test. The bit group categories are processed one after the other, where for each bit group the corresponding SHA calculations are performed through all rounds (t). The value of t is determined according to the definition of SHA, as discussed in Section 2. The Intermediate Hash Value (IHV_{i-1}) of the previous round is assigned to the H_{i-1} and the IHV_i is assigned to the H_i.

Algorithm 4: Odd Ratio

Input: Message (M); `//One Block`
Input: G =[`1, 2, 4, 8`]
Output: Odd Ratio

1 $Determine(SHA)$; `//SHA-1 or SHA-2`
2 **while** $g \in G$ **do**
3 **for** $i \leftarrow 1$ ***to*** $t-1$ **do**
4 $H_i = IHV_i$
5 $H_{i-1} = IHV_{i-1}$
6 **for** $j \leftarrow 1$ ***to*** $t \mod G$ **do**
7 $Res_i^{G_j} = H_i^j \oplus H_{i-1}^j$
8 $\log \frac{Pr(D|\Theta_1)}{Pr(D|\Theta_2)} = \log[\frac{\Upsilon(n+2)}{\Upsilon(k+1)\Upsilon(n-k+1)} p^k (1-p)^{n-k}]$
9 Odd ratio

The IHV value is divided into groups, which are multiple of G. Then the corresponding results of XORing the current and previous analogous bit group position are stored in $Res_i^{G_j}$ array. After processing all bit groups and all internal rounds of the SHA function, the random selection of data (D) is performed by performing n Bernoulli trials considering round (i), bit group (g), and success (k). For each bit group (G), the success probability (p) is equal to 2^{-g}, where g is the

corresponding bit group size. For each bit group, all k successes are counted and the corresponding *logarithmic Bayes factor* is computed for each round (i). Consequently, the *odd ratio* value for each round is the accumulated values of *logarithmic Bayes factor* of the corresponding round and bit group.

6.4.4. Modified SHA

The modified SHA design that we suggest is working by employing the compassion function of the SHA-1 standard and the function manipulators of the SHA-2 standard. The proposed SHA design works as follows:

1. **Padding**: The message padding technique in our design is different. We follow the $10 * 1$ technique where "1" is appended to the end of message then followed by 0's and "1". In this padding method, the message size is not added to the end of message, which protects the message from any targeted attack.

2. **Process**: After padding, the message is divided into blocks of 64-bit. Then, each block is expanded using the expansion equation of the SHA-512. The design works on 80 steps. These steps are divided into four stages, each stage comprises 20 steps. For each stage, the compression function of the SHA-1 is used, as shown in Table 6.1. Each step maintains a distinct constant K. The constant values are generated by taking the least 64-bit of the fractional part of the $\sqrt[3]{Z_p}$ in hexadecimal, where Z_p represent the first 80 prime numbers.

3. **Output**: After processing all blocks, the output hash is produced by the concatenation of the eight working state variables to form 512-bit hash.

6.5. Results and Discussions

The *odd ratio test* is performed on the SHA-1, SHA-2, and A modified SHA function. The modified SHA hash function performs different padding technique by using $10 * 1$ method. In this method, the message size is not added to the end of the message, which excludes the message size from being processed.

As the test performs a massive number of compression function calculations, the execution program was tested using GPGPU unit. The experiments were tested using a configurable experimental environment for large-scale cloud research, Chameleon [19]. On Chameleon, we reserved

a cluster lease with 2-Intel Xeon processors 3.2GHz with 56 threads, 128GB of RAM, and Tesla P100 GPU with 3584 cores.

Table 6.2. Odd ratio results for the SHA-1, SHA-512 and modified SHA

Round	Odd Ratio		
	SHA-1	SHA-512	Modified
1	-1.78×10^8	-1.177×10^7	-1.214×10^8
2	-1.62×10^8	-1.187×10^7	-1.186×10^7
3	-1.55×10^8	-1.168×10^7	-1.159×10^7
4	-1.58×10^8	-1.59×10^7	-1.143×10^7
5	-1.43×10^8	-1.53×10^7	-6.32×10^6
17	-8.68×10^7	-520	-425
20	-7.43×10^6	-65.66	630
21	-6.95×10^5	478	648
27	-100.68	520	645
28	-15.68	680	745
29	200.68	598	635
78	578	633	788
79	598	812	596
80	620	599	623

Table 6.2 shows the results of the *odd ratio* test of the SHA-1, SHA-512, and modified SHA. For space limitation, we omit the results of some rounds. According to the *Bayesian odd ratio*, the number of nonrandom rounds for the SHA-1 is 28. This is because the *odd ratio* values of the first 28 rounds of the SHA-1 give negative values. The SHA-512 performs better than the SHA-1 with 20 nonrandom rounds. The modified version performs the best and produces 17 nonrandom rounds.

Table 6.3 shows the results of performing the *odd ratio* test on the SHA-1, SHA-2, and Modified SHA functions. The SHA-1 produces 28 nonrandom rounds among 80 rounds with nonrandom percentage of 35%. The SHA-224 and SHA-256 produce approximately the same number

Table 6.3. The non-random text results compared to the proposed design

Hash Function	#Non-Random Rounds	Total #Rounds	Non-Random%
SHA-1	28	80	35
SHA-224	19	64	29.68
SHA-256	20	64	31.25
SHA-384	19	80	23.75
SHA-512	20	80	25
Modified SHA	17	80	21.25

of nonrandom rounds with 19 and 20, respectively. The modified SHA function shows the best results in case of nonrandom rounds with 17 non-random rounds.

6.6. Conclusions

This chapter presented a model that examines the randomness of the internal rounds of the SHA-1 and SHA-2 standards. A modified SHA design was proposed to improve the randomness of the internal rounds. The *Bayesian factor* and *odd ratio* tests were used to evaluate the hash functions. The CUDA platform was utilized to perform the experimental analyses, which was implemented with the help of Chameleon large-scale experiments testbed. The results showed that the Modified SHA produced more random rounds other than the other functions.

6.7. References

[1] FIPS PUB. "Secure hash standard (shs)". In: *FIPS PUB 180* 4 (2012), pp. 1–27.

[2] Zeyad Al-Odat, Mazhar Ali, and Samee U Khan. "Mitigation and Improving SHA-1 Standard Using Collision Detection Approach". In: *2018 International Conference on Frontiers of Information Technology (FIT)*. IEEE. 2018, pp. 333–338.

[3] Xiaoyun Wang and Hongbo Yu. "How to Break MD5 and Other Hash Functions". In: *Advances in Cryptology – EUROCRYPT 2005*. Ed. by Ronald Cramer. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 19–35. ISBN: 978-3-540-32055-5.

[4] Zeyad Al-Odat and Samee Khan. "The Sponge Structure Modulation Application to Overcome the Security Breaches for the MD5 and SHA-1 Hash Functions". In: *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*. Vol. 1. IEEE. 2019, pp. 811–816.

[5] NIST DRAFT FIPS PUB and PUB FIPS. "202". In: *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions* (2014).

[6] Juan Soto. "Statistical testing of random number generators". In: *Proceedings of the 22nd National Information Systems Security Conference*. Vol. 10/99. NIST Gaithersburg, MD. 1999, p. 12.

[7] Lawrence Bassham et al. "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications". In: *CSRC | NIST* (Apr. 2010). DOI: 10.6028/NIST.SP.800-22r1a.

[8] Secure Hash Standard. "FIPS Pub 180-1". In: *National Institute of Standards and Technology* 17 (1995), p. 15.

[9] Robert E Kass and Adrian E Raftery. "Bayes factors". In: *Journal of the american statistical association* 90.430 (1995), pp. 773–795.

[10] Jeffrey N Rouder, Paul L Speckman, Dongchu Sun, Richard D Morey, and Geoffrey Iverson. "Bayesian t tests for accepting and rejecting the null hypothesis". In: *Psychonomic bulletin & review* 16.2 (2009), pp. 225–237.

[11] Ali Doganaksoy, Baris Ege, Onur Koçak, and Fatih Sulak. "Cryptographic Randomness Testing of Block Ciphers and Hash Functions." In: *IACR Cryptology ePrint Archive* 2010 (2010), p. 564.

[12] Peter Hellekalek and Stefan Wegenkittl. "Empirical evidence concerning AES". In: *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 13.4 (2003), pp. 322–333.

[13] Alan Kaminsky. *The Coincidence Test: a Bayesian Statistical Test for Block Ciphers and MACs*. 2013.

[14] Ali Doganaksoy, Baris Ege, Onur Koçak, and Fatih Sulak. "Statistical Analysis of Reduced Round Compression Functions of SHA-3 Second Round Candidates." In: *IACR Cryptology ePrint Archive* 2010 (2010), p. 611.

[15] Eric Filiol. "A new statistical testing for symmetric ciphers and hash functions". In: *International Conference on Information and Communications Security*. Springer. 2002, pp. 342–353.

[16] William H Press, Saul A Teukolsky, William T Vetterling, and Brian P Flannery. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

[17] Eric W. Weisstein. *Gamma Function*. "http://mathworld.wolfram.com/GammaFunction.html". Aug. 2019.

[18] Magdalena Szumilas. "Explaining odds ratios". In: *Journal of the Canadian academy of child and adolescent psychiatry* 19.3 (2010), p. 227.

[19] Kate Keahey et al. "Chameleon: a Scalable Production Testbed for Computer Science Research". In: *Contemporary High Performance Computing: From Petascale toward Exascale*. Ed. by Jeffrey Vetter. 1st ed. Vol. 3. Chapman & Hall/CRC Computational Science. Boca Raton, FL: CRC Press, 2018. Chap. 5.

7. A BIG DATA STORAGE SCHEME BASED ON DISTRIBUTED STORAGE LOCATIONS AND MULTIPLE AUTHORIZATIONS

7.1. Introduction

Big Data is defined as a huge amount of different data types that are provisioned through different resources, e.g., social networks, sensor devices, and streaming machines [1]. However, one storage location is unable to handle the increasing size of data; in addition, it is difficult to process Big Data using traditional data processing techniques [2][3].

Big Data is an emerging technology depending on cloud computing where a massive amount of data is processed [4]. The National Institute of Standards and Technology (NIST) defined cloud computing as a model that enables ubiquitous and convenient network access of a shared pool of configured computing resources [5]. Due to increase of data size, the burden on cloud computing sources has increased. An increase demand of cloud computing causes new methods and tools to be invented. Multiple methods are built for supporting cloud computing in processing the Big Data. For instance, rapid update methods should exist to address cloud storage congestion.

The cloud computing technology becomes one of the main components of information computing technology including data acquisition and retrieving from the cloud. Studies by Cisco and IBM show that 2.5 Quintillion Bytes are generated every day and will reach about 40 Yotta Bytes by 2020 [6][7]. The cloud computing provides the suitable service to process the Big Data such as Software as Service (SaaS), Platform as Service (PaaS), and Infrastructure as Service (IaaS). All these services are hired individually or mutually to provide quick and efficient data access [8].

The security of Big Data is crucial because a huge amount of data is stored at the same pool of storage location which leads to data interference [9]. Big Data security is guaranteed

The content of this chapter has been published in the 2019 IEEE 5th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS). The material in this chapter was co-authored by Zeyad Al-Odat, Eman Al-Qtiemat and Samee Khan. Zeyad Al-Odat had primary responsibility for conducting experiments and collecting results. Zeyad Al-Odat was the primary developer of the conclusions that are advanced here. Zeyad Al-Odat also drafted and revised all versions of this chapter. Eman Al-Qtiemat drafted and revised all versions of this chapter. Samee U. Khan revised the material and served as proofreader.

by different technologies, including the following: 1. Encryption, 2. Centralized key management, 3. User access control, 4. Infusion detection and prevention, and 5. Physical security. Moreover, everyone is responsible for Big Data security, e.g., policies, agreement list, and security software are guaranteed by the cloud service provider [10].

One of the main procedures to store the Big Data is the distribution technology. It divides the big data into parts and distributes them over several storage locations [11]. However, many standards need to be addressed in this scheme including data security and retrieval. The data need to be secured against unauthorized access and protected from data tampering and alteration. Data security is achieved using the Encryption techniques, e.g., Advanced Encryption Standard (AES) while data authenticity is achieved using the Secure Hash Algorithm (SHA). In some cases, attackers can hack the Encryption key and gain access to sensitive data.

This chapter proposes a secure and authentic Big Data storage scheme using both Shamir's Secret Sharing (SSS) and SHA. The SSS is used to divide the secret encryption key into parts, this prevents attackers from data decryption even if one part of the encryption key has been obtained. On the other hand, the SHA determines the data integrity through the hash value which is appended with the original data [12].

The rest of chapter is organized as follows: Section 7.2 provides background demonstrations about the secure hash algorithm and Shamir's Secret Sharing; a literature review is presented in Section 7.3; Sections 7.4 exhibits the proposed methodology; results and discussions are in Section 7.5; Section 7.6 concludes the chapter.

7.2. Background

Before going through the details of our proposal, brief descriptions about the secure hash algorithm and Shamir's Secret Sharing will be presented.

7.2.1. Shamir's Secret Sharing

SSS is a method to divide Data (D) into a number of pieces (S_n) where D is easily reconstructable from the minimum number of pieces (S_k) [13]. The SSS is a (k, n) based scheme, where k is the minimum number of pieces that are needed to reconstruct the Data (D) and n is the total number of pieces of data (D). However, D is completely undetermined if the number of knowledgeable pieces is fewer than $k - 1$. Figure 7.1 shows the general structure of the Shamir's

Secret Sharing, where it involves two operations, the divide and reconstruct. More details will be presented in Section 7.4.

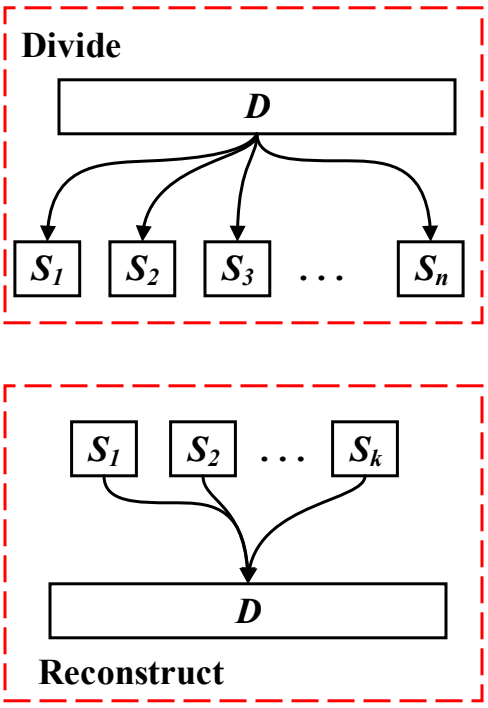

Figure 7.1. Shamir's secret sharing structure

7.2.2. Secure Hash Algorithm

SHA is one of the main cryptography functions. The SHA takes a message (M) of arbitrary size, then through compression function calculations, produces the message hash (H). SHA is used to provide the authenticity and integrity of the data, i.e., ensure that the data are not tampered during transmission or storing.

