

T.R.
GEBZE TECHNICAL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

**PERFORMANCE ANALYSIS OF CRYPTOGRAPHICAL HASH
FUNCTIONS ON ARM CORTEX MICROPROCESSOR**

TURGUT ORHAN
A THESIS SUBMITTED FOR THE DEGREE OF
MASTER OF SCIENCE
DEPARTMENT OF ELECTRONIC ENGINEERING

GEBZE
2020

T.R.
GEBZE TECHNICAL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

**PERFORMANCE ANALYSIS OF
CRYPTOGRAPHICAL HASH FUNCTIONS ON
ARM CORTEX MICROPROCESSOR**

TURGUT ORHAN
**A THESIS SUBMITTED FOR THE DEGREE OF
MASTER OF SCIENCE
DEPARTMENT OF ELECTRONIC ENGINEERING**

THESIS SUPERVISOR
ASSOC. PROF. DR. SERDAR SÜER ERDEM

GEBZE
2020

T.C.
GEBZE TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

KRİPTOGRAFİK HASH
FONKSİYONLARININ ARM KORTEKS
MİKROİŞLEMCİDE PERFORMANS
ANALİZİ

TURGUT ORHAN
YÜKSEK LİSANS TEZİ
ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

DANIŞMANI
DOÇ. DR. SERDAR SÜER ERDEM

GEBZE
2020



YÜKSEK LİSANS JÜRİ ONAY FORMU

GTÜ Fen Bilimleri Enstitüsü Yönetim Kurulu'nun 17/07/2020 tarih ve 2020/34 sayılı kararıyla oluşturulan jüri tarafından 28/07/2020 tarihinde tez savunma sınavı yapılan TURGUT ORHAN'ın tez çalışması ELEKTRONİK MÜHENDİSLİĞİ Anabilim Dalında YÜKSEK LİSANS tezi olarak kabul edilmiştir.

JÜRİ

ÜYE

(TEZ DANIŞMANI) : Doç. Dr. SERDAR SÜER ERDEM

ÜYE

: Dr. Öğr. Üyesi ERDİNÇ ÖZTÜRK

ÜYE

: Dr. Öğr. Üyesi KÖKSAL HOCAOĞLU

ONAY

Gebze Teknik Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu'nun
...../...../..... tarih ve/..... sayılı kararı.

İMZA/MÜHÜR

SUMMARY

In this work, on a STM32F407VG ARM microprocessor performance analysis of an input message that has various length was analyzed using SHA-1, SHA-2 and SHA-3 secure hash algorithms. Thirteen algorithms, one from SHA-1, six from SHA-2, and six from SHA3, were encoded in C language, and the performance of these algorithms were analyzed by running them on the ARM microprocessor. The 32-bit timer in the ARM processor is used to observe how long the algorithms process an input message and how long they can generate a fixed length of hash value. The process to obtain the hash value is handled step by step, after that the pseudo-codes are examined in detail then the coding process is started. In the next step, the codes were imported to the Keil uVision IDE that is used for programming of the ARM processor. Then, using a UART converter USB with a terminal program, time value is sent via serial communication from the microprocessor to the personal computer as a unit of milisecond. As a result of the study, the performance and complexity of all SHA algorithms were obtained.

Key Words: SHA1, SHA2, SHA3, ARM Microprocessor.

ÖZET

Bu çalışmada STM32F407VG ARM mikroişlemcisinde çeşitli uzunluktaki mesajların SHA-1, SHA-2 ve SHA-3 güvenli hash algoritmaları kullanılarak performans analizi incelenmiştir. SHA-1' den bir adet, SHA-2' den altı adet, SHA3' den de altı adet olmak üzere toplamda on üç adet algoritma C dili ile kodlandıktan sonra ARM mikroişlemcisinde koşturularak bu algoritmaların performansları analiz edilmiştir. Algoritmaların bilgi mesajını ne kadar sürede işledikleri ve sabit uzunluktaki hash değerini ne kadar sürede oluşturabildiklerini gözlemlemek için ARM işlemcideki 32 bitlik zamanlayıcı kullanılmıştır. Hash değerinin elde edilmesindeki süreçler adım adım ele alınmış ve sözde kodlar detaylı olarak incelendikten sonra kodlama işlemine geçilmiştir. Sonraki aşamada ise kodlar ARM işlemcinin programlanmasında kullanılan Keil uVision IDE' sine aktarılmıştır. Daha sonra ise UART dönüştürücü vasıtasıyla bir terminal programı kullanarak seri haberleşme ile mikro işlemciden kişisel bilgisayara performans sonucu birimi milisaniye olacak şekilde gönderilmektedir. Yapılan çalışma sonucunda ele alınan tüm SHA algoritmalarının birbirlerine karşı durumları ve karmaşıklık değerleri elde edilmiştir.

Anahtar Kelimeler: SHA1, SHA2, SHA3, ARM Mikroişlemci.

ACKNOWLEDGEMENTS

I would like to express my deep thankfulness to my supervisor, Associate Professor Dr. Serdar Suer ERDEM, who not only shares his scientific knowledge with me but also shows the path to the success. His suggestions and encouragement give me self-confidence to complete this work.

I am also grateful to my colleague at TUBITAK Gokhun ATAŞ for his support and valuable contribution to this study.

Finally, I want to thank my family, my wife Busra ÇAVUS ORHAN, who shows me the beauty of love and my son Muhammed Selim ORHAN, who shows me the immortal friendship.

TABLE of CONTENTS

	<u>Page</u>
SUMMARY	v
ÖZET	vi
ACKNOWLEDGEMENTS	vii
TABLE of CONTENTS	viii
LIST of ABBREVIATIONS and ACRONYMS	x
LIST of FIGURES	xi
LIST of TABLES	xiii
1. INTRODUCTION	1
2. CRYPTOGRAPHIC HASH FUNCTIONS	3
2.1. Applications of Hash Functions	4
2.1.1. Message Authentication	4
2.1.2. Digital Signatures	6
2.1.3. One Way Password File	8
2.1.4. Intrusion Detection	8
2.2. Requirements of Hash Functions	9
2.3. Types of Attacks to Secure Hash functions	11
2.3.1. Brute-Force Attacks	11
2.3.2. Cryptanalysis Attacks	12
3. DEFINITION SECURE HASH ALGORITHM	14
3.1. SHA-1	15
3.1.1. SHA-1 Functions	15
3.1.2. SHA-1 Constants	16
3.1.3. SHA-1 Preprocessing	16
3.1.4. SHA-1 Hash Computation	18
3.2. SHA-2	20
3.2.1. SHA-2 Functions	20
3.2.2. SHA-2 Constants	21
3.2.3. SHA-2 Preprocessing	22
3.2.4. SHA-2 Hash Computation	26

3.3. SHA-3 (Keccak)	30
3.3.1. Keccak-p Permutations	31
3.3.2. Step Mappings	38
3.3.3. Keccak – permutation [b, nr]	43
3.3.4. Sponge Construction	44
3.3.5. Keccak	46
4. EXPERIMENTAL RESULTS	49
4.1. ARM Cortex-4 Microprocessor	49
4.2. Performance Results	52
4.2.1. Rounds of SHA-1 and SHA-2	52
4.2.2. Comparison of SHA-1, SHA-2 and SHA-3	60
5. DISCUSSION AND CONCLUSION	64
REFERENCES	65
BIOGRAPHY	67

LIST of ABBREVIATIONS and ACRONYMS

Abbreviations Explanations

and

Acronyms

$\wedge$	:	Defines bitwise AND.
$\vee$	:	Defines bitwise OR.
$\oplus$	:	Defines bitwise OR.
$+$	:	Addition modulo 2^w .
$\ll$	:	Left-shift operation.
$\gg$	:	Right-shift operation.
$\sim$	:	Defines bitwise complement.
Bit	:	A binary digit 0 or 1.
Byte	:	A group of eight binary digit.
FIPS	:	Federal Information Processing Standard.
NIST	:	National Institute of Standards and Technology.
$\text{ROTL}^n(x)$	:	Defines rotate left $(x \ll n) \vee (x \gg w - n)$.
$\text{ROTR}^n(x)$	:	Defines rotate right $(x \gg n) \vee (x \ll w - n)$.
SHA	:	Secure Hash Algorithm.
$\text{SHR}^n(x)$	:	Defines right shift $(x \gg n)$.
Word	:	Consists of 32 bits (4 bytes) or 64 bits (8 bytes).
XOF	:	Extendable-Output Functions.

LIST of FIGURES

<u>Figure No:</u>	<u>Page</u>
2.1: General diagram of cryptographic hash function.	3
2.2: Message authentication application version A.	4
2.3: Message authentication application version B.	5
2.4: Message authentication application version C.	5
2.5: Message authentication application version D.	6
2.6: Digital signature general diagram.	7
2.7: Digital signature application with extra privacy.	7
2.8: One way password file general diagram.	8
2.9: Hash function properties.	10
2.10: General structure of secure hash code.	13
3.1: Small change effect at message.	14
3.2: Padding process of hash function for 512 block size.	17
3.3: SHA-1 pseudo code steps.	19
3.4: Constant values for SHA-224 and SHA-256.	21
3.5: Constant values for SHA-384, SHA-512, SHA-512/224, SHA-512/256.	22
3.6: Padding process of hash function for 1024 block size.	23
3.7: SHA-256 pseudo code steps.	27
3.8: SHA-512 pseudo code steps.	29
3.9: Demonstration of row, column and lane.	32
3.10: Demonstration of plane, slice and sheet.	32
3.11: Demonstration of 3 dimensional state array.	33
3.12: Demonstration of bits placement to state array.	34
3.13: Demonstration of lanes.	36
3.14: Demonstration of S string.	37
3.15: Demonstration of x, y and z coordinates of state array.	38
3.16: For “ θ ” pseudo code steps.	39
3.17: For “ p ” pseudo code steps.	39
3.18: For “ π ” pseudo code steps.	40
3.19: For “ χ ” pseudo code steps.	41
3.20: Demonstration of χ applied to a single row.	41

3.21: For “RC” pseudo code steps.	42
3.22: For “t” pseudo code steps.	43
3.23: For “Keccak – permutation” pseudo code steps.	44
3.24: The sponge construction.	45
3.25: For “sponge construction” pseudo code steps.	46
3.26: For “padding rule” pseudo code steps.	47
4.1: STM32F407VG microprocessor block diagram.	50
4.2: STM32F4DISCOVERY microprocessor board.	51
4.3: Timer clock configuration via STM32CubeMX.	51
4.4: Timer detail configuration via STM32CubeMX.	52
4.5: All rounds and message digests for SHA-1.	53
4.6: All rounds and message digests for SHA-224.	54
4.7: All rounds and message digests for SHA-256.	55
4.8: All rounds and message digests for SHA-384.	56
4.9: All rounds and message digests for SHA-512.	57
4.10: All rounds and message digests for SHA-512/224.	58
4.11: All rounds and message digests for SHA-512/256.	59
4.12: Demonstration of terminal program.	60

LIST of TABLES

<u>Table No:</u>	<u>Page</u>
2.1: Necessities and descriptions for a hash function.	9
2.2: Hash function resistance properties.	11
3.1: Hash function parameters comparison.	15
3.2: SHA-1 hash constant values.	16
3.3: SHA-1 initial hash values.	18
3.4: SHA-224 hash constant values.	23
3.5: SHA-256 hash constant values.	24
3.6: SHA-384 hash constant values.	24
3.7: SHA-512 hash constant values.	25
3.8: SHA-512/224 hash constant values.	25
3.9: SHA-512/224 hash constant values.	26
3.10: KECCAK-p permutation widths and related quantities.	31
3.11: Demonstration of ρ offset.	40
3.12: Illustration of KECCAK[c] (N, d).	47
3.13: Illustration of SHAKE.	48
4.1: SHA-1 and SHA2 algorithms comparison(msec).	61
4.2: Graphic of SHA-1 and SHA2 algorithms comparison(msec).	61
4.3: SHA-3 and SHAKE algorithms comparison(msec).	62
4.4: Graphic of SHA-3 and SHAKE algorithms comparison(msec).	62

1. INTRODUCTION

In this thesis, hash functions SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256, SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE-128 and SHAKE-256 are implemented on an ARM processor, then are compared their performance results using the same input sequences.

Information is very important for people and institutions and needs to be protected. The leakage of information that has vital value for a person or organization can have serious results for that person or organization. As a result of this reason, data used in information technologies must be secured within the scope of information security. A good information security policy should have three features: confidentiality, integrity and availability [1]. With the confidentiality feature, only people who have right to monitor data can access data. Integrity is the feature which ensures that data is stored correctly and will be stay correctly. Availability, the latest feature guarantees accurate access to data at any time.

The hash functions used to ensure information security for people and institutions. Using the hash functions, when transferring data from one computer to another, it can be guaranteed that the copied data is the same as the data at the source. If even one bit changes in the data, a different message digest will be created by the hash functions and it will be easily noticed that the data is changed. Using the hash functions: Message authentication can be provided, digital signatures can be implemented, one way password file and pseudorandom number generator can be created [2].

Many SHA algorithms have been developed so far. This process started with MD5 and continued with SHA-1, SHA-2 and SHA-3. SHA-2 algorithm represents six different algorithms as stated in Federal Information Processing Standards (FIPS) PUB 180-4: SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256[3]. Moreover SHA-3 is also represents six different algorithms as stated in Federal Information Processing Standards (FIPS) PUB 202: SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE128 and SHAKE256[4].

SHA1 and SHA2 algorithms basically consist of a few steps. In the first step, padding message operation is performed. In the second step, padded message is parsed into m-bit blocks. After that initial values are set for hash computation. Finally,

iteratively computation is performed. The SHA3 algorithm has a different structure than SHA1 and SHA2. Here, a sponge structure is used and calculation is made with KECCAK-p permutation consisting of a sequence of five transformation steps.

In the literature researchers can come across several hash function implementations. These implementations are different from each other considering the properties such as area, speed and throughput. Kim et al. designed the SHA3 hash processor, which produced four different message digests: 512, 384, 256 and 224 bits with the FPGA implementation [5]. Wu and Li made a SHA3 hardware structure that speed up the throughput to 65.5 Gbps, 61.8 Gbps, 47.2 Gbps and 32.7 Gbps for 224, 256, 384 and 512 bit message digest respectively [6]. Sideris et al. have demonstrated two techniques to improve the performance of SHA-2 computing by implementing a hardware application with Nios II processor on FPGA [7]. Gueron has worked on some software techniques to speed SHA-1, SHA-256 and SHA-512[8]. Sobti et al. used the ARM Cortex M3 processor and compared the performance of Grøstl, Blake and JH hash algorithms which were in the competition organized by the National Institute of Standards and Technology (NIST) for SHA-3 [9].

In this study, using ARM cortex-4 microprocessor thirteen algorithms, one from SHA-1, six from SHA-2, and six from SHA3 were compared according to their performance. In these thirteen algorithms, the message digests were obtained after using input messages from three bytes to sixteen kilobytes of data. The input data consists of the following characters continuously: "the quick brown fox jumps over the lazy dog.123456789". In this sentence, all letters at the English language are used at least once and a meaningful expression is created.

This thesis consists of five main sections. The first part is introduction part and includes the purpose and content of the thesis and a literature review. In the second part, usage areas of cryptographic hash functions are mentioned. In addition, features that should have a good hash function are defined and attacks on hash functions are mentioned. In the third section, pseudocodes of all hash functions are analyzed. In the fourth section, experimental results are demonstrated. In the fifth section, the performance results of the algorithms are evaluated and discussed.

2. CRYPTOGRAPHIC HASH FUNCTIONS

A cryptographic hash function takes variable-length data as input and creates a fixed-length hash value from it. A good hash function has a large set of inputs, and fixed-length outputs that are randomly distributed. Because of avalanche effect a smallest change at the input message will cause a visible change at the hash value.

The cryptographic hash function must have two main features. The first of them; It should be practically impossible to create a message suitable for a particular hash value. This can be defined as one-way property. Secondly, it is impossible to find two different messages that result the same hash value in practical. This can be also defined as collision-free property [2].

Using a hash function with these features, it easy to determine whether data has changed or not. At figure 2.1 general perspective of a hash function is illustrated. It is understood from figure 2.1 a hash function is applied to any length of an input message, then a fingerprint is made as fixed length output.

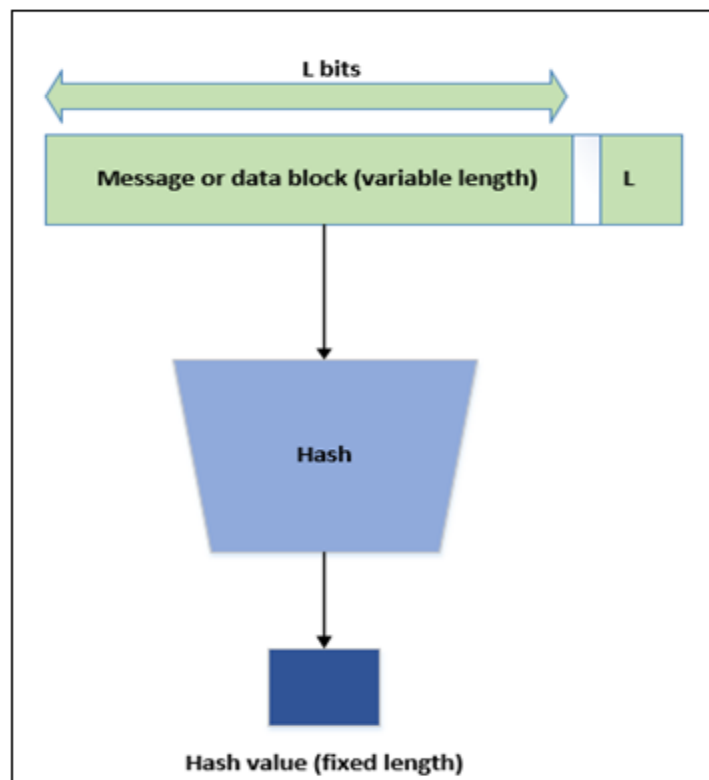


Figure 2.1: General diagram of cryptographic hash function.

2.1. Applications of Hash Functions

Cryptographic hash functions are widely used at most of areas today. These functions are mostly used in security applications and internet protocols. The usage areas of cryptographic hash functions are described below.

2.1.1. Message Authentication

Message Authentication is a mechanism to ensure data integrity. It guarantees that the content of the data is exactly the same for the sender and the receiver. Thus, both the sender and the receiver ensure that the data is not modified, inserted or deleted. Message Authentication applications that using the hash functions are explained at figure 2.2, figure 2.3, figure 2.4 and figure 2.5.

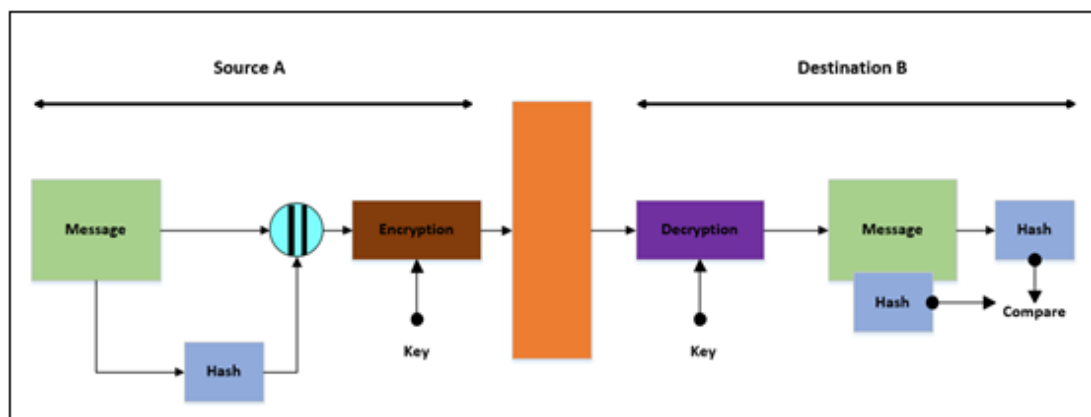


Figure 2.2: Message authentication application version A.

In this type application the hash value of the message is added to the end of the message, then the obtained data is encrypted using symmetric encryption as at shown figure 2.2. Applying encryption to the whole message plus hash value increases confidentiality.

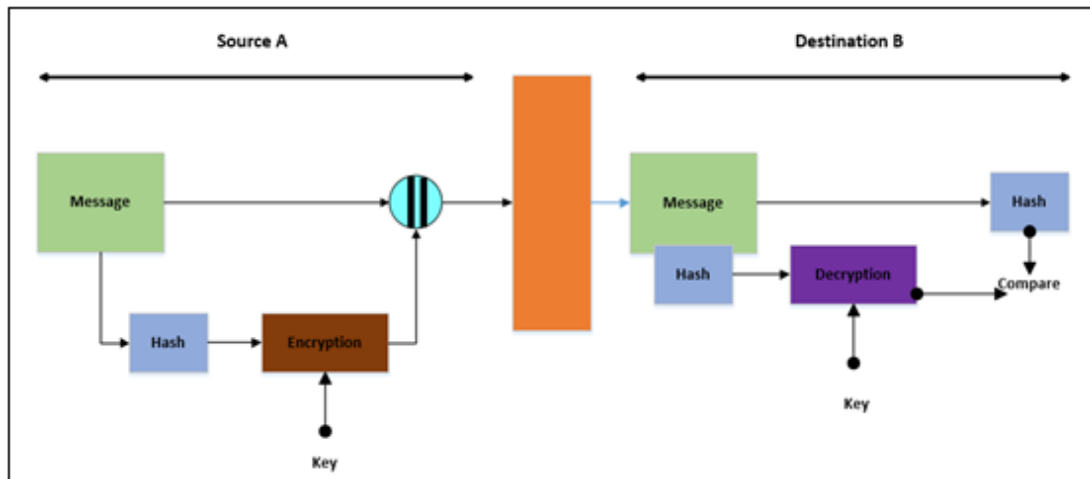


Figure 2.3: Message authentication application version B.

In this type application only the hash value is encrypted with symmetric encryption, then the encrypted hash value is added to the message as shown at figure 2.3. This method is preferred to avoid overprocessing, but in this procedure, confidentiality is not very important.

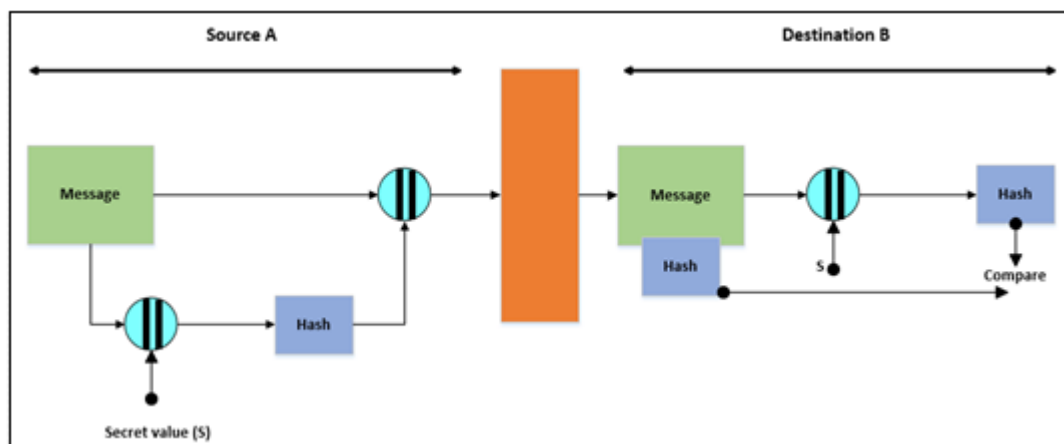


Figure 2.4: Message authentication application version C.

For Message Authentication, the hash function can be implemented without using encryption. In this method a secret value (S) used by the both parties. Firstly, the secret value is added to the message, then the obtained data is hashed. After that operation, the hash value is appended to the message as shown at figure 2.4. Since the secret value is not sent independently, the attacker neither can change the message nor produce false message.

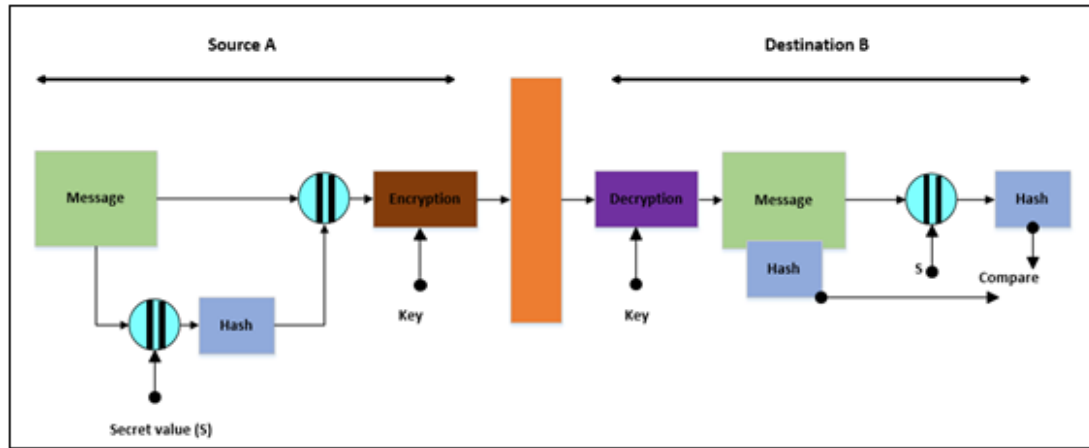


Figure 2.5: Message authentication application version D.

The method at C is applied exactly. Moreover the hash value and the whole message are encrypted with symmetric encryption to increase confidentiality as shown at figure 2.5.

If there is no need to confidentiality, the method at B has advantage over the methods at A and D. Because at A and D methods, the entire message should be encrypted, while at B only the hash value is encrypted. If it is desired to avoid from the encryption step completely, the method at C can be preferred. Because the encryption process is slow. Moreover, most of the encryption hardware equipment does not consist of cheap parts. In order to use encryption algorithms, the license fee must be paid. Considering all these, it is seen that the method at C has serious advantages over other methods.

2.1.2. Digital Signatures

Another area where the hash functions used is digital signature applications. In fact, digital signature and message authentication processes are similar generally. At the digital signature process, the hash value of the message is encrypted via the private key of the user. People who know the user public key value can understand with the digital signature whether or not data integrity has been protected. If the attacker wants to change the data, the user private key value must be obtained.

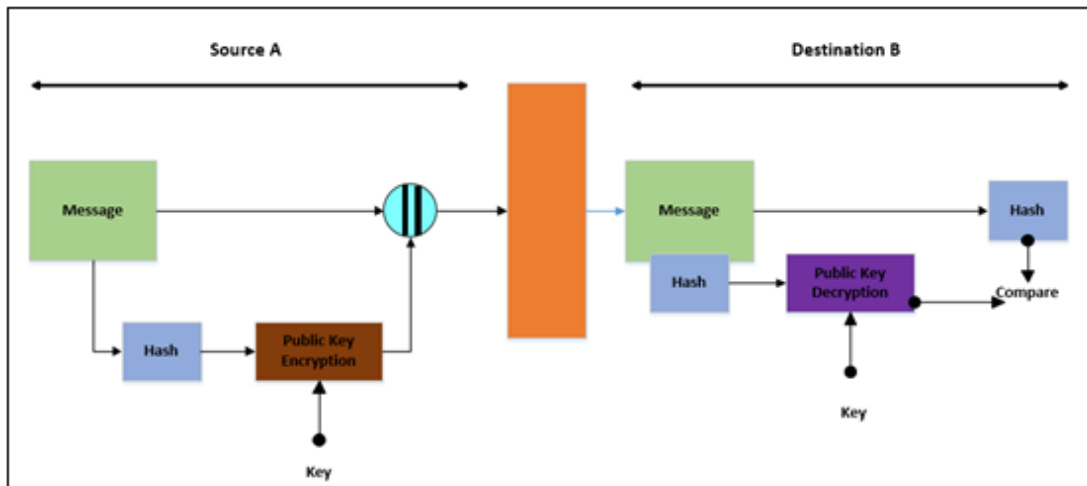


Figure 2.6: Digital signature general diagram.

The figure shows how the hash function is used when creating a digital signature. As shown at figure 2.6, it is seen that the hash code is encrypted by using the public key encryption with the sender's private key value. This flow chart was actually used in the message authentication process.

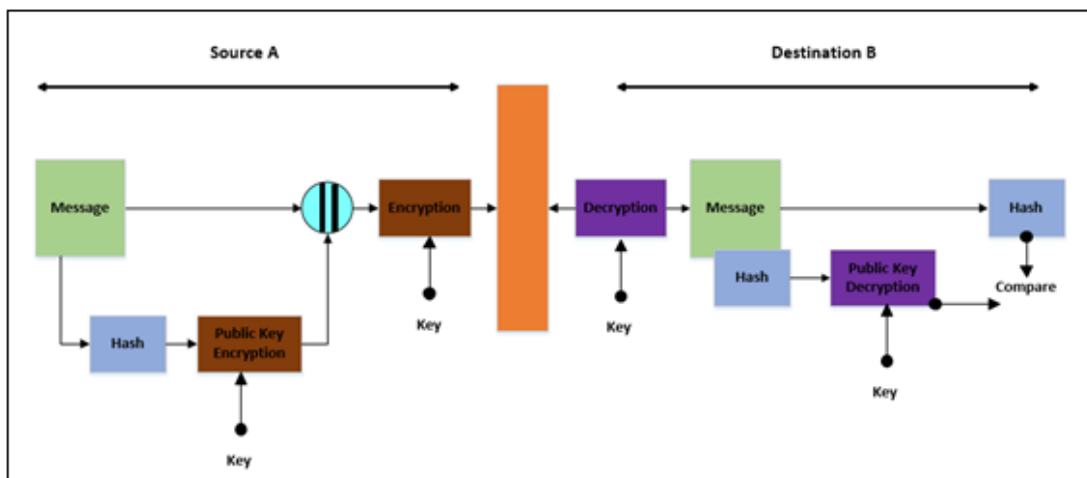


Figure 2.7: Digital signature application with extra privacy.