The secure hash algorithm generates the message hash according to two construction models. The first construction model is Merkle Damgard (MD), which is used to construct the hash functions $MD4$, $MD5$, $SHA-1$, and $SHA-2$ [14]. The second one is the Sponge structure model that constructs the $SHA-3$ hash function [15]. In our proposal, we use the MD structure model to provide data integrity and authenticity. Figure 7.2 shows the function block of the MD structure model. The message M is preprocessed first by padding the input message to make its size a multiple of block size (B).

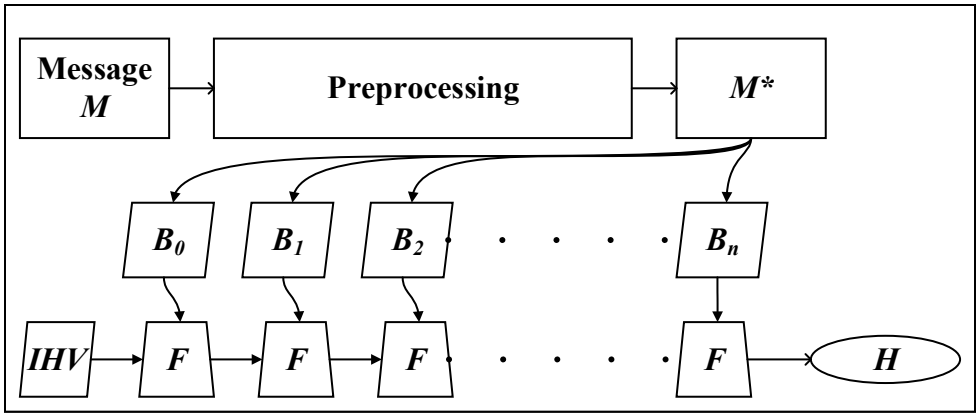

Figure 7.2. General structure of the secure hash algorithm

In the MD hash standards, the maximum message size that each algorithm accepts is dependent on the block size, where the 512-bit block size accepts messages of size less than 2^{64}-bit, while the others accept a message size of 2^{128}-bit. All hash standards perform the following steps:

1. **Message padding**. In this phase, the message is padded with a sufficient number of zeros to make the message size divisible by the block-size.

2. **Message divide**. After the padding phase, the message is divided into equal size blocks (B) where each size is equal to the desired block size, and each hash standard has a specific block size.

3. **Compression function calculation**. The message's blocks are processed sequentially, one at a time, using the round compression functions according to the selected hash standard. Each block is processed a number of times equal to the number of rounds according to the desired hash function. The output of each block is fed as an input to the second block.

4. **Output hash generation**. After processing all message's blocks, the output hash is taken from the output of the last block.

For more details about secure hash algorithms and their compression functions, the reader is referred to [16].

7.3. Related Work

The distributed environment of data centers puts an extra burden to the cloud computing technology for processing and retrieving the Big Data. Particularly, the Big Data security is

considered as an emerging concern in the area of cloud computing because of the distributed nature of data centers [17]. Cheng *et al.* proposed a Big Data storage scheme based on dividing the Big Data into sequenced parts and storing them among multiple cloud storage locations. The proposed work provides a security protection scheme of mapping between different data elements rather than protecting the Big Data. A trapdoor function is employed to encrypt the storage path of the Big Data which in turn protects the data mapping and reduces the encryption time of the Big Data file [17]. However, the proposed design is vulnerable to cryptography attacks which targets the storage path rather than the Big Data, and once breached the Big Data is easily accessible.

The sensitive Big Data security is presented in different publications [11, 18, 19, 20], where the cloud service providers have no chance to access the stored data. Li *et al* in [11] proposed an intelligent cryptography approach for secure distributed sensitive Big Data. The proposed work reduces the chances that cloud operators reach sensitive Big Data. Authors designed a model that securely distributes the Big Data file over multiple storage locations. The Big Data file is divided into parts where each part is encrypted using XOR scheme and sent to the storage locations among the cloud. The proposed design was able to send and retrieve a Big Data file. However, security is related to the security of the Encryption key, and the Big Data file is vulnerable to data modification. In addition, A Security-Aware Efficient Distributed Storage ($SAEDS$) model has been developed in [18] to prevent cloud operators from reaching sensitive data. The algorithm splits the files and saves the data separately in the cloud servers which can significantly increase the scalability of cloud computing in several areas such as the financial industry and governmental agencies. Two algorithms have been proposed to support the $SAEDS$ model: Secure Efficient Data Distributions ($SED2$) is used for data processing prior sending them to the cloud while Efficient Data Conflation ($EDCon$) Algorithm allows users to earn the information by rounding up two data components from distributed cloud servers.

Khan *et al.* proposed a secured Big Data scheme that divides the Big Data file into categories [19]. The proposed design categorizes the Big Data according to its importance, normal and sensitive data. The normal data are stored in the cloud without separation. However, the sensitive data is divided into multiple parts and distributed over the cloud locations. When data are requested, the sensitive data are collected from the storage locations and merged to the normal data and sent to its corresponding user.

Moreover, Dong *et al.* proposed a secure platform to store and share the sensitive Big Data files [20]. The proposed design presents a proxy re-encryption scheme and a process protection method to develop heterogeneous ciphered system functions. In this proposal, authors adopted a process protection technology based on virtual machine monitoring. The virtual machine monitoring is a special key management module that is used to store all public keys that are used in the encryption-decryption process.

A new security framework has been developed in [21] to implement MapReduce tasks on different clusters. The framework allows only a single-sign-on process for jobs submission to G-Hadoop (G-Hadoop is an extension of Hadoop which runs MapReduce tasks on multiple clusters). The proposed model utilizes multiple security solutions such as Secure Sockets Layer (SSL) protocol and a cryptographic mechanism to prevent aggression and misuse of G-Hadoop. This work keeps master-slave architecture of the current G-Hadoop and appending a Certification Authority (CA) server to release proxy and slave credentials.

With the increase of computational power, the Big Data is vulnerable to security breaches, which are not only related to unauthorized access but includes data tampering and sabotage. In this chapter, we are presenting a secure Big Data storage scheme based on secret sharing and modification prevention mechanisms.

7.4. Proposed Methodology

The proposed design is implemented with the aligning to the architecture shown in Figure 7.3. To better understand the figure, please refer to Table 7.1 that shows the used notations in this section.

7.4.1. Distribution Phase

In the distribution phase, the Big Data file (D) is hashed using the SHA-512 hash function. The resulting hash value (H) is appended to the file D, as shown in Figure 7.3. Then, D is divided into parts (D_1, D_2, D_3, ..., D_n), where each part has a unique identification that is needed to reconstruct D. Besides, the hash (H) is appended to the last part of D for storage and encryption. Afterward, the D parts are encrypted using the Encryption key (E) and distributed over multiple cloud locations.

As the Encryption key (E) plays a major rule in the security of D, E is divided into shares (E_1, E_2, ..., E_{x-1}) using the SSS algorithm and distributed over x authorized entities (AEs).

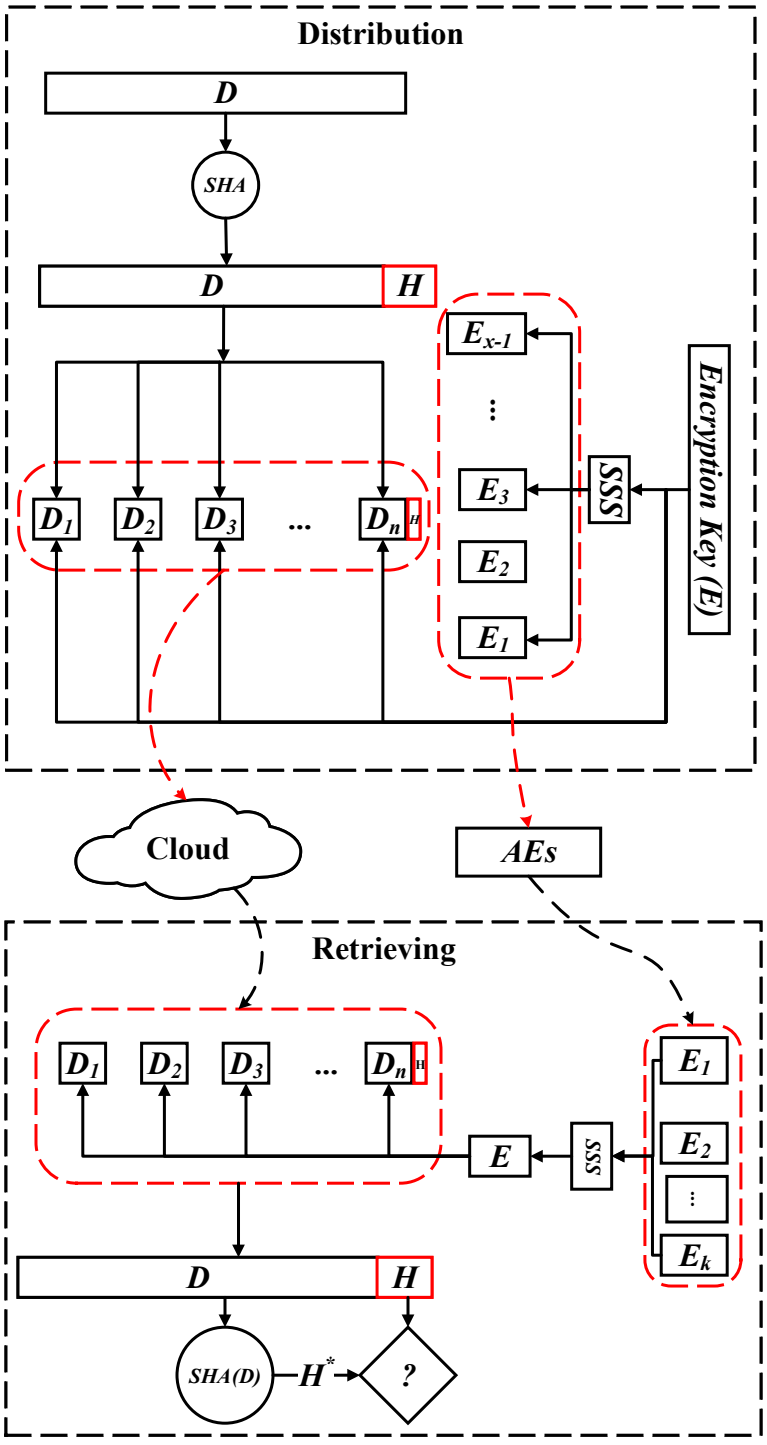

Figure 7.3. Schematic diagram of the proposed methodology

Table 7.1. Notations of the big data storage scheme

Symbols	Meaning
D	The Big Data
SHA	Secure Hash Algorithm ($SHA-512$)
H	Hash value after applying the $SHA-512$
H^*	Hash value after retrieving D
SSS	Shamir's Secret Sharing
D_n	n part of Data D
E	Encryption Key
E_k	k parts of Encryption key E
$L(x)$	Lagrange polynomial to find $L(0)$
CSP	Cloud Service Provider
AE	Authorized Entity
SE	Service Entity that responsible for Data collection

According to the SSS algorithm, the number of shares (k), which are needed to reconstruct the secret key, is represented by a polynomial of power ($k-1$), as shown in Algorithm 5.

Algorithm 5: SSS

Input: Encryption Key (E)
Output: E_0, E_1, ..., E_{x-1}

$Determine(k)$; `//Least number of shares`
for $i \leftarrow 0$ ***to*** $k-1$ **do**
  $a_i = Rand()$
$f(x) = a_0 + a_1x + a_2x^2 + ... + a_{k-1}x^{k-1}$
$Determine(q)$; `//Total number of shares`
for $x \leftarrow 1$ ***to*** $q-1$ **do**
  $E_x = (x, f(x))$

The algorithm shows the general procedure to divide E into shares (E_{x-1}). First, the least number of shares (k) that are needed to reconstruct E is determined. Then, k random numbers (a_i) are generated to construct the polynomial equation ($f(x)$), where k represents the least number of needed shares to reconstruct the secret key (E). Afterward, all shares are constructed using the

polynomial equation ($f(x)$) by determining the total number of secret shares (q), where each share (E_{x-1}) is represented as a pair $(x, f(x))$.

7.4.2. Retrieving Phase

At the Cloud, the Big Data parts are distributed through multiple Cloud locations where the CSP unable to access any of the D parts because of the data encryption. Moreover, the CSP has no access to the Encryption key because it is divided into shares that are distributed to authorized entities only. In our design, one of the authorized entities (SE) is responsible for the data collection and decryption. However, a single authorized entity is unable to retrieve D because SSS algorithm is used in the design.

To retrieve the Big Data file, half of the AEs need to authorize the data access by giving their E_x where k shares are needed to reconstruct E. Once the least number of E_x shares are collected, then the SE computes E using Equation 7.1. Afterward, the data parts (D_n) are collected from the storage locations and assembled according to their identifiers.

Equation 7.1 shows the polynomial calculation procedure to retrieve the Encryption key (E). The equation is called Lagrange polynomials where each point pair $(x, f(x_i))$ refers to one share.

$$L(0) = \sum_{j=0}^{k-1} f(x_j) \prod_{\substack{m=0 \\ m \neq j}}^{k-1} \frac{x_m}{x_m - x_j}, \tag{7.1}$$

where $L(0)$ represents the retrieved key (E).

After decryption, one more step is needed to verify the integrity of data. The secure hash algorithm is used to accomplish the last step, where the SHA-512 is used to compute the hash value (H^*) of the retrieved data D. Then, the computed hash value (H^*) and the appended hash value (H) are compared to determine whether they are equal. If D is correctly gathered and not modified during transmission, the values of H and H^* are equal. Otherwise, D is corrupted and contains tampered contents.

7.5. Results and Discussion

The proposed design is tested and verified using sample files of different sizes. The experiments were tested using a configurable experimental environment for large-scale cloud research, Chameleon [22]. All samples and testing results were generated on the Chameleon environment.

7.5.1. Experimental Analysis

Figure 7.4 shows the used samples and their corresponding sizes in MegaByte (MB). The Maximum test file size is $\approx 5.2GB$ while the minimum test file size is $\approx 110MB$.

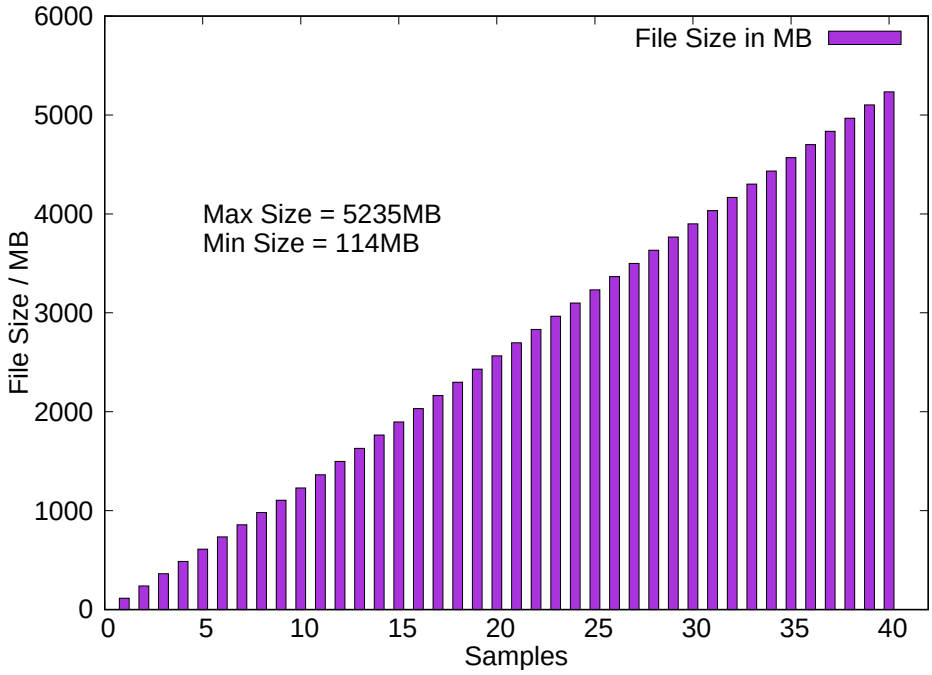

Figure 7.4. Size of the sample file measured in MB.