In the digital signature, if privacy is also important, then at the last process the data can be encrypted with a symmetric secret key as shown figure-7. In short, figure 2.7 is obtained after adding symmetric encryption function to figure 2.6 to increase confidentiality.

2.1.3. One Way Password File

Passwords are not stored as they are in operating systems. Because this situation is not safe enough. As shown at figure 2.8 in order to store password in a secure state, the password is stored in operating systems as a hash value.

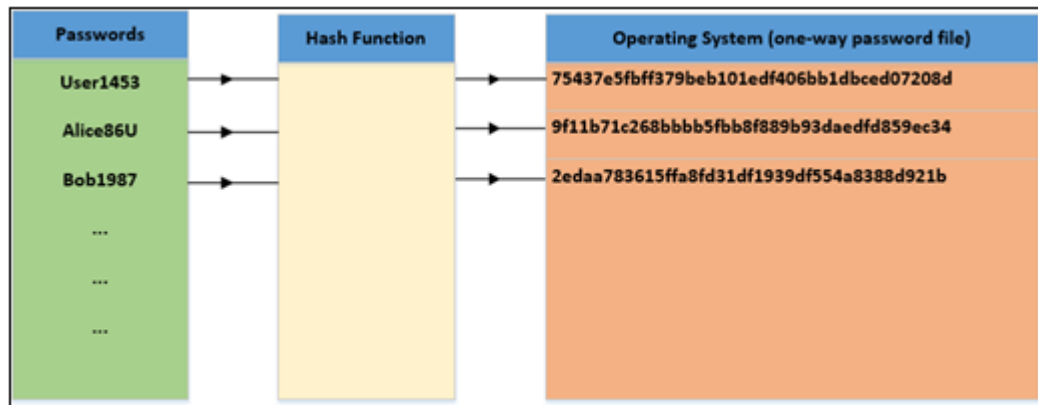


Figure 2.8: One way password file general diagram.

Thus, even if malicious people get the password file, they can never obtain the passwords itself. This system works like this: The user enters password, then the hash value of the entered password is created. This hash value is compared to the hash value that previously stored. Thus, the verification process is carried out. Thanks to this approach, password security is provided by using hash functions.

2.1.4. Intrusion Detection

The hash value of each file is stored in a system. Even if the attacker makes the slightest change in any file, the hash value of the file will change. Thus, the attacker would not be able to hide himself. The attacker wants to make changes at the file while the hash value remains the same. This is an application that is mathematically possible, but practically not possible.

2.2. Requirements of Hash Functions

As shown on table 2.1, cryptographic hash functions should have few features. Some of them are mandatory, and the others are optional.

Table 2.1: Necessities and descriptions for a hash function.

<u>Necessities</u>	<u>Descriptions</u>
• Variable input size • Fixed output size • Efficiency • Preimage resistant (one-way property) • Second preimage resistant (weak collision resistant) • Collision resistant (strong collision resistant) • Pseudorandomness	• H can be applied to a block of data of any size. • H produces a fixed-length output. • $H(x)$ is relatively easy to compute for any given x, making both hardware and software implementations practical. • For any given hash value h, it is computationally infeasible to find y such that $H(y) = h$. • For any given block x, it is computationally infeasible to find $y \neq x$ with $H(y) = H(x)$. • It is computationally infeasible to find any pair (x, y) such that $H(x) = H(y)$. • Output of H meets standard tests for pseudorandomness.

The first three items seen on the table 2.1 are the features that should be in all hash algorithms. These are: "variable input size", "fixed output size", "efficiency". The first feature "variable input size" means that hash function can be applied to any length of data block. This value starts from zero and goes up to the maximum limit specified in the hash function. At the second feature "fixed output size", the hash function produces a hash value with a fixed length. In other words; the input size can be null, one bit or maximum bit that specified in the hash function, at the end of hash process the message digest of the input must have fixed length. The third feature "efficiency" means that it is not difficult to calculate the hash value of any given message as hardware and software.

Fourth feature "preimage resistant": It is easy to get a hash value for a given message, but it is almost impossible to create a message for a given hash value. Thus, thanks to this feature, message authentication is provided. For that reason, this feature is also known as one-way property. If the one-way property was not available, the message would be obtained from the hash value and data security could not be provided.

The fifth feature "secondary preimage resistant": It is guaranteed that it is impossible to have another message that has the same hash value. With this feature, it is possible to prevent forgery. If this feature was not available, the attacker could find

a different message with the same hash value. As a result of this confusion fraud will occurred.

If a hash function has the first five properties on table 2.1, this hash function is called as weak hash function. To make the weak hash function strong, the hash function must also have the sixth property (collision resistant) shown on table 2.1. This feature means that having two different messages with the same hash value is computationally infeasible.

Three resistant properties are expressed as wenn diagram at figure 2.9. As seen that the "collision resistant" feature is a subset of the "second preimage resistant" feature. In other words, if a hash function has a "collision resistant" feature, it also has the "second preimage resistant" feature. On the other hand, a hash function with a "second preimage resistant" feature does not always have "collision resistant" feature.

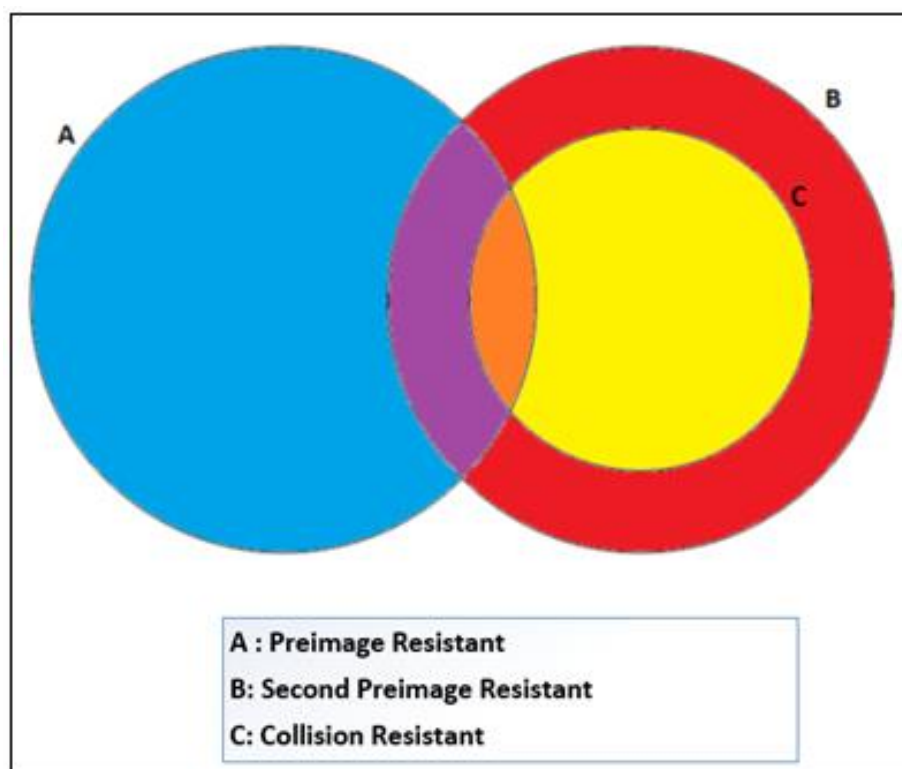


Figure 2.9: Hash function properties.

Table 2.2 summarizes which applications using hash functions need these resistant properties.

Table 2.2: Hash function resistance properties.

Application	Preimage Resistant	Second Preimage	Collision Resistant
Hash + digital signature	YES	YES	YES
Intrusion detection and virus detection		YES	
Hash + symmetric encryption			
One-way password file	YES		
MAC (Message Authentication Code)	YES	YES	YES

Seventh feature "pseudorandomness": If a hash function can produce pseudorandom output, it means that the hash function has this feature. The hash values obtained from the message should appear random in order to provide the three resistant properties mentioned figure 2.9. So these resistant properties are directly related to the pseudorandomness property. If a hash function has the pseudorandomness feature, it can be said that it also has resistant features substantially.

2.3. Types of Attacks to Secure Hash functions

Attacks against hash functions are described at the following sections.

2.3.1. Brute-Force Attacks

The remarkable factor at this attack is the bit length of the hash value. There is no special method or algorithm used at brute-force attack. The only thing that important is the bit length of the hash value. Laatansa et al. have done a research on brute-force and dictionary attack on SHA-1 hashed password [10]. As a result of this study researchers reached the experimental results that if someone uses password with 7 characters or less, then using brute-force attack it can be cracked easily.

2.3.1.1. Preimage and Second Preimage Attacks

In these attacks, the attacker wants to find a message that corresponds to the given hash value. Random messages are generated until collision occurs, in other words until the same hash values are obtained. A hash code with an m bit fixed length, it has 2^m different hash values. If the attacker tries on average 2^{m-1} times, he can find the message that corresponds to that hash value. Chen et al. examined a special hash algorithm in terms of preimage and second preimage attack [11]. With a kind of combined method the researchers could decrease the ideal time complexity of 2^{64} to $2^{60.7}$. As seen from this work, preimage and second preimage attack need more effort to crack hash value.

2.3.1.2. Collision Resistant Attacks

In this attack, the attacker wants to find two different messages that can generate the same hash value. This operation requires less effort compared to preimage and second preimage attacks. It is mentioned at section 2.3.1.1 that for the hash code with an m bit fixed length, it has 2^m hash value. The availability of 2 different messages for the same hash is expected to be found in this attack with a $\sqrt{2^m}$ trial. That value is equal to $2^{m/2}$. That result is also defined as birthday paradox.

2.3.2. Cryptanalysis Attacks

Attacks against hash algorithms can be carried out using some of the algorithm's weaknesses. In a good hash algorithm, the effort spent for the cryptanalytic attack must be equal to or greater than the effort spent at the brute-force attack. In this way, it can be measured whether a hash function is resistant to cryptanalytic attack.

In general, a hash function takes an input message and divides it into fixed-length blocks that consist of b bits. If necessary, padding is applied to the last block to ensure that the last block contains b bits. Also, the last block stores the value of the input message length. Embedding message length information in the last block will make the attacker's job even more difficult. At this state the attacker must find two messages

of equal length or two messages of different lengths that together with their length values which result the same hash value.

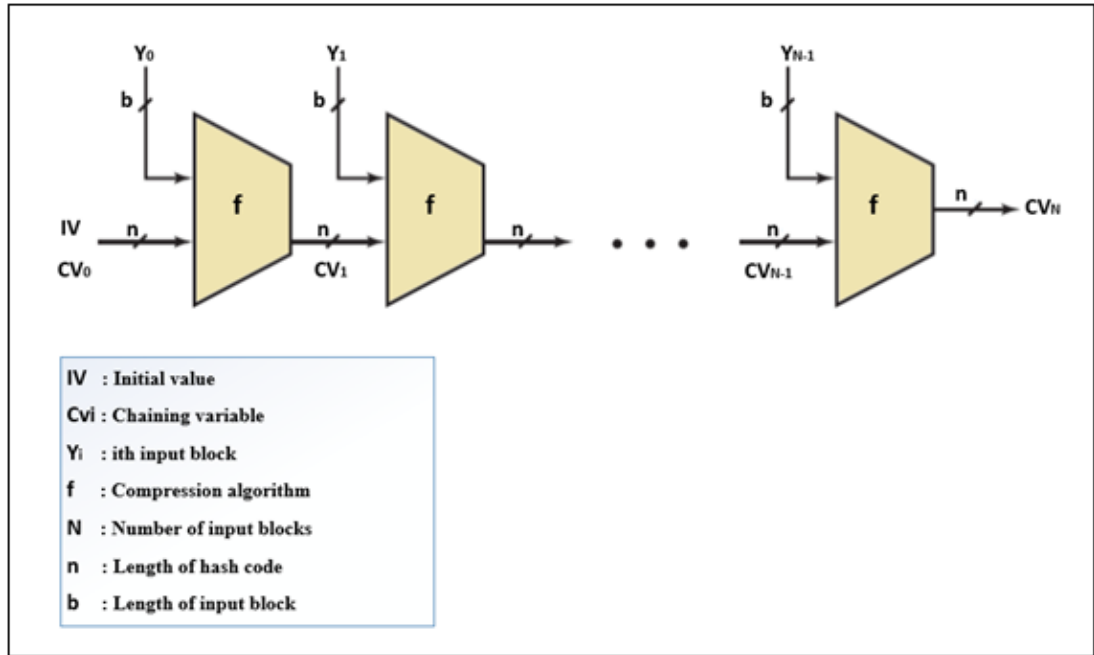


Figure 2.10: General structure of secure hash code.

Figure 2.10 shows the general structure of a secure hash function. This hash function has a compression function that is used repeatedly. It has two input parameters. One of them is n bit input from the previous step that is called as chaining variable. The other one is b bit blocks that is a part of input message. After the input parameters at the compression function is processed, n bit output is produced. Parameter b is often chosen longer than parameter n , so this term is called as compression. If the compression function has collision resistant feature, then the resultant iterated hash function is also collision resistant.

In hash functions, cryptanalyses focus on the internal structure of the compression function. Once the compression function is executed, effective techniques are tried to find collisions. After this process is done, the initial value (IV) used in the compression function should be taken into account. Compression function consists of a series of processing rounds. Thus, the attack requires the analysis of the pattern of the bit changes from round to round.

3. DEFINITION SECURE HASH ALGORITHM

In this section, SHA-1, SHA-2 and SHA3 algorithms will be analyzed. A total of thirteen algorithms will be examined, one from SHA-1, six from SHA-2 and six from SHA-3. All algorithms are iterative, one-way hash functions that can create message digests. By using these algorithms, the integrity of a message is provided. Even the slightest change in the message will result in a different message. For example two input message that very similar are hashed with SHA-1 and SHA-256 digest as shown at figure 3.1 and very different two message digests are obtained. The slightest change at the input message will change hash value incredibly because of the avalanche effect as seen above. Using this feature, it is possible to generate and verify digital signatures and message authentication codes and also random numbers or bits.

Input message-1: "the quick brown fox jumps over the lazy dog" SHA-1 hash result (160 bit): "16312751ef9307c3fd1afbcb993cdc80464ba0f1" SHA-256 hash result (256 bit): "05c6e08f1d9fdafa03147fcb8f82f124c76d2f70e3d989dc8aadb5e7d7450bec" Input message-2: "the quick brown fox jumps over the lazy dog" SHA-1 hash result (160 bit): "99740c93a2d16775503c6162c69953232b8ddea2" SHA-256 hash result (256 bit): "295989955eabb827444656761b4474a857834cea9b32f1f3a9493e68862e9417"

Figure 3.1: Small change effect at message.

All secure hash algorithms can be examined in two stages. The first is preprocessing and the other is hash computation. Preprocessing includes steps of padding message, parsing the padded message into m-bit blocks, and setting the initial values to be used in the hash computation. In the hash calculation firstly, a message schedule is created from the padded message. This schedule is used to generate a series of hash values iteratively, along with functions, constants, and word operations. The last generated hash value is used to determine the message digest. SHA-1 and SHA-2

algorithms differ in maximum message size, message digest size, block size, and data word size. On table 3.1 the basic features of these secure hash algorithms are shown.

Table 3.1: Hash function parameters comparison.

Property / Algorithm	SHA1	SHA224	SHA256	SHA384	SHA512	SHA512/224	SHA512/256
Digest Size	160	224	256	384	512	224	256
Message Size	$<2^{64}$	$<2^{64}$	$<2^{64}$	$<2^{128}$	$<2^{128}$	$<2^{128}$	$<2^{128}$
Block Size	512	512	512	1024	1024	1024	1024
Word Size	32	32	32	64	64	64	64
Number of Steps	80	64	64	80	80	80	80

At SHA-1, SHA-224 and SHA-256, the input message is divided into 512-bit blocks. Each block here is consisted of 16 sequences of 32-bit words.

At SHA-384, SHA-512, SHA-512/224 and SHA-512/256, the input message is divided into 1024-bit blocks. Each block here is consisted of 16 sequences of 64-bit words.

For SHA-1 message digest size is 160 bit and for the other secure hash algorithms message digest size is understandable with the name of algorithms which are appended at the end as numbers. While number of steps for SHA-224 and SHA-256 is 64, for the others this value is 80.

3.1. SHA-1

3.1.1. SHA-1 Functions

From zero to seventy-nine a sequence of logical functions, $f_0, f_1, \dots, f_{79}$ are used at SHA-1. Each function f_t , where t between 0–79, operates on three 32-bit words, x , y , and z , and produces a 32-bit word as output value. The function $f_t(x, y, z)$ can be defined as follows:

$$Ch(x, y, z) = (x \wedge y) \oplus (\sim x \wedge z) \quad 0 \leq t \leq 19 \quad (3.1.a)$$

$$Parity(x, y, z) = x \oplus y \oplus z \quad 20 \leq t \leq 39 \quad (3.1.b)$$

$$Maj(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \quad 40 \leq t \leq 59 \quad (3.1.c)$$

$$Parity(x, y, z) = x \oplus y \oplus z \quad 60 \leq t \leq 79 \quad (3.1.d)$$

3.1.2. SHA-1 Constants

At SHA-1 a unique constant value is used at every twenty round which is consisting of 32-bit words. For each twenty round the constant values are defined with the K_t . These constant words are shown at table 3.2 as hex format.

Table 3.2: SHA-1 hash constant values.

5a827999	$0 \leq t \leq 19$
6ed9eba1	$20 \leq t \leq 39$
8f1bbcdc	$40 \leq t \leq 59$
ca62c1d6	$60 \leq t \leq 79$

3.1.3. SHA-1 Preprocessing

Preprocessing stage consists of padding message, parsing message into message blocks and setting initial hash values.

3.1.3.1. Padding Message

At SHA-1 hash function, padding is performed to ensure that each block length has 512 bits. Padding operation must be done before the hash computation process. Suppose that there is a message with L bit length to be hashed. Firstly, bit “1” is added to the end of the message. Secondly K times zero bits follow bit “1”. For the length information of input message, 64 bits are reserved at the end of bit stream. So that

padding is up to for the first 448 bit. $L+1+K \equiv 448 \pmod{512}$. For instance, the (8-bit ASCII) input message “GTU” has length $8 \times 3 = 24$ bits, so the input message is padded with a one bit and 423 zero bits, after that the input message length is added at the end. As a result of this process the message is become a multiple of 512 bits that is shown at figure 3.2.

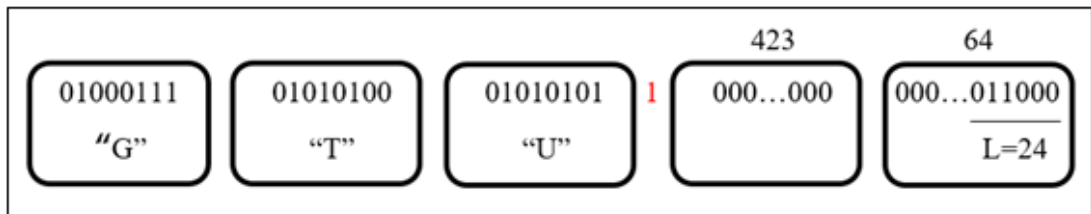


Figure 3.2: Padding process of hash function for 512 block size.

3.1.3.2. Parsing Message

After padding operation is completed at SHA-1, input message become a multiple of 512 bits. So the input message can be divided to 512 bits. As a result of division the message is parsed.

3.1.3.3. Setting the Initial Hash Value

Initial hash values that are constructed from 32-bit hex words are shown at table 3.3 for SHA-1.

Table 3.3: SHA-1 initial hash values.

$H_0^{(0)} =$	67452301
$H_1^{(0)} =$	efcdab89
$H_2^{(0)} =$	98badcfe
$H_3^{(0)} =$	10325476
$H_4^{(0)} =$	c3d2e1f0

3.1.4. SHA-1 Hash Computation

Each message block is processed as described at figure 3.3.

Step1: For each chunk that has 512 bits, the chunk is broken into sixteen 32-bit big-endian words $w[i]$, where $0 \leq i \leq 15$

Step2: for i from 16 to 79 $w[i]$ is described as below.

$w[i] = (w[i-3] \text{ xor } w[i-8] \text{ xor } w[i-14] \text{ xor } w[i-16]) \text{ leftrotate } 1$

Step3: Initialization of hash value is done for this chunk as below:

$a = H_0^{(0)}$

$b = H_1^{(0)}$

$c = H_2^{(0)}$

$d = H_3^{(0)}$

$e = H_4^{(0)}$

Step4: Main loop process is shown below.

for i from 0 to 79

if $0 \leq i \leq 19$ then

$f = (b \text{ and } c) \text{ or } ((\text{not } b) \text{ and } d)$

$k = 0x5A827999$

else if $20 \leq i \leq 39$

$f = b \text{ xor } c \text{ xor } d$

$k = 0x6ED9EBA1$

else if $40 \leq i \leq 59$

$f = (b \text{ and } c) \text{ or } (b \text{ and } d) \text{ or } (c \text{ and } d)$

$k = 0x8F1BBCDC$

else if $60 \leq i \leq 79$

$f = b \text{ xor } c \text{ xor } d$

$k = 0xCA62C1D6$

$\text{temp} = (a \text{ leftrotate } 5) + f + e + k + w[i]$

$e = d$

$d = c$

$c = b \text{ leftrotate } 30$

$b = a$

$a = \text{temp}$

Step5: For that chunk the hash result is below:

$H_0^{(i)} = H_0^{(i-1)} + a$

$H_1^{(i)} = H_1^{(i-1)} + b$

$H_2^{(i)} = H_2^{(i-1)} + c$

$H_3^{(i)} = H_3^{(i-1)} + d$

$H_4^{(i)} = H_4^{(i-1)} + e$

Step6: According to the big-endian format the final hash value is produced as 160-bit as follows.

$hh = (h_0 \text{ leftshift } 128) \text{ or } (h_1 \text{ leftshift } 96) \text{ or } (h_2 \text{ leftshift } 64) \text{ or } (h_3 \text{ leftshift } 32) \text{ or } h_4$

Figure 3.3: SHA-1 pseudo code steps.

3.2. SHA-2

3.2.1. SHA-2 Functions

Both at SHA-224 and SHA-256, six logical functions shown below that are represented by x , y , z are used. Their lengths are defined as 32-bit words. The result of the six functions are also 32-bit word.

$$Ch(x, y, z) = (x \wedge y) \oplus (\sim x \wedge z) \quad (3.2.a)$$

$$Maj(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \quad (3.2.b)$$

$$\sum_0^{256} (x) = ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x) \quad (3.2.c)$$

$$\sum_1^{256} (x) = ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x) \quad (3.2.d)$$

$$\Gamma_0^{256}(x) = ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x) \quad (3.2.e)$$

$$\Gamma_1^{256}(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x) \quad (3.2.f)$$

At all SHA-384, SHA-512, SHA-512/224 and SHA-512/256, also six logical functions shown below that are represented by x , y , z are used. Their lengths are defined as 64-bit words. The result of the six functions are also 64-bit word.

$$Ch(x, y, z) = (x \wedge y) \oplus (\sim x \wedge z) \quad (3.3.a)$$

$$Maj(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \quad (3.3.b)$$

$$\sum_0^{512}(x) = ROTR^{28}(x) \oplus ROTR^{34}(x) \oplus ROTR^{39}(x) \quad (3.3.c)$$

$$\sum_1^{512}(x) = ROTR^{14}(x) \oplus ROTR^{18}(x) \oplus ROTR^{41}(x) \quad (3.3.d)$$

$$r_0^{512}(x) = ROTR^1(x) \oplus ROTR^8(x) \oplus SHR^7(x) \quad (3.3.e)$$

$$r_1^{512}(x) = ROTR^{19}(x) \oplus ROTR^{61}(x) \oplus SHR^6(x) \quad (3.3.f)$$

3.2.2. SHA-2 Constants

3.2.2.1. SHA-224 and SHA-256 Constants

Both at SHA-224 and SHA-256 there are sixty-four constant values that are consisted of 32-bit words. These words are obtained from the first 32-bits of the fractional parts of the cube roots of the first sixty-four prime numbers[3]. These constant words are shown at figure 3.4 from left to right as hex format.

428a2f98	71374491	b5c0fbcf	e9b5dba5	3956c25b	59f111f1	923f82a4	ab1c5ed5
d807aa98	12835b01	243185be	550c7dc3	72be5d74	80deb1fe	9bdc06a7	c19bf174
e49b69c1	efbe4786	0fc19dc6	240ca1cc	2de92c6f	4a7484aa	5cb0a9dc	76f988da
983e5152	a831c66d	b00327c8	bf597fc7	c6e00bf3	d5a79147	06ca6351	14292967
27b70a85	2e1b2138	4d2c6dfc	53380d13	650a7354	766a0abb	81c2c92e	92722c85
a2bfe8a1	a81a664b	c24b8b70	c76c51a3	d192e819	d6990624	f40e3585	106aa070
19a4c116	1e376c08	2748774c	34b0bcb5	391c0cb3	4ed8aa4a	5b9cca4f	682e6ff3
748f82ee	78a5636f	84c87814	8cc70208	90befffa	a4506ceb	bef9a3f7	c67178f2

Figure 3.4: Constant values for SHA-224 and SHA-256.

3.2.2.2. SHA-384, SHA-512, SHA512/224, SHA512/256 Constants

At SHA-384, SHA-512, SHA-512/224 and SHA-512/256 there are eighty constant values consisting of 64-bit words. These words are obtained from the first 64-bits of the

fractional parts of the cube roots of the first eighty prime numbers. These constant words are shown at figure 3.5 from left to right as hex format.

428a2f98d728ae22	7137449123ef65cd	b5c0fbcfec4d3b2f	e9b5dba58189dbbc
3956c25bf348b538	59f111f1b605d019	923f82a4af194f9b	ab1c5ed5da6d8118
d807aa98a3030242	12835b0145706fbe	243185be4ee4b28c	550c7dc3d5ffb4e2
72be5d74f27b896f	80deb1fe3b1696b1	9bdc06a725c71235	c19bf174cf692694
e49b69c19ef14ad2	efbe4786384f25e3	0fc19dc68b8cd5b5	240ca1cc77ac9c65
2de92c6f592b0275	4a7484aa6ea6e483	5cb0a9dcdb41fbd4	76f988da831153b5
983e5152ee66dfab	a831c66d2db43210	b00327c898fb213f	bf597fc7beef0ee4
c6e00bf33da88fc2	d5a79147930aa725	06ca6351e003826f	142929670a0e6e70
27b70a8546d22ffc	2e1b21385c26c926	4d2c6dfc5ac42aed	53380d139d95b3df
650a73548baf63de	766a0abb3c77b2a8	81c2c92e47edaee6	92722c851482353b
a2bfe8a14cf10364	a81a664bbc423001	c24b8b70d0f89791	c76c51a30654be30
d192e819d6ef5218	d69906245565a910	f40e35855771202a	106aa07032bdbl8
19a4c116b8d2d0c8	1e376c085141ab53	2748774cdf8eeb99	34b0bcb5e19b48a8
391c0cb3c5c95a63	4ed8aa4ae3418acb	5b9cca4f7763e373	682e6ff3d6b2b8a3
748f82ee5defb2fc	78a5636f43172f60	84c87814a1f0ab72	8cc702081a6439ec
90befffa23631e28	a4506cebd82bde9	bef9a3f7b2c67915	c67178f2e372532b
ca273eceeaa26619c	d186b8c721c0c207	eada7dd6cde0eb1e	f57d4f7fee6ed178
06f067aa72176fba	0a637dc5a2c898a6	113f9804bef90dae	1b710b35131c471b
28db77f523047d84	32caab7b40c72493	3c9ebe0a15c9bebc	431d67c49c100d4c
4cc5d4becb3e42b6	597f299cfc657e2a	5fcb6fab3ad6faec	6c44198c4a475817

Figure 3.5: Constant values for SHA-384, SHA-512, SHA-512/224, SHA-512/256.

3.2.3. SHA-2 Preprocessing

Preprocessing stage consists of padding message, parsing message into message blocks and setting initial hash values.

3.2.3.1. Padding the Message

Padding process for SHA-224 and SHA-256 are the same as SHA-1 that is shown at 3.1.3.1 section. On the other hand, at SHA-384, SHA-512, SHA-512/224 and SHA-512/256 hash functions, padding is performed to ensure that each block length has 1024 bits. Padding operation must be done before the hash computation process. Suppose that there is a message with L bit length to be hashed. Firstly, bit “1” is added to the end of the message. Secondly K times zero bits follow bit “1”. For the length information of input message, 128 bits are reserved at the end of bit stream. So that padding is up to for the first 896 bit. $L+1+K \equiv 896 \pmod{1024}$. For instance, the (8-bit ASCII) input message “GTU” has length $8 \times 3 = 24$ bits, so the input message is padded with a one bit and 871 zero bits, after that the input message length is added at the end.

As a result of this process the message is become a multiple of 1024 bits that is shown at figure 3.6.

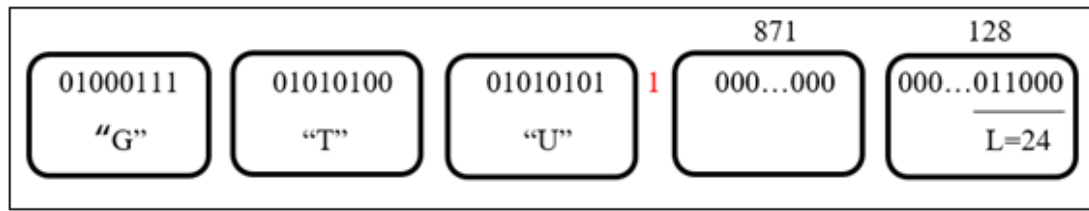


Figure 3.6: Padding process of hash function for 1024 block size.