To test the speed of the proposed design, we measured the elapsed time to collect and hash the test samples, as shown in Figure 7.5. The figure shows that the hashing time is small when compared to the collection time, which reflects the speed of the hash algorithm in processing the samples. However, the figure shows that the hash time slightly increases when the size of the test file is changed. On other hand, the needed time to collect the data parts is relatively related to the sample size, where they are increasing respectively. Moreover, the increase of data file size affect on the size of each part of the data, which is reflected to the total time to collect and hash the sample files.

7.5.2. Security Analysis

We express the security analysis from the Data owner perspective where the security of Big Data is guaranteed by the proposed scheme according to four definitions.

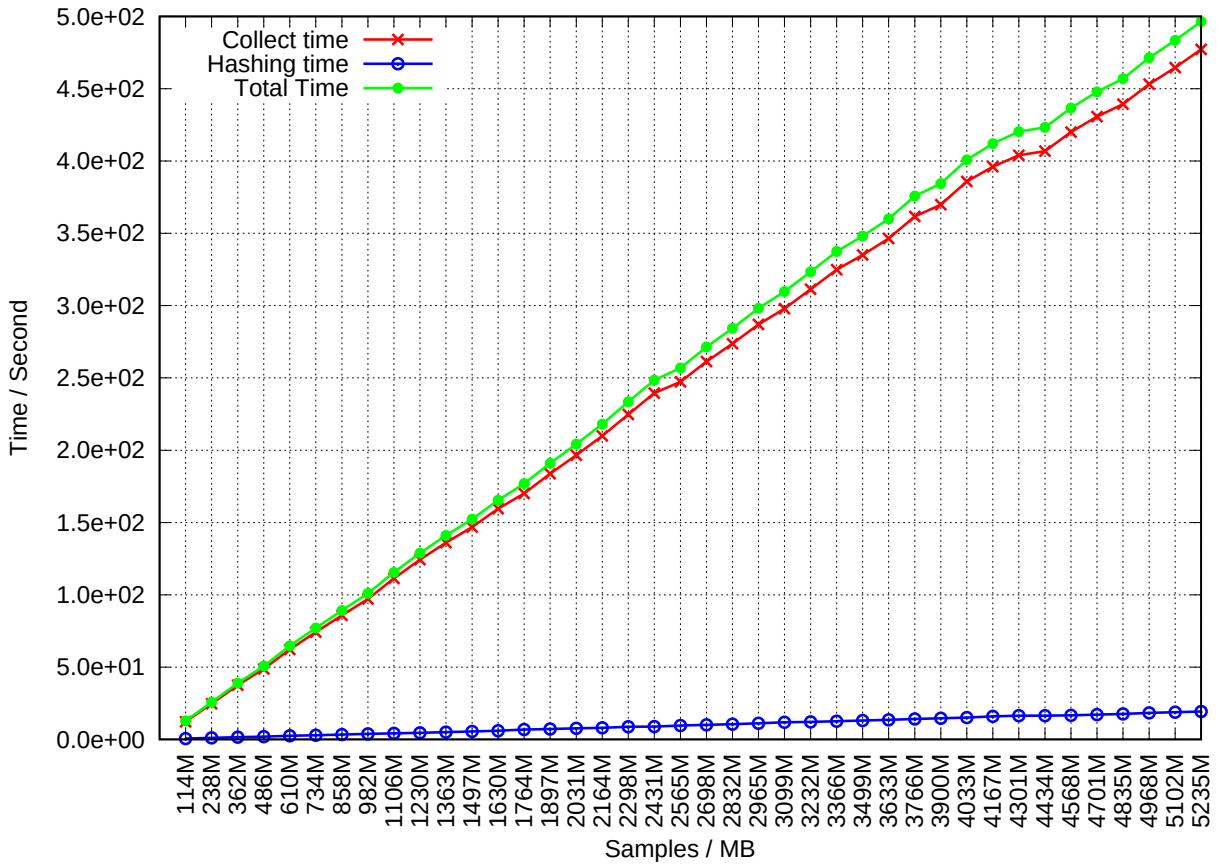

Figure 7.5. The time needed to collect and hash the Big Data files.

Definition 1 *The Big Data parts (D_n) are stored on multiple cloud locations and they are collected by the SE.*

The first level of security is accomplished by the SE, where the distributed data parts are collected and combined together. In addition, the SE acquires one share of the Decryption key.

Definition 2 *The encryption key (E) is divided into shares, where each share is given to an authorized entity, and at least half of the authorized entities must provide their secret shares to reconstruct E.*

The second level of security is obtained using the SSS algorithm, which keeps the encryption key secure even if few shares are breached. This property reduces centralized data control by distributing E over multiple users.

Definition 3 *The data integrity and authenticity are conserved by using the SHA hash algorithm.*

The third level of security is achieved by using the SHA, where the SHA-512 is employed to ensure that the Big Data file is tamper-free during transmission or storage, and the Big Data parts are reconstructed with the correct order.

Definition 4 *The client concerns about data exposure to the CSP is reduced.*

The data owner ensures that the CSP has no access to the stored data. This is achieved by the data encryption and distributed encryption key, where the CSP involvement is only restricted to provide the storage locations. Moreover, the data parts can be distributed through different $CSPs$, where the SE keeps track of each part using the identification number.

The proposed design is compared with other works with respect to data integrity, CSP prevention, centralized control, and average time to retrieve data from the cloud. Table 7.2 shows the comparison between the proposed design and recent works. The proposed design accomplished the security requirements of data integrity and CSP prevention. Moreover, the use of SSS consolidates the proposed design against centralized control.

Table 7.2. Comparison with other works

Work	Data Integrity	CSP Prevention	Decentralized Control	Avg Time/ (s)
[20]	✘	✔	✘	-*
[19]	✘	✔	✘	-*
[18]	✘	✔	✘	35E3
Proposed	✔	✔	✔	3.5E2

* Unreported result.

The proposed design outperforms the other works (the reported results) in term of average time needed to retrieve data from the cloud. The employment of the SHA and SSS algorithms consolidates the cloud-based storage system and increases the level of client trust in the $CSPs$.

7.6. Conclusions

In this chapter, a secure storage scheme for distributed Big Data over the Cloud is presented. The proposed design employs the SHA-512 and SSS to ensure Big Data integrity and security. The experimental results show that the proposed design can handle a large data file with a reasonable

time while preserving the security and authenticity properties. Moreover, the trust level between the cloud service provider and the client is increasing because of the assured data authorization.

Big Data security is an open scope. In the future, further security analysis will be carried out, in particular, the deployment of secure hash algorithms in future designs.

7.7. References

[1] Chandu Thota, Gunasekaran Manogaran, Daphne Lopez, and V Vijayakumar. "Big data security framework for distributed cloud data centers". In: *Cybersecurity breaches and issues surrounding online threat protection*. IGI global, 2017, pp. 288–310.

[2] Ibrahim Abaker Targio Hashem et al. "The rise of "big data" on cloud computing: Review and open research issues". In: *Information systems* 47 (2015), pp. 98–115.

[3] Colin Tankard. "Big data security". In: *Network Security* 2012.7 (2012), pp. 5–8. ISSN: 1353-4858. DOI: https://doi.org/10.1016/S1353-4858(12)70063-6. URL: http://www.sciencedirect.com/science/article/pii/S1353485812700636.

[4] Zhimiao Fang, Zhenghan Wu, and Le Luo. "Research on Computer Information Processing Technology in the "Big Data" Era". In: *IOP Conference Series: Materials Science and Engineering*. Vol. 392. 6. IOP Publishing. 2018, p. 062197.

[5] Hybrid Cloud. "The NIST Definition of Cloud Computing". In: *National Institute of Science and Technology, Special Publication, 800* 145 (2011), pp. 1–18.

[6] Robert Pepper and John Garrity. "The internet of everything: How the network unleashes the benefits of big data". In: *Global Information Technology Report 2014* (2014).

[7] Steve LaValle, Eric Lesser, Rebecca Shockley, Michael S Hopkins, and Nina Kruschwitz. "Big data, analytics and the path from insights to value". In: *MIT sloan management review* 52.2 (2011), p. 21.

[8] Mohamed Abdel-Basset, Mai Mohamed, and Victor Chang. "NMCDA: A framework for evaluating cloud computing services". In: *Future Generation Computer Systems* 86 (2018), pp. 12–29.

[9] Colin Tankard. "Big data security". In: *Network security* 2012.7 (2012), pp. 5–8.

[10] Rongxing Lu, Hui Zhu, Ximeng Liu, Joseph K Liu, and Jun Shao. "Toward efficient and privacy-preserving computing in big data era". In: *IEEE Network* 28.4 (2014), pp. 46–50.

[11] Yibin Li, Keke Gai, Longfei Qiu, Meikang Qiu, and Hui Zhao. "Intelligent cryptography approach for secure distributed big data storage in cloud computing". In: *Information Sciences* 387 (2017), pp. 103–115.

[12] Zeyad Al-Odat, Mazhar Ali, and Samee U Khan. "Mitigation and Improving SHA-1 Standard Using Collision Detection Approach". In: *2018 International Conference on Frontiers of Information Technology (FIT)*. IEEE. 2018, pp. 333–338.

[13] Adi Shamir. "How to Share a Secret". In: *Commun. ACM* 22.11 (Nov. 1979), pp. 612–613. ISSN: 0001-0782. DOI: 10.1145/359168.359176. URL: http://doi.acm.org/10.1145/359168.359176.

[14] Ivan Bjerre Damgård. "A Design Principle for Hash Functions". In: *Advances in Cryptology — CRYPTO' 89 Proceedings*. Ed. by Gilles Brassard. New York, NY: Springer New York, 1990, pp. 416–427.

[15] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. "Sponge functions". In: *ECRYPT hash workshop*. Vol. 2007. 9. Citeseer. 2007.

[16] FIPS PUB. "Secure hash standard (shs)". In: *FIPS PUB 180* 4 (2012), pp. 1–27.

[17] Hongbing Cheng, Chunming Rong, Kai Hwang, Weihong Wang, and Yanyan Li. "Secure big data storage and sharing scheme for cloud tenants". In: *China Communications* 12.6 (2015), pp. 106–115.

[18] Keke Gai, Meikang Qiu, and Hui Zhao. "Security-aware efficient mass distributed storage approach for cloud systems in big data". In: *2016 IEEE 2nd International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing (HPSC), and IEEE International Conference on Intelligent Data and Security (IDS)*. IEEE. 2016, pp. 140–145.

[19] Sana Khan and Ekta Ukey. "Secure Distributed Big Data Storage Using Cloud Computing *". In: *IOSR Journal of Computer Engineering* (2017), pp. 8–12.

[20] Xinhua Dong et al. “Secure sensitive data sharing on a big data platform”. In: *Tsinghua science and technology* 20.1 (2015), pp. 72–80.

[21] Jiaqi Zhao et al. “A security framework in G-Hadoop for big data computing across distributed Cloud data centres”. In: *Journal of Computer and System Sciences* 80.5 (2014), pp. 994–1007.

[22] Kate Keahey et al. “Chameleon: a Scalable Production Testbed for Computer Science Research”. In: *Contemporary High Performance Computing: From Petascale toward Exascale*. Ed. by Jeffrey Vetter. 1st ed. Vol. 3. Chapman & Hall/CRC Computational Science. Boca Raton, FL: CRC Press, 2018. Chap. 5.

8. ANONYMOUS PRIVACY-PRESERVING SCHEME FOR BIG DATA OVER THE CLOUD

8.1. Introduction

Both, cloud computing and Big Data, are two emerging trends in the information technology industries [1]. Cloud computing provides the required services to process and stores the Big Data. These services encompass, as defined by the National Institute of Standards and Technology (NIST), Software as Service (SaaS), Infrastructure as Service (IaaS), and Platform as Service (PaaS) [2]. The Big Data files become huge and complex because of the variety and size of the newly uploaded data to the cloud. For instance, the cloud processes many kinds of data structures from different sources, e.g., social network data, medical records, and commercial transactions [3].

To process the large and complex Big Data, MapReduce technology is widely used by different entities to process their data. The MapReduce collaborates with cloud computing to produce salable and efficient platforms. This collaboration leads to a convenient infrastructure that helps the entities to process and store their data effectively and rapidly [4].

However, the privacy of the big data over the cloud becomes a concern because the sensitive information is distributed over various locations and can be retrieved easily. Particularly, when the Big Data files are not encrypted. This leads to the second issue, which is the encryption and decryption of Big Data files. It becomes one of the main issues of storing and retrieving of Big Data file because of the computational time of the encryption and decryption. For instance, a medical record of a patient inside a big dataset requires the decryption of the whole dataset to retrieve the designated record.

There are four well-known mechanisms of the privacy-preserving and protection; which are encryption, access control, auditing, and differential privacy [5]. However, these mechanisms are still considered as open issues in the area of Big Data and cloud computing [6]. Generally, the

The content of this chapter has been submitted to the 2019 IEEE International Conference on Big Data (IEEE Big Data 2019). The material in this chapter was co-authored by Zeyad Al-Odat and Samee Khan. Zeyad Al-Odat had primary responsibility for conducting experiments and collecting results. Zeyad Al-Odat was the primary developer of the conclusions that are advanced here. Zeyad Al-Odat also drafted and revised all versions of this chapter. Samee Khan drafted, revised all versions of this chapter, and served as proofreader.

uploaded data to the cloud are dynamically updated and not only considered for storage, e.g., online applications. The traditional way to protect data is encryption, which considered as a challenging task because the majority of applications run on unencrypted datasets. Moreover, in most applications, the data owners and users are different entities, which makes the encryption mechanism inefficient to ensure privacy and high-performance systems [7].

To achieve the goals of privacy, access control, and utility of Big Data, the data anonymization is considered as a promising approach. However, the infrastructure of data anonymization moves to the MapReduce framework to support large-scale datasets. The framework infrastructure is provided by a Cloud Service Provider (CSP) [8]. The CSP provides the required services and management protocols to support the uploaded data with effective cost and flexible usage [9]. However, sensitive information is uploaded to the cloud and exponentially increasing. This information comes from different sources and structures. Therefore, different data-mining techniques were proposed to support the process of data extraction while preserving the privacy of the data.

In this chapter, we present an efficient Anonymous Privacy-Preserving Scheme (APPS) for Big Data over the cloud based on the MapReduce framework. The proposed design helps to maintain data privacy and integrity before they are accessed by the MapReduce. The SHA-512 hash function is employed to provide the integrity requirements for the APPS framework. Moreover, the functional encryption plays a major role in our proposal, because it provides on-demand encryption/decryption paradigms.