3.2.3.2. Parsing the Message

After padding operation is completed at SHA-384, SHA-512, SHA-512/224 and SHA-512/256, input message become a multiple of 1024 bits. So, the input message can be divided to 1024 bits. As a result of division, the message is parsed.

3.2.3.3. Setting the Initial Hash Value

Initial hash values for SHA-224 are shown at table 3.4 that are constructed from 32-bit hex words.

Table 3.4: SHA-224 hash constant values.

$H_0^{(0)} =$	c1059ed8
$H_1^{(0)} =$	367cd507
$H_2^{(0)} =$	3070dd17
$H_3^{(0)} =$	f70e5939
$H_4^{(0)} =$	ffc00b31
$H_5^{(0)} =$	68581511
$H_6^{(0)} =$	64f98fa7
$H_7^{(0)} =$	befa4fa4

Initial hash values for SHA-256 are shown at table 3.5 that are constructed from 32-bit hex words.

Table 3.5: SHA-256 hash constant values.

$H_0^{(0)} =$	6a09e667
$H_1^{(0)} =$	bb67ae85
$H_2^{(0)} =$	3c6ef372
$H_3^{(0)} =$	a54ff53a
$H_4^{(0)} =$	510e527f
$H_5^{(0)} =$	9b05688c
$H_6^{(0)} =$	1f83d9ab
$H_7^{(0)} =$	5be0cd19

Initial hash values for SHA-384 are shown at table 3.6 that are constructed from 64-bit hex words.

Table 3.6: SHA-384 hash constant values.

$H_0^{(0)} =$	cbbb9d5dc1059ed8
$H_1^{(0)} =$	629a292a367cd507
$H_2^{(0)} =$	9159015a3070dd17
$H_3^{(0)} =$	152fec8d8f70e5939
$H_4^{(0)} =$	67332667ffc00b31
$H_5^{(0)} =$	8eb44a8768581511
$H_6^{(0)} =$	db0c2e0d64f98fa7
$H_7^{(0)} =$	47b5481dbefa4fa4

Initial hash values for SHA-512 are shown at table 3.7 that are constructed from 64-bit hex words.

Table 3.7: SHA-512 hash constant values.

$H_0^{(0)} =$	6a09e667f3bcc908
$H_1^{(0)} =$	bb67ae8584caa73b
$H_2^{(0)} =$	3c6ef372fe94f82b
$H_3^{(0)} =$	a54ff53a5f1d36f1
$H_4^{(0)} =$	510e527fade682d1
$H_5^{(0)} =$	9b05688c2b3e6c1f
$H_6^{(0)} =$	1f83d9abfb41bd6b
$H_7^{(0)} =$	5be0cd19137e2179

Initial hash values for SHA-512/224 are shown at table 3.8 that are constructed from 64-bit hex words.

Table 3.8: SHA-512/224 hash constant values.

$H_0^{(0)} =$	8C3D37C819544DA2
$H_1^{(0)} =$	73E1996689DCD4D6
$H_2^{(0)} =$	1DFAB7AE32FF9C82
$H_3^{(0)} =$	679DD514582F9FCF
$H_4^{(0)} =$	0F6D2B697BD44DA8
$H_5^{(0)} =$	77E36F7304C48942
$H_6^{(0)} =$	3F9D85A86A1D36C8
$H_7^{(0)} =$	1112E6AD91D692A1

Initial hash values for SHA-512/256 are shown at table 3.9 that are constructed from 64-bit hex words.

Table 3.9: SHA-512/224 hash constant values.

$H_0^{(0)} =$	22312194FC2BF72C
$H_1^{(0)} =$	9F555FA3C84C64C2
$H_2^{(0)} =$	2393B86B6F53B151
$H_3^{(0)} =$	963877195940EABD
$H_4^{(0)} =$	96283EE2A88EFFE3
$H_5^{(0)} =$	BE5E1E2553863992
$H_6^{(0)} =$	2B0199FC2C85B8AA
$H_7^{(0)} =$	0EB72DDC81C52CA2

3.2.4. SHA-2 Hash Computation

3.2.4.1. SHA-256 Hash Computation

Each message block is processed as described at figure 3.7.

Step1: For each chunk that has 512 bits, the chunk is broken into sixteen 32-bit big-endian words $w[i]$, where $0 \leq i \leq 15$

Step2: for i from 16 to 63 $w[i]$ is described as below.

$s0 := (w[i-15] \text{ rightrotate } 7) \text{ xor } (w[i-15] \text{ rightrotate } 18) \text{ xor } (w[i-15] \text{ rightshift } 3)$

$s1 := (w[i-2] \text{ rightrotate } 17) \text{ xor } (w[i-2] \text{ rightrotate } 19) \text{ xor } (w[i-2] \text{ rightshift } 10)$

$w[i] := w[i-16] + s0 + w[i-7] + s1$

Step3: Initialization of hash value is done for this chunk as below:

$a = H_0^{(0)}$

$b = H_1^{(0)}$

$c = H_2^{(0)}$

$d = H_3^{(0)}$

$e = H_4^{(0)}$

$f = H_5^{(0)}$

$g = H_6^{(0)}$

$h = H_7^{(0)}$

Step4: Main loop process is shown below.

for i from 0 to 63

$S1 := (e \text{ rightrotate } 6) \text{ xor } (e \text{ rightrotate } 11) \text{ xor } (e \text{ rightrotate } 25)$

$ch := (e \text{ and } f) \text{ xor } ((\text{not } e) \text{ and } g)$

$temp1 := h + S1 + ch + k[i] + w[i]$

$S0 := (a \text{ rightrotate } 2) \text{ xor } (a \text{ rightrotate } 13) \text{ xor } (a \text{ rightrotate } 22)$

$maj := (a \text{ and } b) \text{ xor } (a \text{ and } c) \text{ xor } (b \text{ and } c)$

$temp2 := S0 + maj$

$h := g$

$g := f$

$f := e$

$e := d + temp1$

$d := c$

$c := b$

$b := a$

$a := temp1 + temp2$

Step5: For that chunk the hash result is below:

$H_0^{(i)} = H_0^{(i-1)} + a$

$H_1^{(i)} = H_1^{(i-1)} + b$

$H_2^{(i)} = H_2^{(i-1)} + c$

$H_3^{(i)} = H_3^{(i-1)} + d$

$H_4^{(i)} = H_4^{(i-1)} + e$

$H_5^{(i)} = H_5^{(i-1)} + f$

$H_6^{(i)} = H_6^{(i-1)} + g$

$H_7^{(i)} = H_7^{(i-1)} + h$

Step6: According to the big-endian format the final hash value is produced as 256-bit as follows.

$hh = (h0 \text{ leftshift } 224) \text{ or } (h1 \text{ leftshift } 192) \text{ or } (h2 \text{ leftshift } 160) \text{ or } (h3 \text{ leftshift } 128) \text{ or } (h4 \text{ leftshift } 96) \text{ or } (h5 \text{ leftshift } 64) \text{ or } (h6 \text{ leftshift } 32) \text{ or } h7$

Figure 3.7: SHA-256 pseudo code steps.

3.2.4.2.SHA-224 Hash Computation

Hash computation of SHA-224 is the same as SHA-256, except that the initial hash values of SHA-224 h0 through h7 are different from SHA-256, and the output is constructed by omitting h7.

3.2.4.3.SHA-512 Hash Computation

Each message block is processed as described at figure 3.8.

Step1: For each chunk that has 1024 bits, the chunk is broken into sixteen 64-bit big-endian words $w[i]$, where $0 \leq i \leq 15$

Step2: for i from 16 to 79 $w[i]$ is described as below.

$s0 := (w[i-15] \text{ rightrotate } 1) \text{ xor } (w[i-15] \text{ rightrotate } 8) \text{ xor } (w[i-15] \text{ rightshift } 7)$

$s1 := (w[i-2] \text{ rightrotate } 19) \text{ xor } (w[i-2] \text{ rightrotate } 61) \text{ xor } (w[i-2] \text{ rightshift } 6)$

$w[i] := w[i-16] + s0 + w[i-7] + s1$

Step3: Initialization of hash value is done for this chunk as below:

$a = H_0^{(0)}$

$b = H_1^{(0)}$

$c = H_2^{(0)}$

$d = H_3^{(0)}$

$e = H_4^{(0)}$

$f = H_5^{(0)}$

$g = H_6^{(0)}$

$h = H_7^{(0)}$

Step4: Main loop process is shown below.

for i from 0 to 79

$S1 := (e \text{ rightrotate } 14) \text{ xor } (e \text{ rightrotate } 18) \text{ xor } (e \text{ rightrotate } 41)$

$ch := (e \text{ and } f) \text{ xor } ((\text{not } e) \text{ and } g)$

$temp1 := h + S1 + ch + k[i] + w[i]$

$S0 := (a \text{ rightrotate } 28) \text{ xor } (a \text{ rightrotate } 34) \text{ xor } (a \text{ rightrotate } 39)$

$maj := (a \text{ and } b) \text{ xor } (a \text{ and } c) \text{ xor } (b \text{ and } c)$

$temp2 := S0 + maj$

$h := g$

$g := f$

$f := e$

$e := d + temp1$

$d := c$

$c := b$

$b := a$

$a := temp1 + temp2$

Step5: For that chunk the hash result is below:

$H_0^{(i)} = H_0^{(i-1)} + a$

$H_1^{(i)} = H_1^{(i-1)} + b$

$H_2^{(i)} = H_2^{(i-1)} + c$

$H_3^{(i)} = H_3^{(i-1)} + d$

$H_4^{(i)} = H_4^{(i-1)} + e$

$H_5^{(i)} = H_5^{(i-1)} + f$

$H_6^{(i)} = H_6^{(i-1)} + g$

$H_7^{(i)} = H_7^{(i-1)} + h$

Step6: According to the big-endian format the final hash value is produced as 512-bit as follows.

$hh = (h0 \text{ leftshift } 448) \text{ or } (h1 \text{ leftshift } 384) \text{ or } (h2 \text{ leftshift } 320) \text{ or } (h3 \text{ leftshift } 256) \text{ or } (h4 \text{ leftshift } 192) \text{ or } (h5 \text{ leftshift } 128) \text{ or } (h6 \text{ leftshift } 64) \text{ or } h7.$

Figure 3.8: SHA-512 pseudo code steps.

3.2.4.4.SHA-384 Hash Computation

Hash computation of SHA-384 is the same as SHA-512, except that the initial hash values of SHA-384 h0 through h7 are different from SHA-512, and the output is constructed by omitting h6 and h7.

3.2.4.5.SHA-512/224 Hash Computation

Hash computation of SHA-512/224 is the same as SHA-512, except that the initial hash values of SHA-512/224 h0 through h7 are different from SHA-512, and the output is constructed by taking only the first 224 bits.

3.2.4.6.SHA-512/256 Hash Computation

Hash computation of SHA-512/256 is the same as SHA-512, except that: the initial hash values of SHA-512/256 h0 through h7 are different from SHA-512, and the output is constructed by taking only the first 256 bits.

3.3. SHA-3 (Keccak)

SHA3-224, SHA3-256, SHA3-384, SHA3-512, and two extendable-output functions SHAKE128, SHAKE256 are defined as SHA-3 cryptographic hash functions [4]. The three numbers after SHA3- indicate length of message digest as bit. However, the suffixes of SHAKE indicate the security strengths of these functions. Moreover, by using SHAKE functions the output digest value can be extended to any desired length. These six cryptographic hash functions have the same structure that is called as sponge construction [4].

In fact the SHA-3 functions are modes of operation of the permutation that is called KECCAK-p. This permutation has most of advantages like modifying its size and security parameters to provide the flexibility in future documents [4].

3.3.1. Keccak-p Permutations

KECCAK-p permutations can be represented by two parameters:

1) Width of permutation: The fixed length of the strings that are permuted. The width value is represented by b .

2) Round: Number of iterations of an internal transformation. The number of rounds are represented by n_r .

The KECCAK-p permutation can be illustrated as KECCAK-p [b, n_r]. The value of b can be chosen from these numbers {25, 50, 100, 200, 400, 800, 1600}. At the same time the value of n_r can be any positive integer.

A round of a KECCAK-p permutation that is illustrated as Rnd consists of a sequence of five transformation steps. The permutation is defined as an array of values for b bits which is repeatedly updated. This array is called as "state" and initially set to the input values of the permutation.

3.3.1.1. Keccak-p State

The KECCAK-p[b, n_r] permutation is consist of b bits for a "state". On table 3.10 two new parameters are seen that related to b value. One of them is w which means $b/25$, the other one is l which means $\log_2(b/25)$.

Table 3.10: KECCAK-p permutation widths and related quantities.

B	25	50	100	200	400	800	1600
W	1	2	4	8	16	32	64
L	0	1	2	3	4	5	6

The input and output states of the permutation can be illustrated as b -bit strings. These states consist of three dimensional 5-by-5-by- w arrays of bits. If a string that is defined as a "state" then its bits are set from 0 to $b-1$ as illustrated below:

$$S = S[0] || S[1] || \dots || S[b - 2] || S[b - 1] \quad (3.4)$$

If a state named A that consists of a 5-by-5-by-w array of bits then its indices are the integer triples (x, y, z) for which $0 \leq x \leq 4$, $0 \leq y \leq 4$, and $0 \leq z \leq w-1$.

One bit that corresponds to x, y, z values is illustrated as A[x, y, z]. As a result of this, A named state array is demonstrated as three-dimensional array.

3.3.1.2. Parts of the State Array

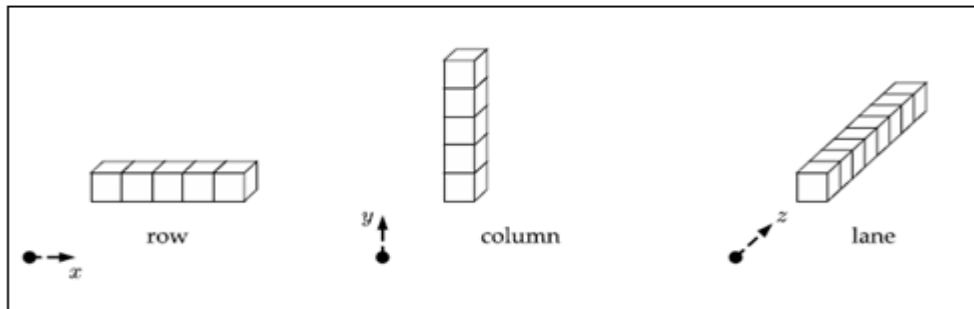


Figure 3.9: Demonstration of row, column and lane.