The rest of chapter is organized as follows. Section 2 gives preliminary information related to our scheme. Section 3 provides a literature review of the related works. The proposed methodology is presented in section 4. Results and Discussion in section 5. Section 6 concludes the chapter.

8.2. Preliminaries

8.2.1. MapReduce

MapReduce is a large-scale data processing framework that allows handling and process big data files in parallel [10]. Google introduced the MapReduce framework in 2004, which got the attention of many researchers from different fields [11]. The MapReduce maintains three features, simplicity, scalability, and fault tolerance. Therefore, many entities got benefited from the MapReduce services.

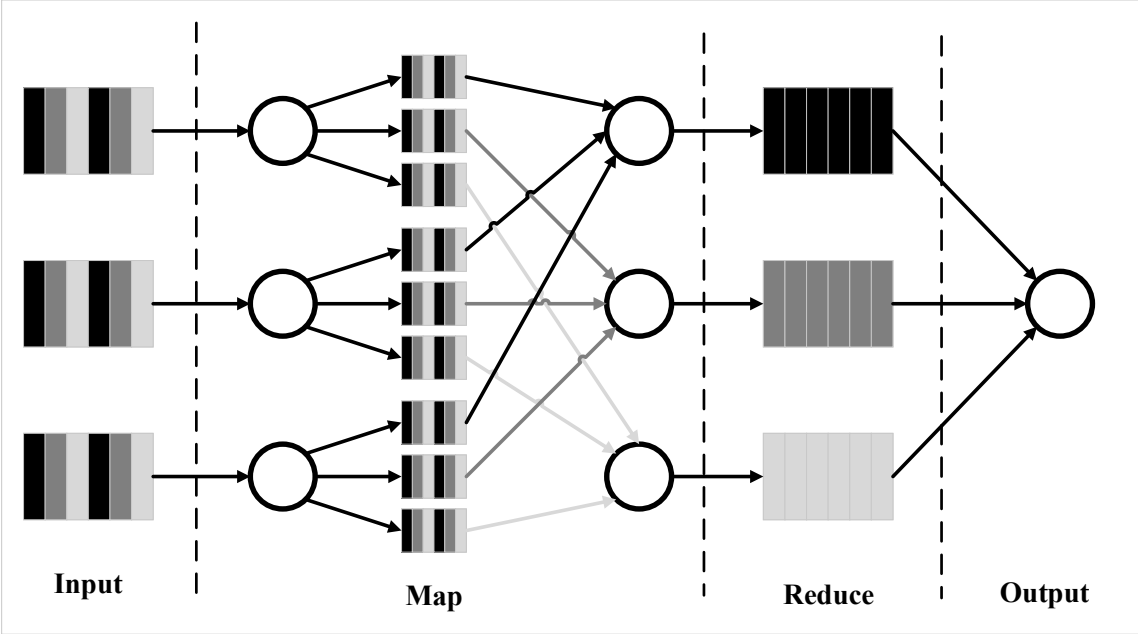

Figure 8.1. General structure of the MapReduce functionality

Figure 8.1 shows the general structure of the MapReduce functionality. The MapReduce consists of two main functions, Map and Reduce. The Map function takes a sequence of values as input and applies a distinct function on each value, i.e., maps an input pair (k_1, v_1) to another pair (k_2, v_2). The reduce function takes a sequence of elements that are linked to the k_2 value and combines them to produce the output pair (k_3, v_3). As shown in the figure, the map function maps the corresponding colors from a big list, then the reduce function combines each color to its corresponding group. Afterward, the output will show the results according to the user query.

MapReduce can be implemented on different implementations schemes depending on the environment. For instance, some implementations are suitable for small networks and others are suitable for large connected machines.

8.2.2. Functional Encryption

The functional encryption (FE) is defined as an encryption methodology that describes the functions of plain text that can be learned according to a distinct function F [12]. The FE scheme consists of four algorithms, `Setup`, `Keygen`, `Enc`, and `Dec`. The details of each algorithm is shown in Table 8.1. The FE works by generates a pair of public and master keys (P_k, MS_k). Then for each defined functionality (F), a secret key (S_k) is generated using the MS_k key. A message (m) is encrypted using the public key (P_k) to produce a ciphered message (c). Afterward, if some type

of information is requested using function (F), the secret key (S_k) of the function (F) is used to reveal the information from c. Noting that, the decryption is applied to a part of the ciphered text, which saves time in case of big data.

Table 8.1. The four algorithms of the functional encryption

Algorithm	**Parameters**	**Definition**
`Setup`	P_k, MS_k	Creates public key (P_k) and master key (MS_k)
`Keygen`	MS_k, F, S_k	Generate secret key (S_k) for the function F using the MS_k
`Enc`	P_k, m, c	Encrypts a message (m) using the P_k to produce encrypted message (c)
`Dec`	S_k, c, y	calculates $y = f(m)$ using the S_k from ciphered data c

8.2.3. Secure Hash Algorithm (SHA)

SHA is one of the main cryptography functions that compress a message (M) of a predefined size to a fixed size output hash (H). SHA is used to check the integrity of data, i.e., ensure that the data are tamper-free during transmission or storing.

The SHA follows two models to construct the internal compression function operations, Merkle Damgard (MD), and Sponge structures. The MD structure is used to construct the hash functions MD4, MD5, SHA-1, and SHA-2 [13]. The Sponge structure is used to constructs the SHA-3 hash function [14]. Because of the compression speed of the MD structure, we adopted the SHA-2 hash function in our proposal.

In the MD hash standards, the maximum message size that each algorithm accepts is dependent on the block size, where the 512-bit block size accepts messages of size less than 2^{64}-bit, while the others accept a message size of 2^{128}-bit. Figure 8.2 shows the general operation of the MD structure model, which is performed according to the following steps:

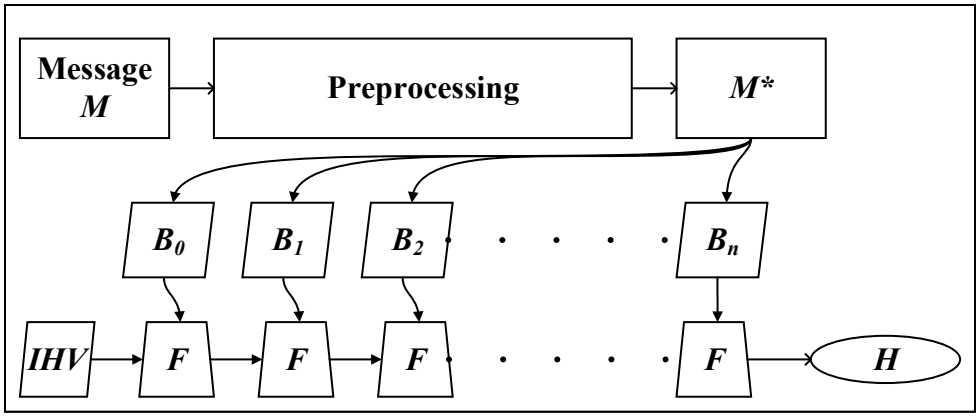

Figure 8.2. General structure of the secure hash algorithm

1. **Preprocessing**. In this phase, the message is padded with a sufficient number of zeros to make the message size divisible by the block-size (B). The message after preprocessing is denoted by M^*.

2. **Message divide**. After padding, the message is divided into equal size blocks (B) to make them ready for compression.

3. **Compression function calculation**. The blocks are processed sequentially, one at a time, using the compression function (F). Each block is processed a number of times equal to the number of rounds that each function employ. The Initial Hash Value (IHV) is fed as an input to process the first block. Then, the output of processing each block is fed as input to the next block calculation. This process continues until all blocks are processed.

4. **Output hash generation**. After processing all message's blocks, the hash value is taken from the output of the last block calculation.

For more details about secure hash algorithms and their compression functions, the reader is referred to [15].

8.2.4. Threat Model

- Privacy-Preserving. The privacy of the Big Data is considered a crucial issue, particularly, when all data are stored in the cloud. There should be a high level of trust between the data owner and the CSP.

- Integrity Threat. The shared data in the cloud might be corrupted by an adversary or the CSP loses the stored data due to hardware failure. Moreover, the collision attack forms a serious threat when a weak secure hash algorithm is used [16].

- Encryption/Decryption Time. The time of encryption and decryption need to be enhanced, especially, in the big data environment. As the data are increasing over time, then the encryption will be an obstacle to overcome.

8.3. Literature Review

The privacy of big data files over the cloud is considered as a crucial issue; because of the amount of sensitive information that might be compromised by adversaries. Many researchers in the field of big data security focused on the privacy-preserving of sensitive information over the cloud [17, 18, 19].

A secure privacy-preserving scheme for on-demand cloud service was proposed in [17]. The proposed work helps to prevent data loss over the cloud and increase the level of trust between the cloud service provider and the data owners. Moreover, the privacy of the stored information over the cloud is maintained and the data owners ensure that the CSP is not able to access the encrypted information on the cloud. the work focuses on the portability of users' information with trusted CSP and store them in locations that are ambiguous from the CSP.

Usually, the datasets on the cloud change frequently. Therefore proper privacy-preserving schemes are needed to support the incremental nature of the data over the cloud. Aldeen *et al.* proposed an innovative privacy-preserving scheme for incremental data on the cloud [18]. The proposed design employs an anonymization technique to attain data privacy over distributed and incremental big data. The proposed design works by dividing the data into small parts then stores them on distributed locations over the cloud. The anonymized data are divided according to a predefined anonymization level (k). Then, any new updates to the data will be handled by initializing the anonymized datasets. The evaluation of the design showed enhancements in the execution time of the incremental design over the classical data incremental method.

The privacy of mobile cloud systems is also considered as the same issue as the big data over the cloud. Lo and Gokay prop a hybrid mobile-cloud model based on the concept of cloudlet [19]. Two models were employed to construct the hybrid model, the cooperative and centralized models.

One master cloudlet is responsible for the management of the data transfer, ensures the data privacy, and security of communication channels. The proposed design was tested and verified using Mobile Cloud Computing Simulator (MCCSIM) to measure the delay time and consumed power when applying the design.

However, the encryption and decryption operations are considered as part of the dilemma when dealing with big data. Furthermore, the dilemma will increase when considering secure big data schemes with efficient cryptography approaches [20]. To enhance the computational cost of the big data encryption, Li *et al.* proposed an encryption scheme based on Attribute-Based Encryption (ABE) [21]. The proposed design combines the ABE with MapReduce functionality. The encryption of the big data is outsourced to a trusted CSP by maintaining a master node and several slave nodes. The master node provides a set of available slave nodes to perform a specific task. Then all involved salve nodes are employed as mappers to get the task completed using the MapReduce paradigm. The computational cost, using the proposed design, is reduced into four exponentiations which save more time than the classical encryption method.

The idea of Homomorphic Encryption (HE) for faster encryption of big data was adopted by Wang *et al* [22]. They proposed a faster HE scheme for big data over the cloud. A Fully HE (FHE) along with Dijk, Gentry, Halevi and Vaikuntanathan's (DGHV) schemes were employed to improve the encryption speed. The public key size is reduced using Pseud Random Number Generator (PRNG) in cubic form rather than linear form. The security of the proposed design was considered and proved under the error-free approximation problem. Moreover, the system model of the FHE scheme of big data is illustrated.

The anonymous data distribution of Big Data over the cloud is also adopted by many researchers [23, 24, 25]. The anonymous data are distributed over the cloud using the MapReduce framework, where the privacy requirements need to be maintained. Zhang *et al.* proposed a privacy-preserving architecture over the cloud using the MapReduce and anonymization functionalities. The proposed design is built on top of MapReduce framework to achieve the security components of the design. The design support flexible and dynamical data update, and cost-effective framework [23]. The MapReduce framework for data anonymization is also adopted by Mohammed *et al.* [24]. They presented a centralized and distributed anonymization scheme for healthcare data. Through the design, they propose a new privacy model called LKC-privacy. To further enhance the security of

anonymized Big Data, An enhanced secured MapReduce design is presented in [25]. They propose the Secure Map Reduce (SMR) model to guarantee the privacy and security concerns, which is built as a layer between the MapReduce and Hadoop Distributed File System (HDFS).

In this chapter, we propose an Anonymous Privacy-Preserving Scheme (APPS) based on the MapReduce framework. This proposal helps to overcome the security and computational issues that appeared in the area of Big Data processing. In the subsequent text, we present our proposal and all related design algorithms.

8.4. Proposed Methodology

We present the APPS model based on MapReduce and functional encryption paradigms. In the APPS, the privacy of big data on the cloud is preserved using the functional encryption and MapReduce. Moreover, we employed the SHA-512 hash function to preserve the integrity requirements of the APPS. To save the encryption time, some available nodes on the cloud are incorporated to achieve the encryption task. Before going through the details of our proposal, please refer to Table 8.2 to clarify the meanings of the symbols that will be used in the design.

Table 8.2. Notations of the APPS design

Symbols	**Meaning**
FE	Functional Encryption
SHA-512	Secure Hash Algorithm (*SHA-512*)
CT	Ciphered-Text
MEK	Mapper Encryption Key
MS_k	Master key
P_k	Public key
S_k	Private key
ENC	Encryption
DEC	Decryption
I_{key}	policy attributes used to generate private key
I_{enc}	policy attributes to functionally encrypt data
CSP	Cloud Service Provider

8.4.1. General Structure

Figure 8.3 shows the general structure of the proposed scheme. Two levels of MapReduce were utilized to achieve encryption and decryption tasks. Before uploading data to the cloud, the user specifies the number of available nodes in the cloud before uploading the data. The big data file is divided into smaller parts then they are transferred to the predefined nodes (each node is called a *mapper*) for encryption. After encryption, the SHA-512 hash function is applied to each part. After hashing, the Ciphered-Text (CT) of each part is stored in the cloud. On another hand, the MapReduce task is performed to retrieve the requested information from a data part. The data part that contains the desired information is determined first. Then, the integrity of the selected part is examined using the SHA-512 hash function. Afterward, the second level of MapReduce is applied to the selected part. After determining the part that contains the desired information, the FE is used to decrypt the information that is requested by a user. The complete operations are described as follows:

1. **Divide Phase.** The Big Data file is divided into smaller parts, where each part is signed using the SHA-512 hash function. The hash value of each part is appended to the end of its corresponding part.

2. **Upload Phase.** The data owner uploads the divided data and a set of instructions to the mappers. Each mapper receives part of the data and the encryption instructions to produce functionally encrypted ciphered-text. More details about the instructions are coming through this section.

3. **Map Phase.** Each mapper takes attribute instructions and Mapper Encryption Key and produce the functionally encrypted cipher-text (CT_i).

4. **Reduce Phase.** The reduce function is performed when a result is requested. The mappers work on the reduce phase to bring the result to a user. then the user needs to decrypt the received result using the associated keys. More details will be elaborated later.

5. **Integrity Check.** The associated hash value is used to check the integrity of the received data.

6. **FE phase.** The desired part of data is revealed using the FE algorithm.

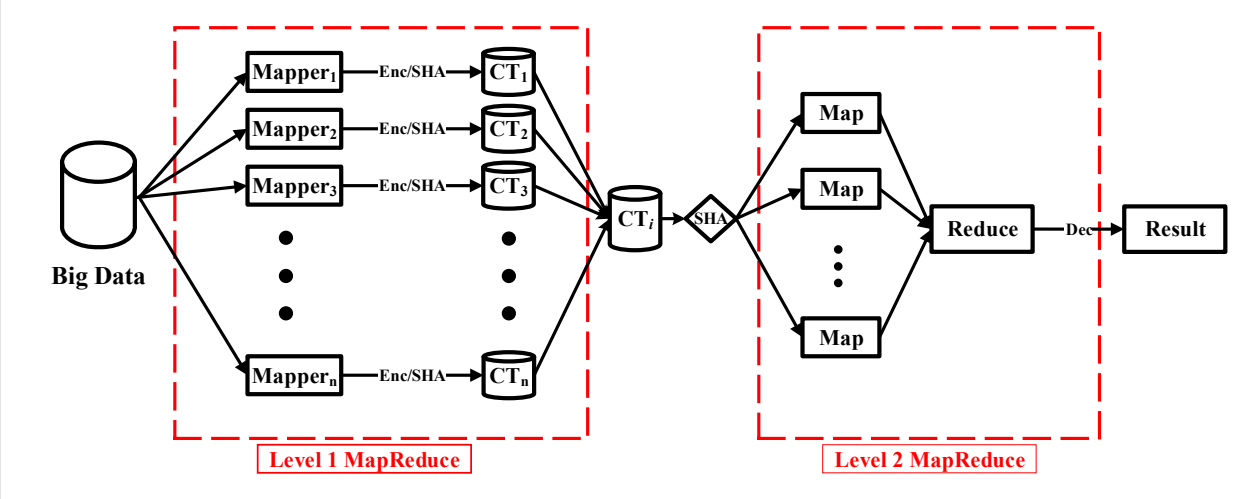

Figure 8.3. General structure of the proposed design

8.4.2. System Model

The proposed system comprises the following algorithms:

1. `Setup(`λ`,`I_{enc}`)`: Number of security parameters (S) are taken as input to produce public (P_k) and Master key (MS_k); a security parameter is denoted by λ. According to a set of encryption policies (I_{enc}), a set of Mappers Encryption Keys ($\{MEk_i\}_{i=1}^{n}$) for n mappers in the cloud are produced.

2. `KeyGen(`I_{key}, MS_k`)`: The master key (MS_k) and a set of attributes (I_{key}) are used to generate a private key (S_k) for these set of attributes.

3. $ENC(I_{enc}, \{MEk_i\}_{i=1}^{n})$: The encryption policies (I_{enc}) and the $\{MEC_i\}_{i=1}^{n}$ are used to encrypt the message (m). It produces the ciphered-text (CT_i).

4. `DEC(CT,`S_k`)`: The ciphered-text (CT) and the secret key (S_k) are used to decrypt the message according to the policy attributes identified by the I_{key} and I_{enc}.

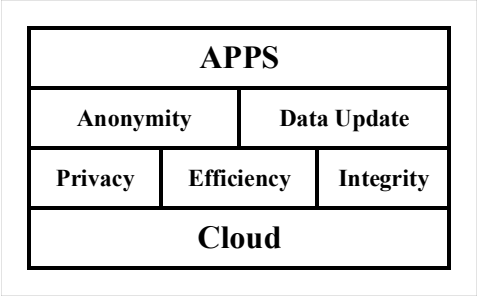

Figure 8.4. APPS framework

8.4.3. Data Management and Anonymization

In our proposal, we used Hadoop data anonymization implementation to deploy our algorithm. Hadoop provides an efficient data anonymization implementation and a reliable infrastructure that processes the data in parallel. In our proposal, five main properties are maintained:

1. data anonymization,
2. privacy-preserving,
3. data Update,
4. efficient Encryption/Decryption scheme, and
5. data integrity.

The anonymized data are kept hidden from the data owners, and only accessed by the MapReduce functionality. The privacy of the data owners is preserved using the utilized anonymization technique, which hides the identity of the data owner. In the case of incremental data, the proposed design supports the addition of new data, which can be added and retrieved efficiently using the MapReduce platform. The time of data encryption and decryption is reduced significantly because the operations of encryption and decryption are delegated to the CSP. The integrity of data is maintained by using the SHA-512 hash function.

8.4.4. APPS Construction

Our Anonymous Privacy-Preserving Scheme (APPS) utilizes the MapReduce framework to achieve flexibility and efficiency. To understand the construction model, basic information about Bilinear Map needs to be addressed.

Definition 5 *Let $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_3$ be multiplicative cyclic groups of order p. Suppose g_1 is the generator of $\mathbb{G}_1$, and g_2 is the generator of $\mathbb{G}_2$. Then, a bilinear map e is defined as $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_3$ and satisfies the bilinearity, non-degeneracy, and computability properties.*

Each property is defined as follows:

- Bilinearity: $\forall\ u \in \mathbb{G}_1$, $v \in \mathbb{G}_2$, and $a, b \in \mathbb{Z}_p$, then $e(u^a, v^b) = e(u, v)^{ab}$.
- Non-degeneracy: $\forall\ g_i\ \exists\ e \Rightarrow e(g_1, g_2, ..., g_i) \neq 1$.
- Computability: $\forall\ u, v \in \mathbb{G}_1$ and $\mathbb{G}_2$ respectively, and $a, b \in \mathbb{Z}_p$. $\exists\ \mathbb{A}\ s.t\ \mathbb{A} \Rightarrow e$

The APPS construction is illustrated as follows:

– **Setup:** The setup stage is executed by the data owners. A bilinear group $\mathbb{G}$ is selected with order p and a generator g. two integers (a, $b \in \mathbb{Z}_p$) are randomly selected and a random oracle is defined using the hash function ($H : \{0,1\}^* \rightarrow \mathbb{G}$). The public and master keys are generated according to the aforesaid setup, where $P_k = (\mathbb{G}, H(.), g, h = g^b, e(g,g)^a)$ and $MS_k = (b, g^a)$.

– **KeyGen(I_{key}, MS_k).** The *KeyGen* algorithm is run for each group of policy attributes (I_{key}). Two random integers are selected (r, r_j) for each attribute j and compute the private key $S_k = g^{\frac{a+r}{b}}, g^r.H(j)^{r_j}, g^{r_j}$.

– **MEKGen**. A randomly, an integer (s) and 1-degree polynomial $R(.)$ are selected, where $R(0) = s$. Produces n splits on s by randomly selecting $s_1, s_2, \ldots, s_n$, where $s = s_1 + s_2 + \cdots + s_n$. Then, the values of MEK are assigned to the values of the s splits ($MEK_i = s_i$, for $i = 1, 2, \ldots, n$).

– **ENC(I_{enc}, MEK)**. The encryption is executed by the MapReduce framework. The encryption policy attributes (I_{enc}) and MEK are uploaded to the cloud, each to a distinct mapper node. Each mapper uses the corresponding MEK to generate the CT_i. Where, for each data part the output is $Map(I_{enc}, MEK_i) \rightarrow (I_{enc}, CT_{mapper_i})$, which maps between the MEKs and the CTs.

– **DEC(CT, S_k).** When certain data are requested from a user, the desired part of data is retrieved, i.e., (this is done by the set of attributes and data parts pairs, and examined for integrity using the SHA-512 hash function. The functional encryption is applied to the data part, according to the type of requested information, e.g., the user ask only for names and birthdays. Where the retrieved message is retrieved by computing Equation (8.1),

$$m = \frac{e(g,g)^{as}}{\frac{e(h^s, g^{\frac{a+r}{b}})}{e(g,g)^{rs}}}. \tag{8.1}$$

8.5. Results and Discussion

8.5.1. Deployment Environment

We deployed our design based on ChameleonCloud environment [26]. ChameleonCloud is a configurable experimental environment for large-scale cloud research at the University of Chicago and the University of Texas at Austin. The structure of the deployment environment is depicted in Figure 8.5. A Kernal-based Virtual Machine (KVM) virtualization is installed on top of Linux (Ubuntu LTS 16.0.4) operating system and the reserved hardware. Hadoop is installed via OpenStack to facilitate the intensive MapReduce operations.

8.5.2. Experimental Test

To test the validity of our approach, the ChameleonCloud environment is utilized with the following setup:

1. We generate synthesized datasets with different size.
2. Utilize Hadoop framework by deploying the Hadoop Distributed File System (HDFS) appliance that is provided by the ChameleonCloud environment.
3. The OpenStack environment is used to manage the virtual machine environment and resource scheduling.
4. The KVM virtualization software is used to provide unified storage resources. This is provided by deploying the MVAPICH appliance that is provided by Chameleon, which is responsible for MPI clustering of the KVM virtual machine with InfiniBand enabled.
5. Utilize the Ubuntu 16.04 LTS appliance, which is supported by Chameleon.

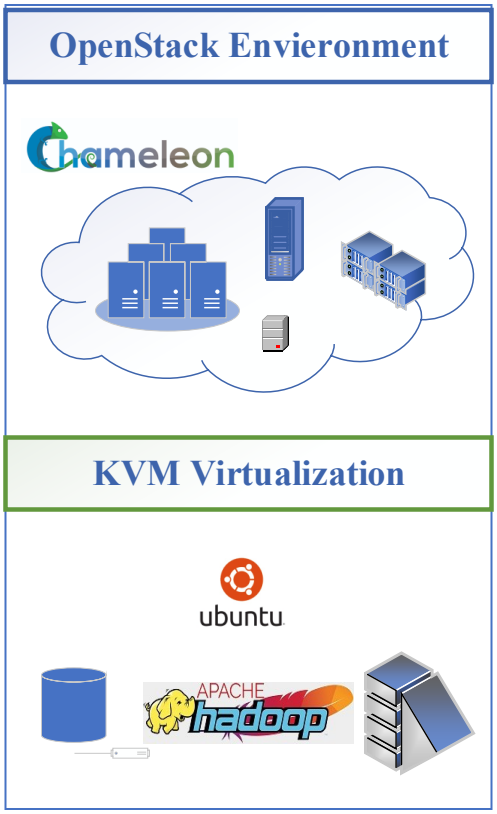

Figure 8.5. Deployment environment structure

6. The size of the synthesized datasets is ranging from 200 MB to 4.0 GB. The execution time is measured to evaluate system efficiency.

The proposed design is compared with the centralized anonymous approach by Mohammed *et al.* and privacy scheme by Zhang *et al.* Figure 8.6 shows the comparison between our proposal and the other approaches. In our design, we included higher data sizes than the data presented in the other works. The centralized approach shows a good start in the execution time with lower size, but a significant increase in the execution time when the dataset's size slightly increase. Approximately, a linear increase in the execution time for our proposal and the work of Zhang *et al.* with a noticeable advantage of our work in the analogous range of dataset's sizes. Furthermore, our work has been tested on higher synthesized datasets' size than the other works.

8.5.3. Final Remarks

The security of the proposed scheme is analyzed according to the following threats.

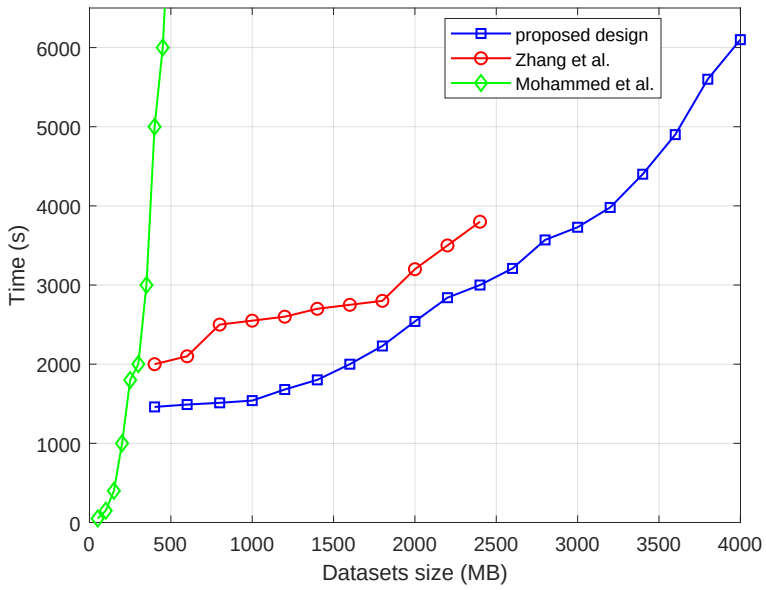

Figure 8.6. Execution time of the proposed design and other works

1. Data Integrity. The APPS model is secure against data integrity and authenticity issues. The secure hash algorithm SHA-512 ensures that the data are not tampered or modified during storage or transmission.

2. Privacy-Preserving. The APPS model guarantees the privacy of the uploaded data. Furthermore, if one of the mapper nodes is curious, the model still able to preserve the privacy of the uploaded data as well as the data owners identities. This is held according to the Bilinear maps assumption.

3. The Encryption/Decryption time is reduced significantly, as shown in the experimental results. Moreover, the FE showed the ability to decrypt part of data in case of a specific type of information is required.

8.6. Conclusions

This chapter presented an Anonymous Privacy-Preserving Scheme (APPS) for big data over the cloud. The proposed design employed the SHA-512 hash function, the functional encryption algorithm, and the MapReduce framework. Through two levels of MapReduce, the proposed design enhance the encryption/decryption task of big data. The employment of the functional encryption helped to decrypt the desired part of data rather than all data, which saves time and efforts. The

Integrity of the data is conserved using the SHA-512 hash function and the privacy of the uploaded data is conserved using the anonymous MapReduce framework.

In the future, further experiments will be conducted to evaluate the performance of the proposed scheme. More security issues and threats will be analyzed in the future.

8.7. References

[1] Chaowei Yang, Qunying Huang, Zhenlong Li, Kai Liu, and Fei Hu. "Big Data and cloud computing: innovation opportunities and challenges". In: *International Journal of Digital Earth* 10.1 (2017), pp. 13–53.

[2] Fang Liu et al. "NIST cloud computing reference architecture". In: *NIST special publication* 500.2011 (2011), pp. 1–28.

[3] Min Chen, Shiwen Mao, and Yunhao Liu. "Big data: A survey". In: *Mobile networks and applications* 19.2 (2014), pp. 171–209.

[4] Jens Dittrich and Jorge-Arnulfo Quiané-Ruiz. "Efficient big data processing in Hadoop MapReduce". In: *Proceedings of the VLDB Endowment* 5.12 (2012), pp. 2014–2015.

[5] Ning Cao, Cong Wang, Ming Li, Kui Ren, and Wenjing Lou. "Privacy-preserving multi-keyword ranked search over encrypted cloud data". In: *IEEE Transactions on parallel and distributed systems* 25.1 (2013), pp. 222–233.

[6] Surajit Chaudhuri. "What next?: a half-dozen data management research goals for big data and the cloud". In: *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI symposium on Principles of Database Systems*. ACM. 2012, pp. 1–4.

[7] K Wang, R Chen, BC Fung, and PS Yu. "Privacy-preserving data publishing: A survey on recent developments". In: *ACM Computing Surveys* (2010).

[8] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters". In: *Communications of the ACM* 51.1 (2008), pp. 107–113.