As shown at figure 3.9, one-dimensional subarrays on the x-axis direction are called rows, on the y-axis direction are called columns, and on the z-axis direction are called lanes. In each subarrays every cube is representing a bit.

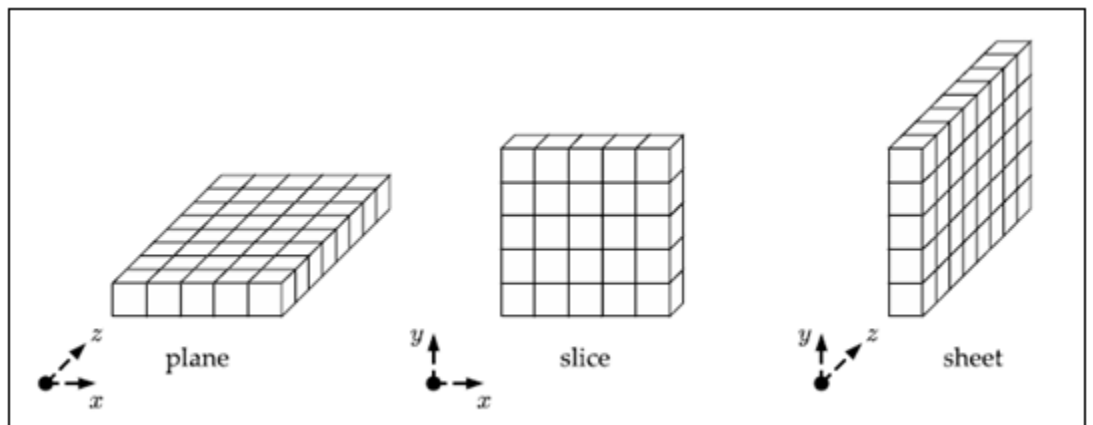


Figure 3.10: Demonstration of plane, slice and sheet.

As shown at figure 3.10, two-dimensional subarrays on the x-axis direction are called planes, on the y-axis direction are called slices, and on the z-axis direction are called sheets.

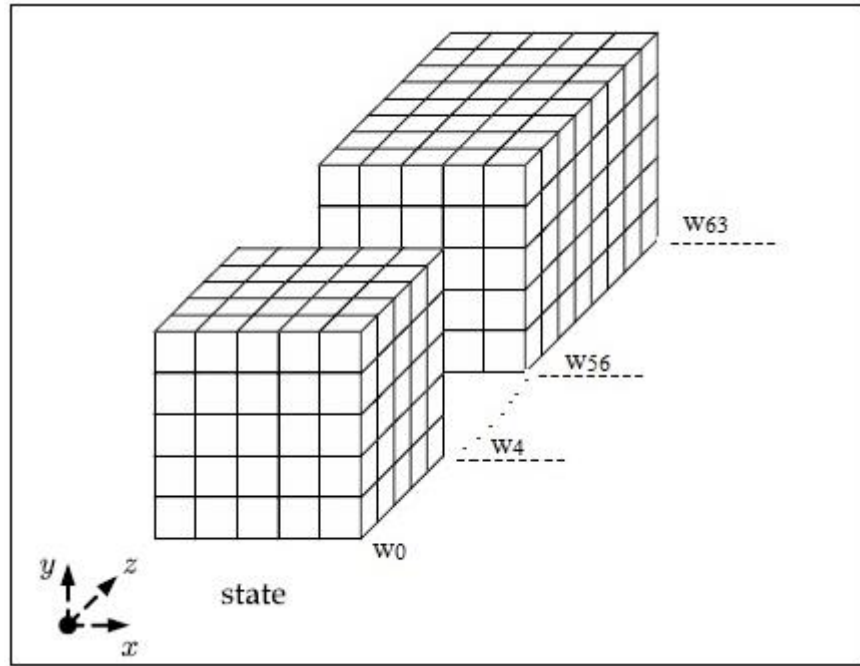


Figure 3.11: Demonstration of 3 dimensional state array.

As shown at the figure 3.11 the state array for a KECCAK-p permutation have parameters as $b=1600$ and as a result of b value w is equal to 64.

3.3.1.3. Converting Strings to State Arrays

The state for the KECCAK-p[b , n_r] permutation is illustrated with a b bits string named S illustrated below.

For all three values of x , y , z where $0 \leq x \leq 4$, $0 \leq y \leq 4$, and $0 \leq z \leq w-1$,

$$A[x, y, z] = S[w(5y + x) + z] \quad b = 1600, w = 64 \quad (3.5)$$

According to state array $A[x, y, z]$, bits placement is done and illustrated at figure 3.12.

Lane(0,0)	Lane(1,0)	Lane(4,0)	Plane(0)
A[0, 0, 0] = S [0]	A[1, 0, 0] = S [64]	A[4, 0, 0] = S [256]	
A[0, 0, 1] = S [1]	A[1, 0, 1] = S [65]	A[4, 0, 1] = S [257]	
A[0, 0, 2] = S [2]	A[1, 0, 2] = S [66]	A[4, 0, 2] = S [258]	
...			
A[0, 0, 61] = S [61]	A[1, 0, 61] = S [125]	A[4, 0, 61] = S [317]	
A[0, 0, 62] = S [62]	A[1, 0, 62] = S [126]	A[4, 0, 62] = S [318]	
A[0, 0, 63] = S [63]	A[1, 0, 63] = S [127]	A[4, 0, 63] = S [319]	
and			
A[0, 1, 0] = S [320]	A[1, 1, 0] = S [384]	A[4, 1, 0] = S [576]	Plane(1)
A[0, 1, 1] = S [321]	A[1, 1, 1] = S [385]	A[4, 1, 1] = S [577]	
A[0, 1, 2] = S [322]	A[1, 1, 2] = S [386]	A[4, 1, 2] = S [578]	
...			
A[0, 1, 61] = S [381]	A[1, 1, 61] = S [445]	A[4, 1, 61] = S [637]	
A[0, 1, 62] = S [382]	A[1, 1, 62] = S [446]	A[4, 1, 62] = S [638]	
A[0, 1, 63] = S [383]	A[1, 1, 63] = S [447]	A[4, 1, 63] = S [639]	
and			
A[0, 2, 0] = S [640]	A[1, 2, 0] = S [704]	A[4, 2, 0] = S [896]	Plane(2)
A[0, 2, 1] = S [641]	A[1, 2, 1] = S [705]	A[4, 2, 1] = S [897]	
A[0, 2, 2] = S [642]	A[1, 2, 2] = S [706]	A[4, 2, 2] = S [898]	
...			
A[0, 2, 61] = S [701]	A[1, 2, 61] = S [765]	A[4, 2, 61] = S [957]	
A[0, 2, 62] = S [702]	A[1, 2, 62] = S [766]	A[4, 2, 62] = S [958]	
A[0, 2, 63] = S [703]	A[1, 2, 63] = S [767]	A[4, 2, 63] = S [959]	
and			
A[0, 3, 0] = S [960]	A[1, 3, 0] = S [1024]	A[4, 3, 0] = S [1216]	Plane(3)
A[0, 3, 1] = S [961]	A[1, 3, 1] = S [1025]	A[4, 3, 1] = S [1217]	
A[0, 3, 2] = S [962]	A[1, 3, 2] = S [1026]	A[4, 3, 2] = S [1218]	
...			
A[0, 3, 61] = S [1081]	A[1, 3, 61] = S [1085]	A[4, 3, 61] = S [1277]	
A[0, 3, 62] = S [1082]	A[1, 3, 62] = S [1086]	A[4, 3, 62] = S [1278]	
A[0, 3, 63] = S [1083]	A[1, 3, 63] = S [1087]	A[4, 3, 63] = S [1279]	
and			
A[0, 4, 0] = S [1280]	A[1, 4, 0] = S [1344]	A[4, 4, 0] = S [1536]	Plane(4)
A[0, 4, 1] = S [1281]	A[1, 4, 1] = S [1345]	A[4, 4, 1] = S [1537]	
A[0, 4, 2] = S [1282]	A[1, 4, 2] = S [1346]	A[4, 4, 2] = S [1538]	
...			
A[0, 4, 61] = S [1341]	A[1, 4, 61] = S [1405]	A[4, 4, 61] = S [1597]	
A[0, 4, 62] = S [1342]	A[1, 4, 62] = S [1406]	A[4, 4, 62] = S [1598]	
A[0, 4, 63] = S [1343]	A[1, 4, 63] = S [1407]	A[4, 4, 63] = S [1599]	

Figure 3.12: Demonstration of bits placement to state array.

3.3.1.4. Converting State Arrays to Strings

It is mentioned at 3.3.1.3. section that state array is denoted by A . After concatenating lanes and planes of A array S can be constructed. Interval of i and j can be represented as: $0 \leq i \leq 4$, $0 \leq j \leq 4$.

$$\text{Lane } (i, j) = A[i, j, 0] || A[i, j, 1] || \dots || A[i, j, w - 2] || A[i, j, w - 1] \quad (3.6)$$

Here b value is chosen 1600, so that w becomes 64, then each lane can be constructed as shown at figure 3.13.

$Lane(0, 0) = A[0, 0, 0] \parallel A[0, 0, 1] \parallel A[0, 0, 2] \parallel \dots \parallel A[0, 0, 62] \parallel A[0, 0, 63]$
 $Lane(1, 0) = A[1, 0, 0] \parallel A[1, 0, 1] \parallel A[1, 0, 2] \parallel \dots \parallel A[1, 0, 62] \parallel A[1, 0, 63]$
 $Lane(2, 0) = A[2, 0, 0] \parallel A[2, 0, 1] \parallel A[2, 0, 2] \parallel \dots \parallel A[2, 0, 62] \parallel A[2, 0, 63]$
 $Lane(3, 0) = A[3, 0, 0] \parallel A[3, 0, 1] \parallel A[3, 0, 2] \parallel \dots \parallel A[3, 0, 62] \parallel A[3, 0, 63]$
 $Lane(4, 0) = A[4, 0, 0] \parallel A[4, 0, 1] \parallel A[4, 0, 2] \parallel \dots \parallel A[4, 0, 62] \parallel A[4, 0, 63]$
 $Lane(0, 1) = A[0, 1, 0] \parallel A[0, 1, 1] \parallel A[0, 1, 2] \parallel \dots \parallel A[0, 1, 62] \parallel A[0, 1, 63]$
 $Lane(1, 1) = A[1, 1, 0] \parallel A[1, 1, 1] \parallel A[1, 1, 2] \parallel \dots \parallel A[1, 1, 62] \parallel A[1, 1, 63]$
 $Lane(2, 1) = A[2, 1, 0] \parallel A[2, 1, 1] \parallel A[2, 1, 2] \parallel \dots \parallel A[2, 1, 62] \parallel A[2, 1, 63]$
 $Lane(3, 1) = A[3, 1, 0] \parallel A[3, 1, 1] \parallel A[3, 1, 2] \parallel \dots \parallel A[3, 1, 62] \parallel A[3, 1, 63]$
 $Lane(4, 1) = A[4, 1, 0] \parallel A[4, 1, 1] \parallel A[4, 1, 2] \parallel \dots \parallel A[4, 1, 62] \parallel A[4, 1, 63]$
 $Lane(0, 2) = A[0, 2, 0] \parallel A[0, 2, 1] \parallel A[0, 2, 2] \parallel \dots \parallel A[0, 2, 62] \parallel A[0, 2, 63]$
 $Lane(1, 2) = A[1, 2, 0] \parallel A[1, 2, 1] \parallel A[1, 2, 2] \parallel \dots \parallel A[1, 2, 62] \parallel A[1, 2, 63]$
 $Lane(2, 2) = A[2, 2, 0] \parallel A[2, 2, 1] \parallel A[2, 2, 2] \parallel \dots \parallel A[2, 2, 62] \parallel A[2, 2, 63]$
 $Lane(3, 2) = A[3, 2, 0] \parallel A[3, 2, 1] \parallel A[3, 2, 2] \parallel \dots \parallel A[3, 2, 62] \parallel A[3, 2, 63]$
 $Lane(4, 2) = A[4, 2, 0] \parallel A[4, 2, 1] \parallel A[4, 2, 2] \parallel \dots \parallel A[4, 2, 62] \parallel A[4, 2, 63]$
 $Lane(0, 3) = A[0, 3, 0] \parallel A[0, 3, 1] \parallel A[0, 3, 2] \parallel \dots \parallel A[0, 3, 62] \parallel A[0, 3, 63]$
 $Lane(1, 3) = A[1, 3, 0] \parallel A[1, 3, 1] \parallel A[1, 3, 2] \parallel \dots \parallel A[1, 3, 62] \parallel A[1, 3, 63]$
 $Lane(2, 3) = A[2, 3, 0] \parallel A[2, 3, 1] \parallel A[2, 3, 2] \parallel \dots \parallel A[2, 3, 62] \parallel A[2, 3, 63]$
 $Lane(3, 3) = A[3, 3, 0] \parallel A[3, 3, 1] \parallel A[3, 3, 2] \parallel \dots \parallel A[3, 3, 62] \parallel A[3, 3, 63]$
 $Lane(4, 3) = A[4, 3, 0] \parallel A[4, 3, 1] \parallel A[4, 3, 2] \parallel \dots \parallel A[4, 3, 62] \parallel A[4, 3, 63]$
 $Lane(0, 4) = A[0, 4, 0] \parallel A[0, 4, 1] \parallel A[0, 4, 2] \parallel \dots \parallel A[0, 4, 62] \parallel A[0, 4, 63]$
 $Lane(1, 4) = A[1, 4, 0] \parallel A[1, 4, 1] \parallel A[1, 4, 2] \parallel \dots \parallel A[1, 4, 62] \parallel A[1, 4, 63]$
 $Lane(2, 4) = A[2, 4, 0] \parallel A[2, 4, 1] \parallel A[2, 4, 2] \parallel \dots \parallel A[2, 4, 62] \parallel A[2, 4, 63]$
 $Lane(3, 4) = A[3, 4, 0] \parallel A[3, 4, 1] \parallel A[3, 4, 2] \parallel \dots \parallel A[3, 4, 62] \parallel A[3, 4, 63]$
 $Lane(4, 4) = A[4, 4, 0] \parallel A[4, 4, 1] \parallel A[4, 4, 2] \parallel \dots \parallel A[4, 4, 62] \parallel A[4, 4, 63]$

Figure 3.13: Demonstration of lanes.

Constructing plane is realized as below:

$$Plane(j): Lane(0, j) \parallel Lane(1, j) \parallel Lane(2, j) \parallel Lane(3, j) \parallel Lane(4, j) \quad (3.7)$$

Finally, S can be constructed as shown at figure 3.14.