[9] M Prakash and G Singaravel. "An approach for prevention of privacy breach and information leakage in sensitive data mining". In: *Computers & Electrical Engineering* 45 (2015), pp. 134–140.

[10] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: a flexible data processing tool". In: *Communications of the ACM* 53.1 (2010), pp. 72–77.

[11] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters". In: *OSDI'04: Sixth Symposium on Operating System Design and Implementation*. San Francisco, CA, 2004, pp. 137–150.

[12] Dan Boneh, Amit Sahai, and Brent Waters. "Functional encryption: Definitions and challenges". In: *Theory of Cryptography Conference*. Springer. 2011, pp. 253–273.

[13] Ivan Bjerre Damgård. "A Design Principle for Hash Functions". In: *Advances in Cryptology — CRYPTO' 89 Proceedings*. Ed. by Gilles Brassard. New York, NY: Springer New York, 1990, pp. 416–427.

[14] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. "Sponge functions". In: *ECRYPT hash workshop*. Vol. 2007. 9. Citeseer. 2007.

[15] FIPS PUB. "Secure hash standard (shs)". In: *FIPS PUB 180* 4 (2012), pp. 1–27.

[16] Zeyad Al-Odat, Mazhar Ali, and Samee U Khan. "Mitigation and Improving SHA-1 Standard Using Collision Detection Approach". In: *2018 International Conference on Frontiers of Information Technology (FIT)*. IEEE. 2018, pp. 333–338.

[17] Dhasarathan Chandramohan, Thirumal Vengattaraman, and Ponnurangam Dhavachelvan. "A secure data privacy preservation for on-demand cloud service". In: *Journal of King Saud University-Engineering Sciences* 29.2 (2017), pp. 144–150.

[18] Yousra Abdul Alsahib S Aldeen, Mazleena Salleh, and Yazan Aljeroudi. "An innovative privacy preserving technique for incremental datasets on cloud computing". In: *Journal of biomedical informatics* 62 (2016), pp. 107–116.

[19] A Tawalbeh Lo'ai and Gokay Saldamli. "Reconsidering Big Data Security and Privacy in Cloud and Mobile Cloud Systems". In: *Journal of King Saud University-Computer and Information Sciences* (2019).

[20] Priya P Sharma and Chandrakant P Navdeti. "Securing big data hadoop: a review of security issues, threats and solution". In: *Int. J. Comput. Sci. Inf. Technol* 5.2 (2014), pp. 2126–2131.

[21] Jingwei Li, Chunfu Jia, Jin Li, and Xiaofeng Chen. "Outsourcing encryption of attribute-based encryption with mapreduce". In: *International Conference on Information and Communications Security*. Springer. 2012, pp. 191–201.

[22] Dan Wang, Bing Guo, Yan Shen, Shun-Jun Cheng, and Yong-Hong Lin. "A faster fully homomorphic encryption scheme in big data". In: *2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)(*. IEEE. 2017, pp. 345–349.

[23] Xuyun Zhang, Chang Liu, Surya Nepal, Chi Yang, and Jinjun Chen. "Privacy preservation over big data in cloud systems". In: *Security, Privacy and Trust in Cloud Systems*. Springer, 2014, pp. 239–257.

[24] Noman Mohammed, Benjamin Fung, Patrick CK Hung, and Cheuk-Kwong Lee. "Centralized and distributed anonymization for high-dimensional healthcare data". In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 4.4 (2010), p. 18.

[25] Priyank Jain, Manasi Gyanchandani, and Nilay Khare. "Enhanced Secured Map Reduce layer for Big Data privacy and security". In: *Journal of Big Data* 6.1 (2019), p. 30.

[26] *ChameleonCloud*. [Online; accessed Aug. 2019]. Aug. 2019. URL: `https://www.chameleoncloud.org`.

9. AN EFFICIENT CLOUD AUDITING SCHEME FOR DATA INTEGRITY AND IDENTITY PRIVACY OF MULTIPLE UPLOADERS

9.1. Introduction

Cloud computing provides an efficient and wide range of services over scalable storage locations. These services include data sharing between multiple users, where each user can edit the existing data or upload a new one.

The data in the cloud are susceptible to lose or damage due to software or hardware failures of the cloud service provider (CSP) [1]. The traditional way to verify the existence of data is to retrieve the entire file from the cloud and check the data signature or hash using the conventional cryptography approaches. However, the verification process becomes harder because retrieving large data file consumes time and efforts [2].

Provable data possession is the method of verifying that the cloud server still possesses the stored data [3]. This includes traditional or other verification methods. The need to verify the integrity of data over the cloud becomes essential because cloud service providers offer the computation services of cloud data files directly without download. Therefore, many techniques were proposed to allow the data owners or a hired third party to verify the existence of data without any need for data download [4].

In some cases, e.g., election and top-secret reports, the data owners delegate the process of results verification to a Third Party Auditor (TPA). But, when the TPA reveals the identity of each member of the data owners, then a risk of data confidentiality will arise [5]. Therefore, the TPA is only allowed to check the correctness and integrity of data without any knowledge about the identity of the owners. Moreover, the stored data might be lost due to software/hardware failures or an adversary tries to corrupt users' data. The CSP tries to deceive the data owners or the TPAs

The content of this chapter has been published in the IEEE Cloud Summit 2019 the 3rd IEEE International Conference on Cloud and Fog Computing Technologies and Applications. The material in this chapter was co-authored by Zeyad Al-Odat and Samee Khan. Zeyad Al-Odat had primary responsibility for conducting experiments and collecting results. Zeyad Al-Odat was the primary developer of the conclusions that are advanced here. Zeyad Al-Odat also drafted and revised all versions of this chapter. Samee Khan drafted, revised all versions of this chapter, and served as proofreader.

by a fake report about the existence of their data [6]. Therefore, a complete auditing scheme has to be able to limit the TPA privileges and provide an authentic tool to check data integrity.

This chapter provides a Privacy-Preserving scheme of Multiple data Uploaders on the Cloud (PPMUC). The proposed work presents an efficient data verification scheme to check the integrity and existence of stored data over the cloud. Moreover, the PPMUC hides the data owners identities from being revealed to the TPA.

The rest of chapter is organized as follows. Section 2 gives preliminary information related to our scheme. Section 3 provides a literature review of the related works. The proposed methodology is presented in section 4. Results and Discussion in section 5. Section 6 concludes the chapter.

9.2. Preliminaries

Before going through the details of our proposal, brief descriptions about the system components, the general structure of the TPA model, and the threat model need to be addressed.

9.2.1. System Components

There are three main components in the TPA scheme which construct the verification system of shared data on the cloud.

1. Cloud Service Provider (CSP). The CSP provides the storage locations for the shared data files and responds to the TPA queries (challenges). Each data block is stored as a tuple of three values (index, block (m), and tag). The index represents the message identifier, the block field represents the block's content, and tag contains the signature of the block creator.

2. Data Uploader. The uploader is any group member of the data owners that has a secret key to sign his corresponding block. Then, the corresponding block content, index, and tag are uploaded to the shared storage location.

3. TPA. The verification of the existence of data in the cloud is given to the TPA. The TPA performs the verification process by sending a challenging command to the CSP. The validity of the data integrity is checked according to the received response from the CSP.

9.2.2. General Structure of the TPA

When data owners hire a TPA to audit their data, the general procedure is accomplished according to Figure 9.1. The figure shows the general structure of TPA scheme. The data owners

(Alice and Bob) have the same priority to create, upload, and modify the data blocks. Each user in the group signs the data block when any action is performed. The TPA verifies the existence of data by sending a *challenge* command to the CSP. The proof of data existence is determined according to the CSP *response* to the TPA *challenge* [7].

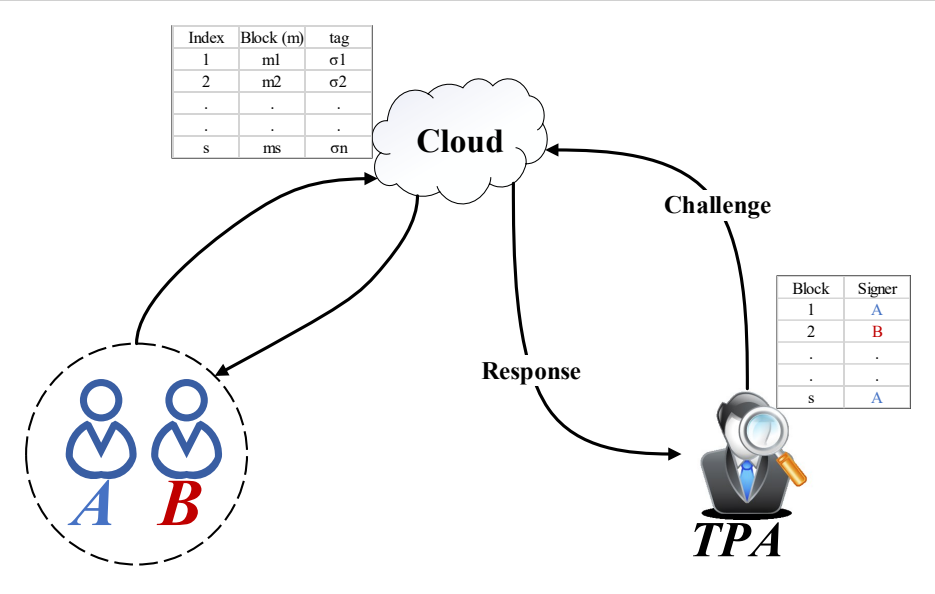

Figure 9.1. Alice and Bob share their data in the cloud, and the TPA verifies the existence of data.

However, the TPA can determine the identity of each block signer, which leads to the disclosure of the identity of the data uploaders.

9.2.3. Threat Model

Two major threats are related to this area of study.

1. Integrity Threat. The shared data in the cloud might be corrupted by an adversary, or the CSP loses the stored data due to software/hardware failure of the cloud system. In such a case, the CSP will be averse to inform the data owners about the data loss to avoid losing customers. Therefore, the CSP tries to deceive the TPA by forging the response report to provide a positive inquiry [8]. Moreover, a collision attack forms a serious threat when a weak secure hash algorithm is used [9].

2. Uploader Privacy Threat. The identity of the data signer is confidential and only related to the group of data owners. Therefore, during the verification process, the TPA is only allowed

to check the data integrity without knowing the identity of each data signer. Once the TPA reveals the signer of each data block, then it will be easy for the TPA to discriminate between the high and low-value users. Consequently, determine the high confidential data from the level of its corresponding signer [10].

9.3. Related Work

The idea of anonymous signature schemes was first introduced by Chaum and Heyst [11]. In their design, a trust group manager creates a group of users and assigns special secret keys distributed over the group members. Some users can create and sign messages using the secret keys on behalf of the group. A public verifier is unable to distinguish the identities of data signers. However, the group manager can revoke the identity of the signers.

In 2001 Rivest *et al.* came up with a *ring signature scheme* which generates a group of multiple users without a manager [12]. In this scheme, each user creates his own secret keys pair (public-private) and signs a message in such a way that no one in the group can determine his identity. However, in some cases, when a single user signs a message with his private key — without declaring a set of possible signers — then his identity might be revealed.

The anonymous signature scheme is applied to shared cloud data when multiple users work on shared data blocks. Each user has a distinct private-public key pair and the identity of each block signer (user) is anonymous from the others [13]. The idea of the group signature, where a group manager can trace the group members actions, was presented in [14]. The proposed work presents an anonymous and tractable group data sharing over the cloud. The authors employed the group signature scheme by Chaum and Heyst to implement their design. The experimental results showed an improvement in the verification time but showed a degradation in time when user identity is revoked. Using the same group signature approach, Li *et al.* employed the group signature for the privacy-preserving of mobile sensing data [15]. The proposed design divides the regions of cellular infrastructure into groups with each group contains a number of users. The group signature allows the manager of each group to determine the misbehaving user inside the group region.

However, the data uploaders over the cloud, in some cases, want to hide their identity from other users or entities. To hide the signer identities from all users and verifiers, Wang *et*

al. proposed a privacy scheme for public data auditing [16]. The proposed design provides a public data auditing over the cloud with an identity-privacy approach. The identities of the data uploaders are made hidden from both the group members and the third party auditors. In their design, they exploit the ring signature scheme to anonymously upload data blocks and verify data integrity over the cloud. Moreover, Wu *et al.* in [17] followed the same ring signature approach for privacy-preserving. An Anonymous Cloud Auditing scheme with Multiple Uploaders (ACAMU) was presented. The proposed design helps the data owners to hide their identity from a hired third-party auditor. A mathematical model was studied and prove the efficiency of their proposal. However, no experiment was conducted to show the applicability of their design. Moreover, to achieve the privacy-preserving of multiple data uploaders, Cong *et al.* proposed a scheme that combines the public key homomorphic authenticator with random masking. The proposed design achieves the privacy requirements of auditing and identity hiding [18].

In this chapter, we are introducing a Privacy-Preserving model for Multiple data Uploaders over the Cloud (*PPMUC*). In our design, we used a modified ring signature scheme to deploy our proposal, and the Secure Hash Algorithm (SHA-512) is used to consolidate the proposed design [19]. In the subsequent section, Details about the *PPMUC* will be presented.

9.4. Proposed Methodology

The *PPMUC* helps to verify the existence of data over the cloud through a hired TPA. Brief descriptions about the Bilinear Map and Bilinear Diffie-Hellman (*BDH*) are presented before going through the details of our proposal. The reader is advised to refer to Table 9.1, which lists the notations that are used in this section. Please refer to the table to clarify the meaning of each symbol.

9.4.1. Bilinear Map

To understand the *PPMUC*, the Bilinear Map mathematical notation and definition need to be addressed. The subsequent definitions identify the meaning and properties of the bilinear map and BDH. For more details about these definitions, the reader is advised to read [20, 21, 22].

Definition 6 *Let* $\mathbb{G}_1$*,* $\mathbb{G}_2$*, and* $\mathbb{G}_3$ *be multiplicative cyclic groups of order p. Suppose* g_1 *is the generator of* $\mathbb{G}_1$*, and* g_2 *is the generator of* $\mathbb{G}_2$*. Then, a bilinear map e is defined as* $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_3$ *and satisfies the bilinearity, non-degeneracy, and computability properties.*

Table 9.1. Notations of the PPMUC design

Symbols	Meaning
$\mathbb{A}$	Efficient algorithm **A**
$\forall$	For all
H	Hash value after applying the $SHA-512$
$\exists$	Exists
$\mathbb{Z}_p$	big prime p
$\mathbb{G}$	Group
e	bilinear map
θ	Tag
π	message response $L(0)$
CSP	Cloud Service Provider
Puk_n	public key of user n
Prk_n	private key of user n

Each property is defined as follows:

- Bilinearity: $\forall\ u \in \mathbb{G}_1$, $v \in \mathbb{G}_2$, and $a, b \in \mathbb{Z}_p$, then $e(u^a, v^b) = e(u, v)^{ab}$.
- Non-degeneracy: $\forall\ g_i\ \exists\ e \Rightarrow e(g_1, g_2, ..., g_i) \neq 1$.
- Computability: $\forall\ u, v \in \mathbb{G}_1$ and $\mathbb{G}_2$ respectively, and $a, b \in \mathbb{Z}_p$. $\exists\ \mathbb{A}\ s.t\ \mathbb{A} \Rightarrow e$

9.4.2. Bilinear Diffie-Hellman (BDH) Assumption

Definition 7 ***BDH Problem.*** *The advantage (ϵ) of Algorithm $\mathbb{A}$ in solving Discrete Logarithm (DL) problem for $\mathbb{G}_1$ of the bilinear map e is defined as:*

$$\epsilon \leq Pr[A(g, g^a, g^b) \leftarrow \mathbb{A}(g, g^a, g^b).$$

Then, with a negligible advantage, $\mathbb{A}$ satisfies the BDH assumption.

9.4.3. PPMUC Model

The *PPMUC* model consists of six functions: create, generate, sign, challenge, response, and verify. Each function performs designated operations to fulfill the security requirements and protect the system against any possible threat.

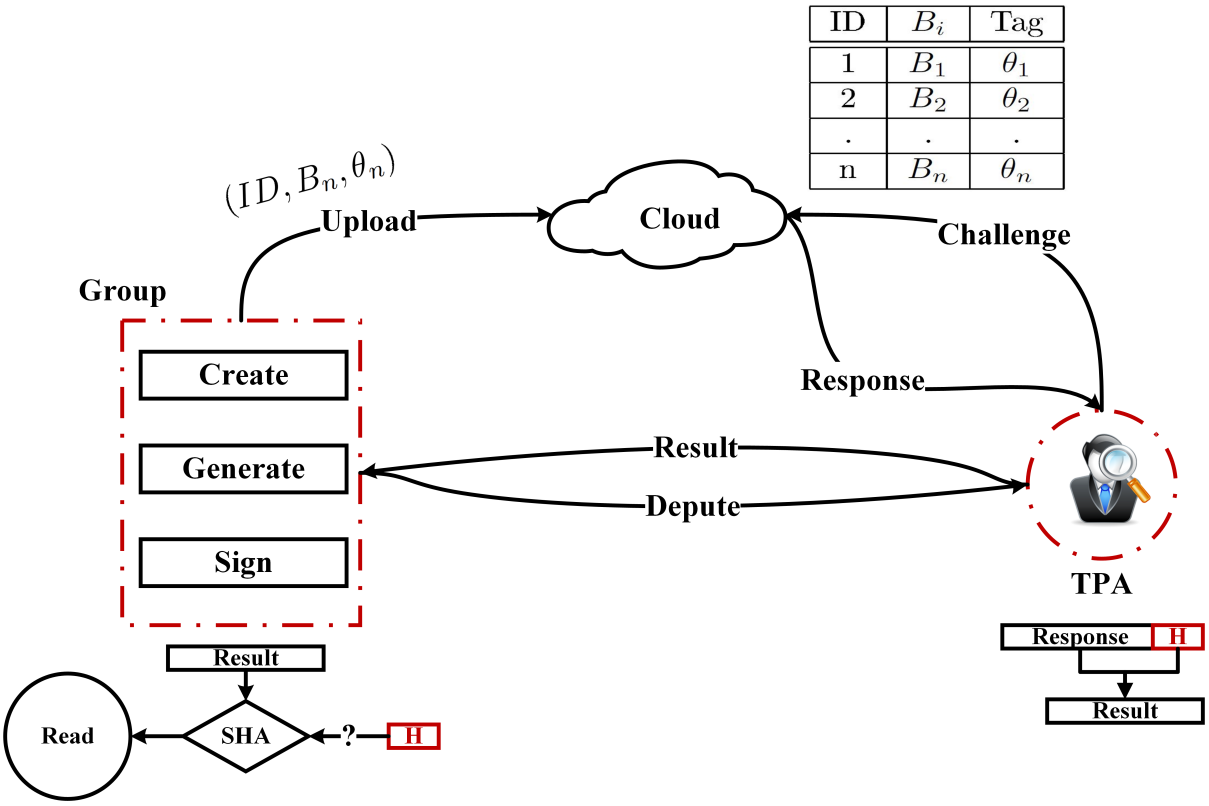

Figure 9.2. General architecture of the PPMUC scheme

1. Create. This function is responsible for the group creation where n group members are added to the group.

2. Generate. Each member of the group generates a pair of public and private keys (Puk_n, Prk_n) that is assigned to the n^{th} user.

3. Sign. The n^{th} uploader uses his corresponding private key (Prk_n) to calculate the tag $(\theta_{n,m})$ of the n^{th} block (B_n), and uploads the tuple $(ID, B_n, \theta_{n,m})$ to the cloud.

4. Challenge. When the group members want to audit their data, they send all public keys of the group members along with randomly picked block indices to the TPA. Then, the TPA generates the challenge and sends it to the CSP.

5. Response. Once the CSP receives the challenge from the TPA, the response (π, θ) is prepared and sent back to the TPA.

6. Verify. The existence of the data is verified by the TPA according to the CSP response. Then, the TPA prepares the verification report, signs it using the Secure Hash Algorithm (SHA-512), and sends it back to the Group members.

Figure 9.2 shows the general structure of the proposed scheme. In this design, all group members have the same priority without group manager. Each user creates his own keys pair (Pub,Prk), which is used to generate a block tag (θ). Afterward, the tuple (ID, B_n, θ_n) is uploaded to the cloud for sharing and storage.

When the group members want to verify their data, they send a "depute" request to the TPA which contains all users public keys (Puk_n)[1] and a subset of randomly picked block indices (Q). Then, the TPA prepares the "challenge" order and sends it to the CSP.

The CSP receives the challenge request from the TPA and starts to compute the response using the blocks (B_n) and their corresponding tags (θ_n), where $n \in Q$. Once completed, the CSP sends the response back to the TPA as (π, θ^*) pair, where π refers to the blocks calculations and θ^* refers to the tags' calculations. The TPA verifies the correctness of the received response by verifying the following:

$$Result \stackrel{?}{\leftarrow} Verify(\pi, \theta^*, Puk_n, Q),$$

where *result* is the output of the verification process that is conducted by the TPA.

In case of a collision attack by an adversary ($\mathbb{A}$), the TPA uses the Secure Hash Algorithm (SHA-512) to sign the result before sending it to the Group. On receiving the result report, the group members check the correctness of the received report by computing the SHA-512 value of the report. Once verified, they read the query report about their data integrity and existence.

9.4.3.1. PPMUC Procedure

The PPMUC process is accomplished according to the following setup:

- A user u_i, who wants to upload the data block (B_j), selects a random value (x_i). This value is used to generate the public-private pair (Prk_i, puk_i), where $x_i = Prk_i$.

[1]Public keys give no clue about the identity of their corresponding users.

- User u_i picks a one time random value b_t for all group members, where $t \neq i$, and assigns $\theta_t = g_1^{b_t}$. Afterward, user U_i signs a message block (B_j) using (9.1).

$$\theta_{intendi,j} = \left(\frac{\Gamma}{\prod_{t \neq i} Puk_t^{b_t}} \right)^{1/x_i}, \tag{9.1}$$

  where $\Gamma = H(j).g_1^{B_j}$, and H is the secure hash algorithm function. After the completion of block generation and sign, the index (j), block (B_j), and tag ($\theta_{i,j}$) are combined as one tuple and uploaded to the cloud for sharing.

- To check the integrity and existence of the data, the TPA sends a challenge request to the CSP. This request contains the public keys of the group members and a set of randomly picked block indices (Puk_i, Q).

- At the server side, the CSP computes the response using (9.2) and (9.3).

$$\pi = \sum_{j \in Q} j.B_j \tag{9.2}$$

$$\theta^* = \prod_{i,j \in Q} e(\theta_{i,j}, Puk_i), \tag{9.3}$$

  where e is the bilinear map between θ and Puk. When the response is ready, the CSP sends it to the TPA as (θ^*, π).

- The verification process is accomplished by the TPA after receiving the CSP's response. The TPA checks the correctness of the received response using (9.4).

$$\theta^* \stackrel{?}{=} e(\Gamma, Pub_k). \tag{9.4}$$

  The bilinear map (e) between the Γ and the Puk_k must be equal to the θ^* value.

- The TPA computes the (SHA-512) hash value for the result, appends it to the report, and sends it back to the client.

- The client (group) receives the report from the TPA. The report is verified first by computing the SHA-512 hash value and compare it with the append hash value. Then, if the two hashes are equal, the report contains the correct auditing information about the data on the cloud.

9.5. Results and Discussion

The proposed design provides a secure infrastructure against three main security threats: 1. a cheat CSP, 2. an eager TPA, and 3. data integrity. The cheat CSP tries to deceive the TPA or group members about the existence of their data in case of hardware or software failure. An eager TPA tries to know the identity of each block signer to distinguish the highest value member. On another hand, the data integrity must be verified even though the existence of data is verified; because some adversaries want to corrupt the shared or stored data.

9.5.1. Security Model

For each security threat, we built a security model to verify the efficiency of the proposed design. This model includes mathematical proofs and experimental results. The mathematical proofs are deduced through the bilinear map, the BDH assumptions, and the results presented there within [20, 21, 22, 23]. The experiments were conducted to test the speed and efficiency of the proposed design.

9.5.1.1. Data Integrity

The probability of the adversary ($\mathbb{A}$) to deceive the TPA is negligible according to the BDH assumption. The adversary ($\mathbb{A}$) tries to win a challenge game according to the following setup:

1. $\mathbb{A}$ has all public keys, indices, and blocks (B_n).
2. $\mathbb{A}$ receives the set of selected indices query (Q).
3. $\mathbb{A}$ responds to the challenge by generating (π, θ^*) pair.

Deduction 1 *The **PPMUC** scheme is strong against CSP deception, because the probability that a fake response pass the verification process is negligible, s.t.,*

$$Pr_{\mathbb{A}}\left[\begin{array}{cc} & (ID, n, B_n) \leftarrow A \\ Verify(\pi, \theta^*, Puk_n, Q, S) & Q \leftarrow A \\ & (\pi, \theta^*) \leftarrow A(challenge) \end{array}\right] = \lambda,$$

where λ is a security parameter with a negligible probability. This is achieved according to the bilinear map assumptions, where the adversary ($\mathbb{A}$) has a negligible probability (λ) to deceive the TPA.

9.5.1.2. Privacy-Preserving

The second security issue is the identity of the data uploader. In some cases, an eager TPA (adversary B) tries to reveal the identity of the data uploader which might contain confidential information. The probability of B to reveal the identity of block signer is no more than $1/n$, for n group users.

Deduction 2 *The **PPMUC** scheme attains the identity privacy property if for a polynomial-time adversary ($\mathbb{B}$) the probability of adversary advantage is negligible (no more than $1/n$), s.t.,*

$$Pr_{\mathbb{B}}\begin{bmatrix} & (Puk_n, Q) \leftarrow B \\ n = n^{'} & (1, 2, .., n) \leftarrow B \\ & (\pi, \theta^*) \leftarrow Response \\ & n^{'} \leftarrow B(\pi, \theta^*) \end{bmatrix} \leq 1/n$$

According to this deduction, the probability that an adversary can determine the highest value group member will not exceed the equally distributed probability ($1/n$). This property is guaranteed by the BDH assumption [22].

9.5.1.3. Experimental Results

To test our design, we prepared a set of group users and auditing tasks. The experiments were conducted on a Linux operating system with 3.4 GHz i_7 CPU and 16GB of RAM. We tested our design on a 3GB of synthesized data for two parameters, key generation time and auditing time. Figure 9.3 shows the experimental results of creation time after applying our design. The creation time increases with the number of users in the group. This is because the number of parameters increases with each new user, which requires more computations every time a key and tag are created. On the other hand, we prepared different auditing tasks to verify the existence of the data using the proposed work. Figure 9.4 shows the time needed to verify randomly chosen blocks of data. During all experiments, the required time for verification stayed approximately the same, because the number of parameters between the server and the TPA is the same.

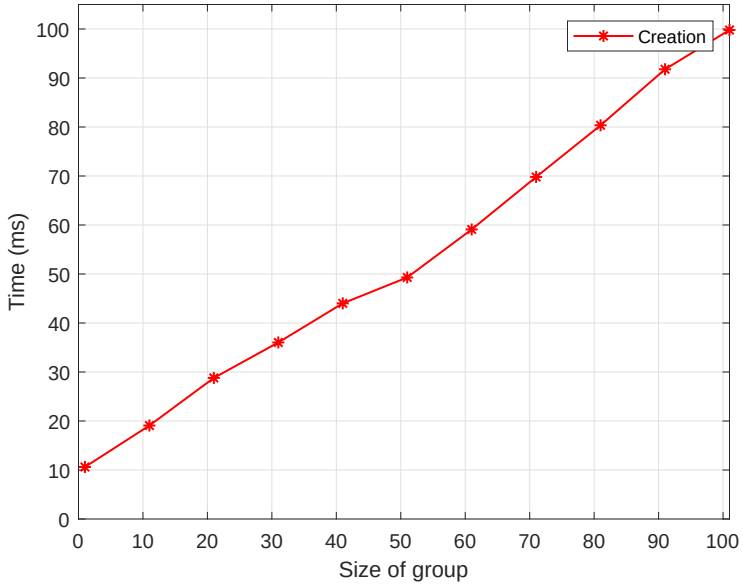

Figure 9.3. Creation time

9.5.1.4. Comparison with Other Schemes

The proposed design is compared with other works with respect to speed —which includes the time needed to create a group and its corresponding signatures— and the time needed to accomplish auditing tasks. Table 9.2 shows a comparison between our proposal and related work from the literature; our proposal showed the best results with respect to the creation time and auditing time. The results in the table correspond to a group of 20 members and 20 auditing tasks. However, the result of reference [18] is reported for one task.

Table 9.2. Comparison with other works for 20 tasks

Work	Group creation time (ms)	Auditing time (ms)
[16]	52	2500
[18]	*	430/task
Proposed	30	1000

* Unreported result.

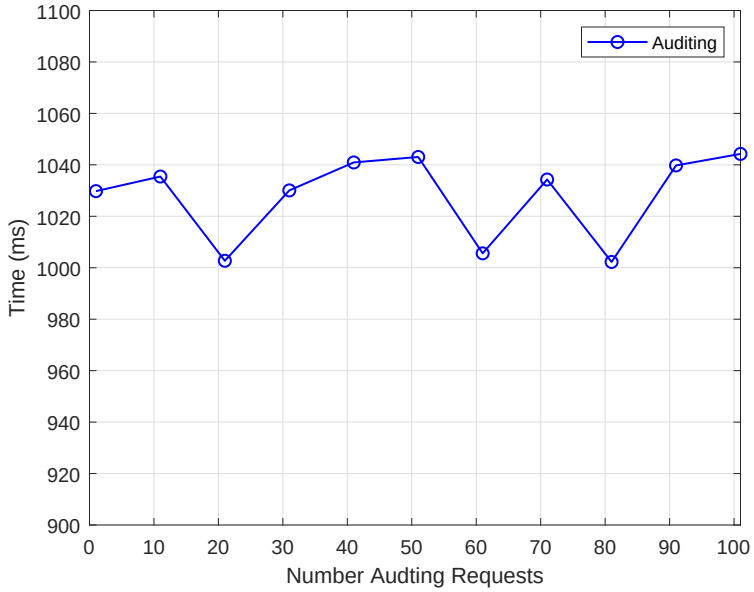

Figure 9.4. Auditing time

Moreover, our design produces a small tag size for the uploaded data blocks when compared with the other works. This will reduce the storage needed for the verification, and consequently reduces the price of renting a storage from the CSP.