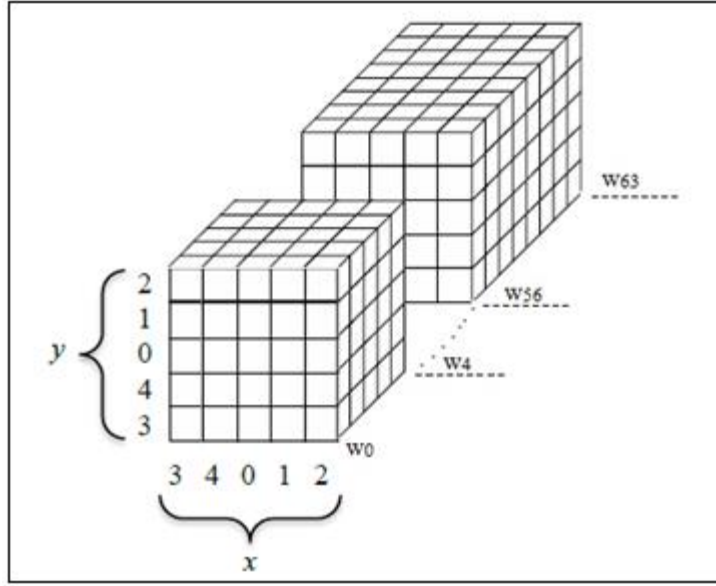


Figure 3.15: Demonstration of x, y and z coordinates of state array.

At figure 3.15 x,y,z coordinates are defined for the usage of step mapping process.

3.3.2. Step Mappings

At KECCAK-p[b, nr] permutation there are five steps namely θ , ρ , π , χ , and ι . These five step mappings(transformations) are comprising a round of KECCAK-p[b, nr].

3.3.2.1. Specification of θ

At figure 3.16 how θ function work is demonstrated. Its pseudo code steps are explained.

A : Input state array A' : Output state array Step 1: For all pairs (x, z) such that $0 \leq x \leq 4$ and $0 \leq z \leq w-1$ $C[x, z] = A[x, 0, z] \oplus A[x, 1, z] \oplus A[x, 2, z] \oplus A[x, 3, z] \oplus A[x, 4, z]$. Step 2: For all pairs (x, z) such that $0 \leq x \leq 4$ and $0 \leq z \leq w-1$ $D[x, z] = C[(x-1) \bmod 5, z] \oplus C[(x+1) \bmod 5, (z-1) \bmod w]$. Step 3: For all triples (x, y, z) such that $0 \leq x \leq 4$, $0 \leq y \leq 4$, and $0 \leq z \leq w-1$ $A'[x, y, z] = A[x, y, z] \oplus D[x, z]$.
--

Figure 3.16: For “ θ ” pseudo code steps.

The purpose of θ is to make XOR each bit in the state with the parities of two columns in the array. Especially, for the bit $A[x_0, y_0, z_0]$, the x-coordinate of one of the columns is $(x_0 - 1) \bmod 5$, with the same z-coordinate, z_0 , while the x-coordinate of the other column is $(x_0 + 1) \bmod 5$, with z-coordinate $(z_0 - 1) \bmod w$.

3.3.2.2. Specification of ρ

At figure 3.17 how θ function work is demonstrated. Its pseudo code steps are explained.

A : Input state array A' : Output state array Step 1: For all z such that $0 \leq z \leq w-1$, let $A'[0, 0, z] = A[0, 0, z]$. Step 2: Let $(x, y) = (1, 0)$. Step 3: For t from 0 to 23: a. for all z such that $0 \leq z \leq w-1$, let $A'[x, y, z] = A[x, y, (z - (t + 1)(t + 2)/2) \bmod w]$; b. let $(x, y) = (y, (2x + 3y) \bmod 5)$. Step 4: Return A'.
--

Figure 3.17: For “ ρ ” pseudo code steps.

The purpose of ρ is to rotate the bits of each lane by a length called the offset. Consequently, at the lane for each bit, the z coordinate is modified by adding the offset that is shown on table 3.11.

Table 3.11: Demonstration of ρ offset.

	x=3	x=4	x=0	x=1	x=2
y=2	153	231	3	10	171
y=1	55	276	36	300	6
y=0	28	91	0	1	190
y=4	120	78	210	66	253
y=3	21	136	105	45	15

3.3.2.3. Specification of π

At figure 3.18 how π function work is demonstrated. Its pseudo code steps are explained.

A : Input state array A' : Outpuy state array Step 1: For all triples (x, y, z) such that $0 \leq x \leq 4$, $0 \leq y \leq 4$, and $0 \leq z \leq w-1$, let $A'[x, y, z] = A[(x + 3y) \bmod 5, x, z]$. Step 2: Return A'.

Figure 3.18: For “ π ” pseudo code steps.

The purpose of π is to rearrange the positions of the lanes for any slice. For instance, the bit x=0 and y=0 is shown at the center of the slice.

3.3.2.4. Specification of χ

At figure 3.19 how χ function work is demonstrated. Its pseudo code steps are explained.

A : Input state array

A' : Output state array

Step 1:

For all triples (x, y, z) such that $0 \leq x \leq 4$, $0 \leq y \leq 4$, and $0 \leq z \leq w-1$, let

$A'[x, y, z] = A[x, y, z] \oplus ((A[(x+1) \bmod 5, y, z] \oplus 1) \cdot A[(x+2) \bmod 5, y, z])$.

Step 2:

Return A'.

Figure 3.19: For “ χ ” pseudo code steps.

The purpose of χ is to make XOR each bit with a non-linear function of two other bits in its row, as illustrated at figure 3.20[14].

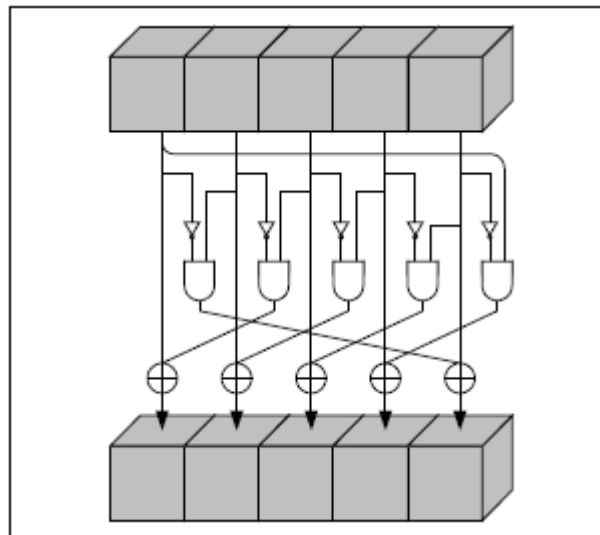


Figure 3.20: Demonstration of χ applied to a single row.

3.3.2.5. Specification of ι

While computing KECCAK-p[b,n_r], the ι mapping is parameterized by the round index i_r . At the $\iota(A, i_r)$ algorithm shown at figure 3.22, this i_r parameter determines $l + 1$ bits of a lane value called the *round constant*, denoted by RC . Each of these $l + 1$ bits are generated by a function that is based on a linear feedback shift register. This function is illustrated by rc and specified at Round Constant Algorithm shown at figure 3.21.

Round Constant Algorithm:

t : Input integer

$rc(t)$: Output bit

Step 1:

If $t \bmod 255 = 0$, return 1.

Step 2:

Let $R = 10000000$.

Step 3:

For i from 1 to $t \bmod 255$, let:

a. $R = 0 \parallel R$;

b. $R[0] = R[0] \oplus R[8]$;

c. $R[4] = R[4] \oplus R[8]$;

d. $R[5] = R[5] \oplus R[8]$;

e. $R[6] = R[6] \oplus R[8]$;

f. $R = \text{Trunc}_8[R]$.

Step 4:

Return $R[0]$.

Figure 3.21: For “RC” pseudo code steps.

$\mathfrak{t}(\mathbf{A}, ir)$ Algorithm:

$\mathbf{A}$: Input state array

ir : Round index

$\mathbf{A}'$: Output state array

Step 1:

For all triples (x, y, z) : $0 \leq x \leq 4$, $0 \leq y \leq 4$, and $0 \leq z \leq w-1$, let $\mathbf{A}'[x, y, z] = \mathbf{A}[x, y, z]$.

Step 2:

Let $RC = 0^w$.

Step 3:

For j from 0 to l , let $RC[2j-1] = rc(j + 7ir)$.

Step 4:

For all z such that $0 \leq z < w$, let $\mathbf{A}'[0, 0, z] = \mathbf{A}'[0, 0, z] \oplus RC[z]$.

Step 5:

Return $\mathbf{A}'$.

Figure 3.22: For “ $\mathfrak{t}$ ” pseudo code steps.

The purpose of $\mathfrak{t}$ is to modify some of the bits of Lane(0,0) in a manner that depends on the round index ir . The other 24 lanes are not affected by $\mathfrak{t}$.

3.3.3. Keccak – permutation $[b, n_r]$

Round function Rnd consists of a state array $\mathbf{A}$ and a round index ir . This Rnd function is result of the five step mappings that called as θ , ρ , π , χ , and $\mathfrak{t}$.

$$\text{Rnd}(\mathbf{A}, ir) = \mathfrak{t}(\chi(\pi(\rho(\theta(\mathbf{A})))), ir) \quad (3.8)$$

The KECCAK - $p[b, n_r]$ permutation is the result of n_r iterations of Rnd shown at figure 3.23.

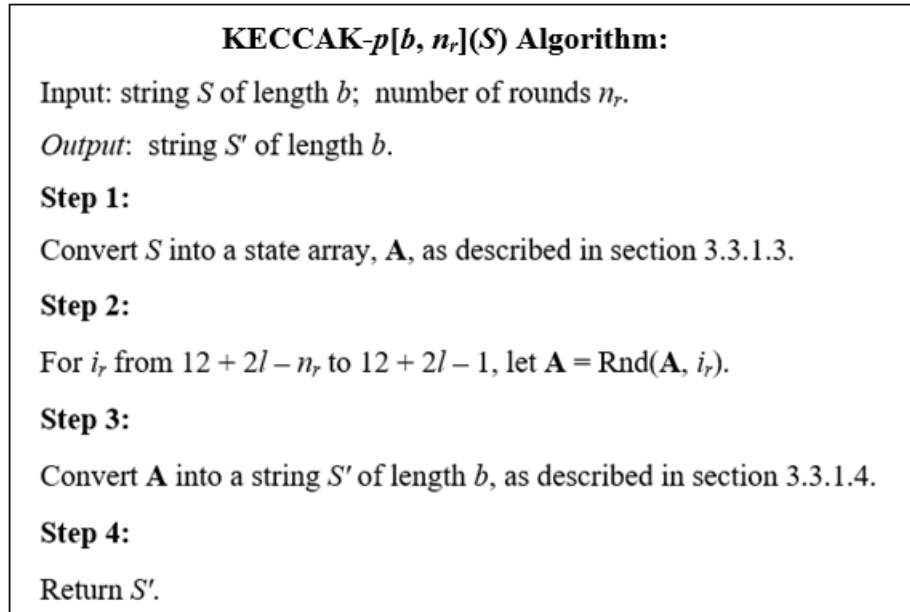


Figure 3.23: For “Keccak – permutation” pseudo code steps.

3.3.4. Sponge Construction

The sponge construction is a structure for specifying functions on binary data with arbitrary output length. The structure has following three components:

1. A function on fixed-length strings, demonstrated by f .
2. A parameter named as rate, demonstrated by r .
3. A padding rule, demonstrated by pad .

Moreover this sponge function has two inputs. One of them is a bit string that demonstrated by N . The other one is bit length of output string, demonstrated by d . As a result, sponge function is illustrated like this: $\text{SPONGE}[f, \text{pad}, r](N, d)$. Firstly, arbitrary number of input bits are absorbed into the state of the function. After that, arbitrary number of output bits are squeezed out of its state[14]. The sponge construction is shown at figure 3.24.

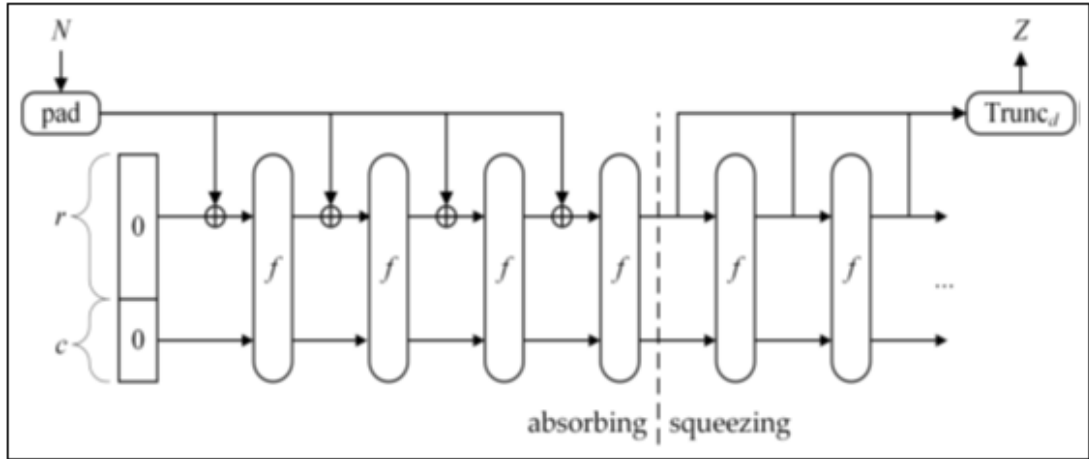


Figure 3.24: The sponge construction.

The sponge function f is invertible. The r that abbreviation of rate is a positive integer and less than b value. The c that abbreviation of capacity is equal to $(b-r)$ and also a positive integer. The pad that abbreviation of padding rule is a function that makes padding operation. In general, x is a positive integer and m is a non-negative integer and the output $\text{pad}(x, m)$. $\text{Length}(\text{pad}(x, m))$ plus m results a positive multiple of x . Here x equals to rate value, and m equals to length of N value. Consequently, the $\text{SPONGE}[f, \text{pad}, r]$ function on (N, d) is specified as at figure 3.25.

Input: string N , nonnegative integer d .
Output: string Z such that length of Z equals to d .

Step 1:
Let $P = N \parallel \text{pad}(r, \text{len}(N))$.

Step 2:
Let $n = \text{len}(P)/r$.

Step 3:
Let $c = b - r$.

Step 4:
Let $P_0, \dots, P_{n-1}$ be unique sequence of strings of length r such that $P = P_0 \parallel \dots \parallel P_{n-1}$.

Step 5:
Let $S = 0^b$.

Step 6:
For i from 0 to $n-1$, let $S = f(S \oplus (P_i \parallel 0^c))$.

Step 7:
Let Z be the empty string.

Step 8:
Let $Z = Z \parallel \text{Trunc}_r(S)$.

Step 9:
If $d \leq |Z|$, then return $\text{Trunc}_d(Z)$; else continue.

Step 10:
Let $S = f(S)$, and continue with Step 8.

Figure 3.25: For “sponge construction” pseudo code steps.

3.3.5. Keccak

KECCAK can be denoted as a family of sponge functions. The padding rule for KECCAK, defined as multi-rate padding. Related algorithm $\text{pad}_{10^*1}(x, m)$ is illustrated at figure 3.26.

Input: positive integer x , non-negative integer m .

Output: string P such that $m + \text{len}(P)$ is a positive multiple of x .