9.6. Conclusions

This chapter introduced a privacy-preserving scheme for multiple data uploaders over the cloud. The proposed design helps to overcome two security issues of shared data, which are the data integrity and identity privacy. Using this approach, the public verifiers are not able to revoke the identity of message block signer, and a CSP is unable to cheat the TPA in the case of data loss or corruption.

9.7. References

[1] Keiko Hashizume, David G Rosado, Eduardo Fernández-Medina, and Eduardo B Fernandez. "An analysis of security issues for cloud computing". In: *Journal of internet services and applications* 4.1 (2013), p. 5.

[2] Muhammad Imran et al. "Provenance based data integrity checking and verification in cloud environments". In: *PloS one* 12.5 (2017), e0177576.

[3] Giuseppe Ateniese et al. “Remote data checking using provable data possession”. In: *ACM Transactions on Information and System Security (TISSEC)* 14.1 (2011), p. 12.

[4] Sura Khalil Abd et al. “Cloud computing security risks with authorization access for secure multi-tenancy based on AAAS protocol”. In: *TENCON 2015-2015 IEEE Region 10 Conference*. IEEE. 2015, pp. 1–5.

[5] Robbie Simpson and Tim Storer. “Third-party verifiable voting systems: Addressing motivation and incentives in e-voting”. In: *Journal of information security and applications* 38 (2018), pp. 132–138.

[6] Sutirtha Chakraborty, Shubham Singh, and Surmila Thokchom. “Integrity checking using third party auditor in cloud storage”. In: *2018 Eleventh International Conference on Contemporary Computing (IC3)*. IEEE. 2018, pp. 1–6.

[7] Kochumol Abraham and Win Mathew John. “Proving possession and retrievability within a cloud environment: A comparative survey”. In: *Int. J. Comput. Sci. Inf. Technol.* 5.1 (2014), pp. 478–485.

[8] Yun Xue Yan, Lei Wu, Wen Yu Xu, Hao Wang, and Zhao Man Liu. “Integrity Audit of Shared Cloud Data with Identity Tracking”. In: *Security and Communication Networks* 2019 (2019). doi.org/10.1155/2019/1354346, pp. 1–11.

[9] Zeyad Al-Odat, Mazhar Ali, and Samee U Khan. “Mitigation and Improving SHA-1 Standard Using Collision Detection Approach”. In: *2018 International Conference on Frontiers of Information Technology (FIT)*. IEEE. 2018, pp. 333–338.

[10] Shareeful Islam, Moussa Ouedraogo, Christos Kalloniatis, Haralambos Mouratidis, and Stefanos Gritzalis. “Assurance of security and privacy requirements for cloud deployment models”. In: *IEEE Transactions on Cloud Computing* 6.2 (2018), pp. 387–400.

[11] David Chaum and Eugène Van Heyst. “Group signatures”. In: *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer. 1991, pp. 257–265.

[12] Ronald L Rivest, Adi Shamir, and Yael Tauman. “How to leak a secret”. In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2001, pp. 552–565.

[13] Boyang Wang, Sherman SM Chow, Ming Li, and Hui Li. "Storing shared data on the cloud via security-mediator". In: *2013 IEEE 33rd International Conference on Distributed Computing Systems*. IEEE. 2013, pp. 124–133.

[14] Jian Shen, Tianqi Zhou, Xiaofeng Chen, Jin Li, and Willy Susilo. "Anonymous and traceable group data sharing in cloud computing". In: *IEEE Transactions on Information Forensics and Security* 13.4 (2018), pp. 912–925.

[15] Ya-Cheng Li and Shin-Ming Cheng. "Privacy Preserved Mobile Sensing Using Region-Based Group Signature". In: *IEEE Access* 6 (2018), pp. 61556–61568.

[16] Boyang Wang, Baochun Li, and Hui Li. "Oruta: Privacy-preserving public auditing for shared data in the cloud". In: *IEEE transactions on cloud computing* 2.1 (2014), pp. 43–56.

[17] Ge Wu, Yi Mu, Willy Susilo, and Fuchun Guo. "Privacy-preserving cloud auditing with multiple uploaders". In: *International Conference on Information Security Practice and Experience*. Springer. 2016, pp. 224–237.

[18] C. Wang, Q. Wang, K. Ren, and W. Lou. "Privacy-Preserving Public Auditing for Data Storage Security in Cloud Computing". In: *2010 Proceedings IEEE INFOCOM*. Mar. 2010, pp. 1–9. DOI: 10.1109/INFCOM.2010.5462173.

[19] Zeyad Al-Odat and Samee Khan. "The Sponge Structure Modulation Application to Overcome the Security Breaches for the MD5 and SHA-1 Hash Functions". In: *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*. Vol. 1. IEEE. 2019, pp. 811–816.

[20] Neal Koblitz and Alfred Menezes. "Pairing-based cryptography at high security levels". In: *IMA International Conference on Cryptography and Coding*. Springer. 2005, pp. 13–36.

[21] Bert den Boer. "Diffie-Hellman is as Strong as Discrete Log for Certain Primes". In: *Advances in Cryptology — CRYPTO' 88*. Ed. by Shafi Goldwasser. New York, NY: Springer New York, 1990, pp. 530–539.

[22] Dan Boneh. "The decision diffie-hellman problem". In: *International Algorithmic Number Theory Symposium*. Springer. 1998, pp. 48–63.

[23] Jan Camenisch and Anna Lysyanskaya. “Signature schemes and anonymous credentials from bilinear maps”. In: *Annual International Cryptology Conference*. Springer. 2004, pp. 56–72.

10. CONCLUSIONS AND FUTURE WORK

10.1. Conclusions

This thesis focused on secure hash algorithms and their corresponding security issues.

In Chapter 3 we presented two methods to improve the SHA-1 standard against collision attack. The first design relies on Stevens's approach for detecting SHA-1 collision attack, in which the input message is checked against collision possibility according to three values $(\delta W_s, \delta B_i, i)$. These values belong to the previously published works of disturbance vectors that lead to a collision. After each round, the system is checked for the aforesaid three values, then they are used to extract the sibling message from the given one. Finally, compare IHV_{k+1}, IHV'_{k+1} of both messages if they were equal then the original message is crafted with collision forgery. The second approach is based on two blocks collision and the backward expansion calculation equation. The initial hash value (IHV_0) is processed using 80 steps SHA-1 function. Then applying backward expansion equation to get IHV'_0. For the messages that crafted with collision attack, both IHV_0 and IHV'_0 will be equal. The truncated SHA-512/160 is suggested to replace suspicious message's hash outputs.

Chapter 4 presented a Sponge structure modulation for the MD5 and SHA-1 is presented. The proposed design helps to solve the weaknesses of the MD5 and SHA-1 against security breaches. We investigated our proposal toward collision and length extension attacks. The results showed that the proposed design is resistant to the aforesaid attacks. The strength of our design comes from the strength of the Sponge structure, where the internal permutation function manipulates the data many times with five different steps $(\theta, \rho, \pi, \chi, \iota)$.

In Chapter 5, an improved version of the SHA-1 and SHA-2 hash function is presented. The proposed design consolidates the SHA-1 and SHA-2 against collision and length extension attacks. The proposed design preserves the general properties of the SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512 hash functions., e.g., the hash length. The design employs the function manipulators of the SHA-1 and SHA-2 hash standards, which add more randomness to the proposed design and strengthen it against collision and length extension attacks. The testing results proved the efficiency of our proposal through different factors, which are avalanche effect, testing vectors,

hamming distance, and bit-hit. Moreover, our proposal is effective against collision and length extension attacks as shown in the provided examples.

Chapter 6 presented a model that examines the randomness of the internal rounds of the SHA-1 and SHA-2 standards. A modified SHA design was proposed to improve the randomness of the internal rounds. The *Bayesian factor* and *odd ratio* tests were used to evaluate the hash functions. The CUDA platform was utilized to perform the experimental analyses, which was implemented with the help of Chameleon large-scale experiments testbed. The results showed that the Modified SHA produced more random rounds other than the other functions.

In Chapter 7, a secure storage scheme for distributed Big Data over the Cloud is presented. The proposed design employs the SHA-512 and SSS to ensure Big Data integrity and security. The experimental results show that the proposed design can handle a large data file with a reasonable time while preserving the security and authenticity properties. Moreover, the trust level between the cloud service provider and the client is increasing because of the assured data authorization. Big Data security is an open scope. In the future, further security analysis will be carried out, in particular, the deployment of secure hash algorithms in future designs.

Chapter 8 presented an Anonymous Privacy-Preserving Scheme (APPS) for big data over the cloud. The proposed design employed the SHA-512 hash function, the functional encryption algorithm, and the MapReduce framework. Through two levels of MapReduce, the proposed design enhance the encryption/decryption task of big data. The employment of the functional encryption helped to decrypt the desired part of data rather than all data, which saves time and effort. The Integrity of the data is conserved using the SHA-512 hash function and the privacy of the uploaded data is conserved using the anonymous MapReduce framework.

Chapter 9 introduced a privacy-preserving scheme for multiple data uploaders over the cloud. The proposed design helps to overcome two security issues of shared data, which are data integrity and identity privacy. Using this approach, the public verifiers are not able to revoke the identity of the message block signer, and a CSP is unable to cheat the TPA in the case of data loss or corruption.

10.2. Directions for Future Research

The scope of research is still open in the area of secure hash algorithms including attacks and applications. In this thesis, we offered several research topics that analyze security threats,

proper improvements, and applications related to big data over the cloud. Also, we offer some directions for future researches:

1. The collision detection scheme presented in Chapter 3. This approach is based on the construction of a differential-path using the main published disturbance vectors. This approach can be extended to include new disturbance vectors. On another hand, this work is suitable for the Merkle-Damgård structure hash standards. Therefore, if a new collision attack is announced for the SHA-2 standard, then the proposed approach is applicable.

2. The Sponge structure model in Chapter 4 can be extended to include other hash or cryptography functions because the sponge structure proved its strength against many security bugs that threaten cryptography functions.

3. The scope still open for future researches that aim to consolidate the hash standards against security breaches of the cryptographic hash functions. The security analyses of the SHA-3 hash function can be conducted and compared with the performance of other designs, as shown in Chapter 5.

4. The randomness analyses in Chapter 6 can be applied to the SHA-3 hash function (Keccak). Moreover, more experiments can be conducted to test the Avalanche and Difference analyses for the Secure Hash Algorithms.

5. Chapter 7, Chapter 8, and Chapter 9 presented the privacy-preserving scheme for data integrity and identity privacy. Further experiments need to be conducted to cover other possible threats. Moreover, an optimization regarding the speed and memory need to be studied. Moreover, the privacy of data over the cloud needs to be taken into consideration when a third party auditor present.

APPENDIX. LIST OF PUBLICATIONS

[1] **Zeyad A. Al-Odat**, Mazhar Ali, Assad Abbas, and Samee U. Kan. "Secure Hash Algorithms and the corresponding FPGA optimization Techniques". In: *ACM Computing Surveys* (2019).

[2] **Zeyad A. Al-Odat** and Samee U. Khan. "A Modified Secure Hash Algorithm Architecture to Circumvent Collision and Length Extension Attacks". In: *IEEE Transactions on Information Forensics and Security* (2019).

[3] **Zeyad A. Al-Odat**, Mazhar Ali, and Samee U Khan. "Mitigation and Improving SHA-1 Standard Using Collision Detection Approach". In: *2018 International Conference on Frontiers of Information Technology (FIT)*. IEEE. 2018, pp. 333–338.

[4] **Zeyad A. Al-Odat**, Assad Abbas, and Samee U Khan. "Randomness Analyses of the Secure Hash Algorithms, SHA-1, SHA-2 and Modified SHA". In: *2019 International Conference on Frontiers of Information Technology (FIT)*. IEEE. 2019.

[5] **Zeyad A. Al-Odat** and Samee U Khan. "Constructions and Attacks on Hash Functions". In: *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE. 2019.

[6] **Zeyad A. Al-Odat** and Samee U Khan. "Anonymous Privacy-Preserving Scheme for Big Data Over the Cloud". In: *2019 IEEE International Conference on Big Data (IEEE Big Data 2019)*. IEEE. 2019.

[7] **Zeyad A. Al-Odat** and Samee Khan. "The Sponge Structure Modulation Application to Overcome the Security Breaches for the MD5 and SHA-1 Hash Functions". In: *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*. 10.1109/COMPSAC.2019.00119. 2019.

[8] **Zeyad A. Al-Odat**, Eman Al-Qtiemat, and Samee Khan. "A Big Data Storage Scheme Based on Distributed Storage Locations and Multiple Authorizations". In: *2019 IEEE 5th International Conference on Big Data Security and Privacy (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing, (HPSC) and IEEE In-*

ternational Conference on Intelligent Data and Security. Vol. 5. 978-1-7281-0006-7/19. DOI 10.1109/BigDataSecurity/HPSC/IDS.2019.00014. 2019, pp. 13–18.

[9] **Zeyad A. Al-Odat** and Samee U Khan. "An Efficient Cloud Auditing Scheme for Data Integrity and Identity-Privacy of Multiple Uploaders". In: *IEEE Cloud Summit 2019 the 3rd IEEE International Conference on Cloud and Fog Computing Technologies and Applications*. Aug. 2019.

[10] **Zeyad A. Al-Odat**, Sudarshan K. Srinivasan, Sana Shuja, Eman Al-Qtiemat, and Mohana Asha Latha Dubasi. "IoT-Based Secure Embedded Scheme for Insulin Pump Data Acquisition and Monitoring". In: *CYBER 2018, The Third International Conference on Cyber-Technologies and Cyber-Systems*. Vol. 3. Cyper-2018. thinkmind.org. 2018, pp. 90–93.

[11] **Zeyad A. Al-Odat**, Sudarshan K. Srinivasan, Eman M. Al-Qtiemat, and Sana Shuja. "A Reliable IoT-Based Embedded Health Care System for Diabetic Patients". In: *International Journal On Advances in Internet Technology* 12.1&2 (2019).

[12] Eman M Al-Qtiemat, Sudarshan K Srinivasan, **Zeyad A. Al-Odat**, Mohana Asha Latha Dubasi, and Sana Shuja. "Synthesis of Formal Specifications From Requirements for Refinement-based Real Time Object Code Verification". In: *International Journal On Advances in Security* 12.1 & 2 (2019).

[13] **Zeyad A. Al-Odat**, Sudarshan K Srinivasan, Eman Al-qtiemat, and Sana Shuha. "A Secure Storage Scheme for Healthcare Data Over the Cloud Based on Multiple Authorizations". In: *The Fourth International Conference on Cyber-Technologies and Cyber-Systems (Cyber 2019)*. IARIA. 2019.

[14] Mohana Asha Latha Dubasi, Sudarshan K Srinivasan, Sana Shuja, and **Zeyad A. Al-Odat**. "Refinement Checker for Embedded Object Code Verification". In: *The Fourth International Conference on Cyber-Technologies and Cyber-Systems (Cyber 2019)*. IARIA. 2019.