Step 1:

Let $j = (-m - 2) \bmod x$.

Step 2:

2. Return $P = 1 \parallel 0^j \parallel 1$.

Figure 3.26: For “padding rule” pseudo code steps.

The sign * at “pad10*1” indicates that “0” bit is can be omitted or repeated as necessary in order to produce an output string of the desired length.

Moreover, if b value is restricted to 1600 that can be 25, 50, 100, 200, 400, 800, 1600 then the KECCAK family is demonstrated as $\text{KECCAK}[c]$.

$$\text{KECCAK}[c] = \text{SPONGE}[\text{KECCAK} - p[1600, 24], \text{pad10} * 1, 1600 - c] \quad (3.9)$$

Here if an input bit string chosen as N and output length as d ;

$$\text{KECCAK}[c] (N, d) = \text{SPONGE}[\text{Kec-}p[1600, 24], \text{pad10} * 1, 1600 - c] (N, d).$$

3.3.5.1.SHA3 Hash Functions

Here SHA3 can be derived from $\text{KECCAK}[c] (N, d)$ as shown at table 3.12:

Table 3.12: Illustration of $\text{KECCAK}[c] (N, d)$.

$\text{SHA3-224}(M) =$	$\text{KECCAK}[448] (M \parallel 01, 224)$
$\text{SHA3-256}(M) =$	$\text{KECCAK}[512] (M \parallel 01, 256)$
$\text{SHA3-384}(M) =$	$\text{KECCAK}[768] (M \parallel 01, 384)$
$\text{SHA3-512}(M) =$	$\text{KECCAK}[1024] (M \parallel 01, 512)$

At every time the the digest length is half of capacity value. As shown above, 01 bits are concatenated to messeage M . As a resut of this operation SHA3 operates properly.

3.3.5.2. SHA3 Extendable-Output Functions

Two SHA3 XOF that abbreviation of Extendable-Output Functions (SHAKE128 and SHAKE256) are defined from KECCAK[c] as shown at table 3.13.

Table 3.13: Illustration of SHAKE.

$\text{SHAKE128}(M, d) =$	$\text{KECCAK}[256] (M \parallel 1111, d)$
$\text{SHAKE256}(M, d) =$	$\text{KECCAK}[512] (M \parallel 1111, d)$

1111 is concatenated to message M. As a result of this operation SHA3 XOF operates properly.

4. EXPERIMENTAL RESULTS

4.1. ARM Cortex-4 Microprocessor

In this study, performance of hash algorithms for the same input message will be evaluated using ARM Cortex-M4 32-bit STM32F407VG microprocessor. This microprocessor can operate up to 168 MHz frequency. Its flash memory can store up to 1 Mbyte and SRAM can store up to 192 Kbytes. It has two APB and three AHB buses for inputs and outputs and peripherals.

Many features available in this microprocessor such as ADCs, DACs, SPIs etc. are not used in this work. For this reason, only UARTs and timers to be mentioned which are used in this study.

This microprocessor has twelve general-purpose 16-bit timers and two general-purpose 32-bit timers. It also has four USARTs and 2 UARTs.

TIM2 that is one of the 32-bit timers will be used at interrupt mode to measure the performance of hash algorithms and USART2 is also used to demonstrate timer value at terminal program. At figure 4.1 TIM2 and USART2 which are connected to APB1 bus are shown in red rectangulars[12].

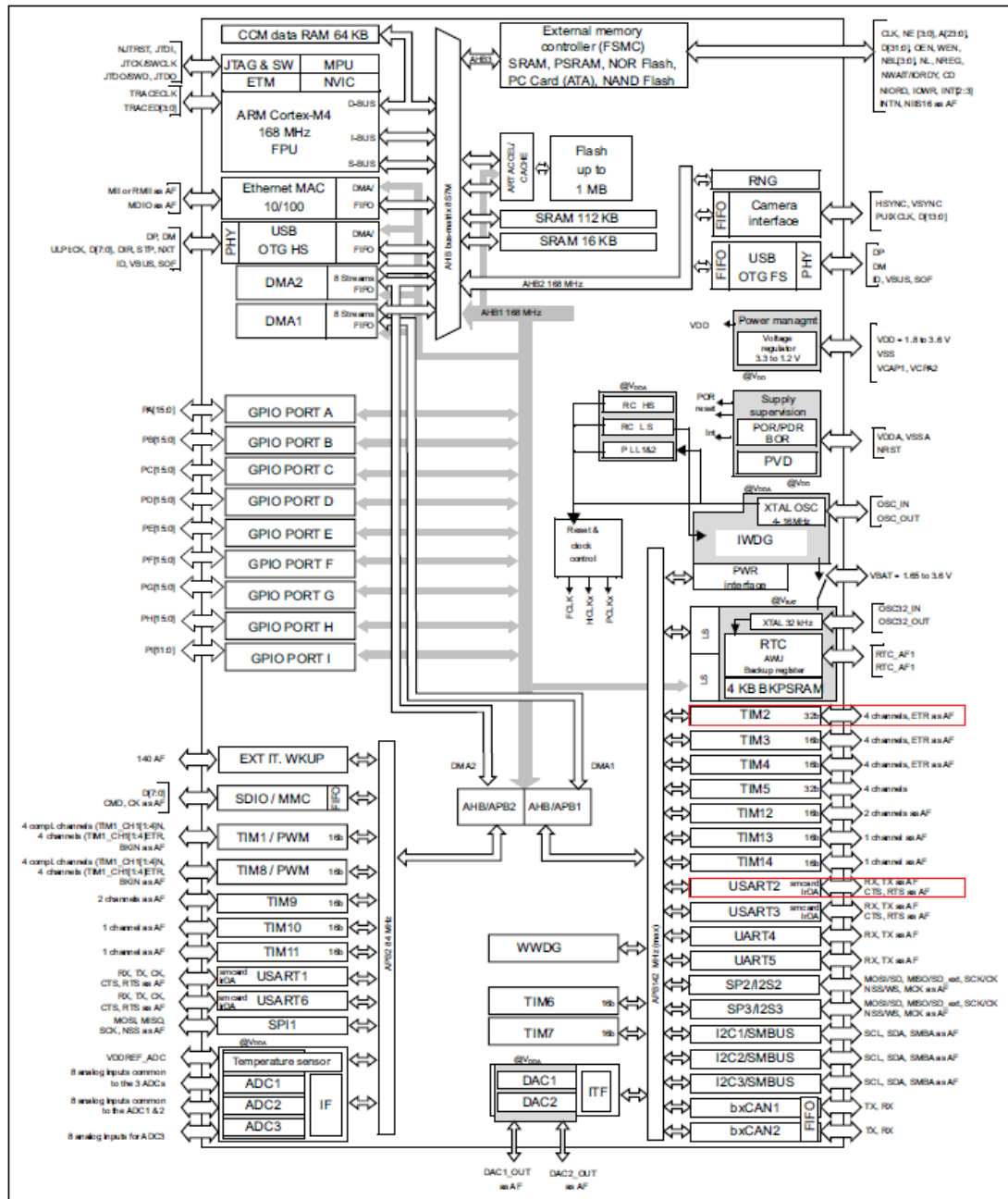


Figure 4.1: STM32F407VG microprocessor block diagram.

To program a microprocessor linker is needed. As shown at figure 4.2 STM32F4 DISCOVERY module is used in this work because of its ST-LINK/V2 embedded debug tool feature[13]. Hash algorithms are coded using C language on Keil IDE. Moreover for configuration of 32-bit timer TIM2 STM32CubeMX that is shown figure 4.3 is used.

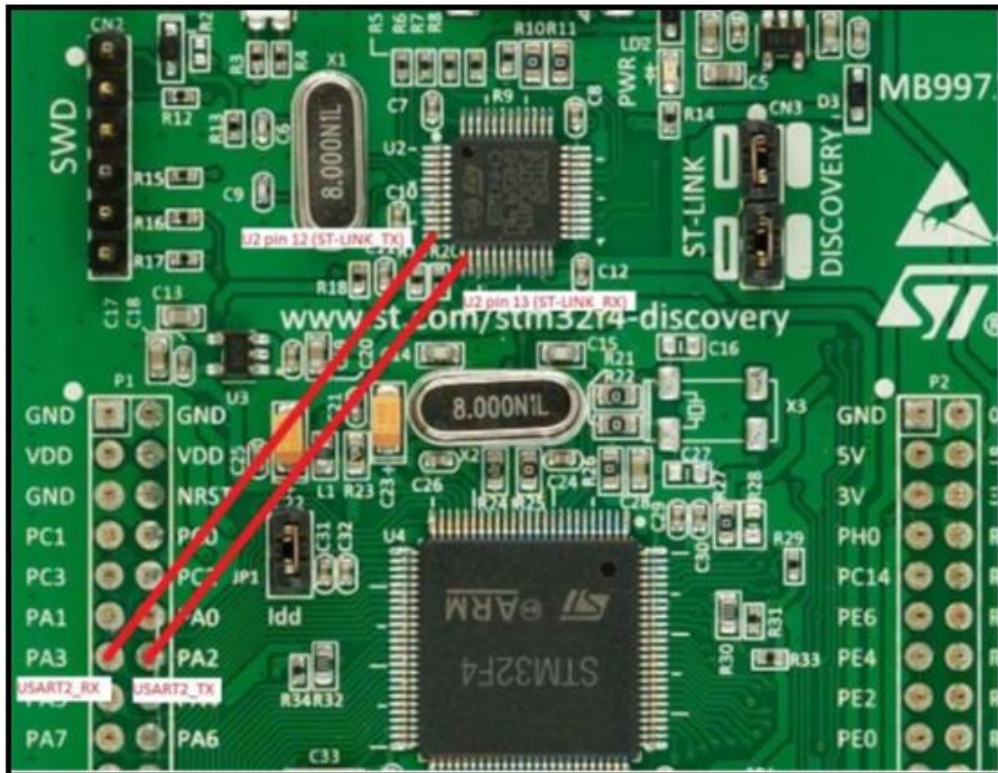


Figure 4.2: STM32F4DISCOVERY microprocessor board.

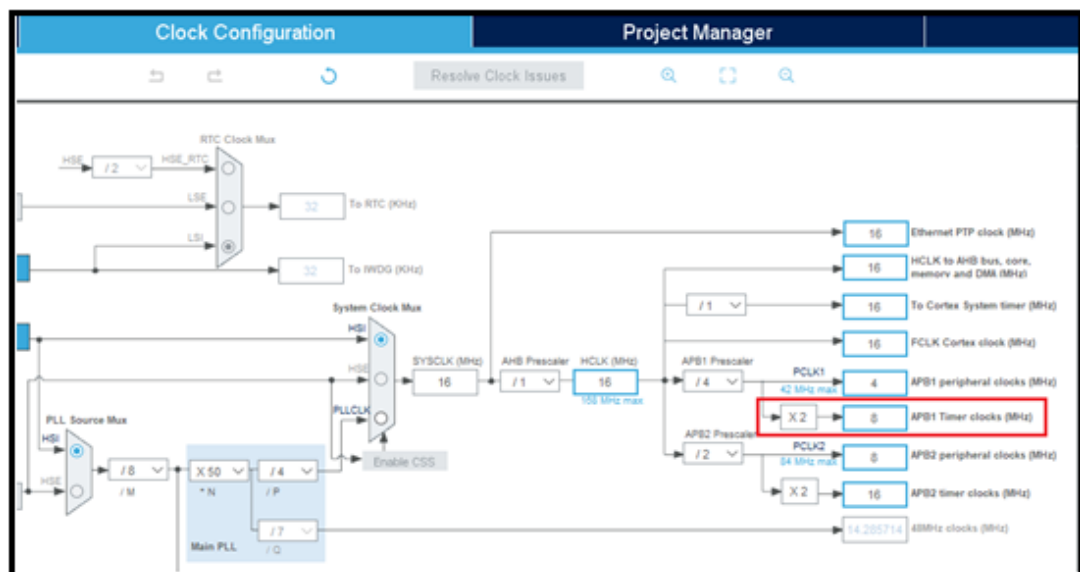


Figure 4.3: Timer clock configuration via STM32CubeMX.

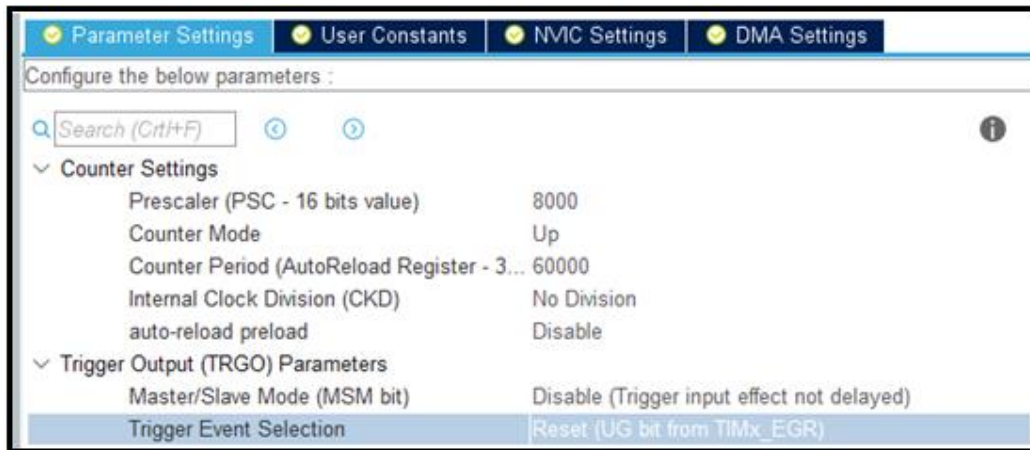


Figure 4.4: Timer detail configuration via STM32CubeMX.

TIM2 timer is configured to operate at 8 MHz frequency that is shown at figure 4.4. Prescaler value is chosen as 8000 that can be 0 to 65565. By configuring prescaler value as 8000 TIM2 frequency is decreased to 1KHz which was 8 MHz before. After that counter period is chosen as 60000 that can be 0 to 2^{32} . By this configuration of counter period as 60000 the timer TIM2 will count until one minute is completed. While programming hash algorithms the counter value is configured to zero like this: TIM2->CNT = 0. After that hash algorithm start processing. At the end of hash function counter value is stored to a 16-bit variable like this: value = (TIM2->CNT).

The value demonstrate performance of that hash algorithm. By using this method performance of all hash algorithms can be stored and then can be compared.

4.2. Performance Results

4.2.1. Rounds of SHA-1 and SHA-2

Firstly input message is selected as “GTU”. After that all rounds and message digests are shown at figure 4.5, figure 4.6, figure 4.7, figure 4.8, figure 4.9, figure 4.10, figure 4.11 respectively for SHA-1 SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256. For these hash algorithms all rounds and message digests is get by using Code::blocks IDE. Although this study is related to compare hash algorithms on ARM microprocessor, by compiling all codes at the Code::blocks IDE the rounds are also obtained and shown.

	a	b	c	d	e
Round 0	:e708ee33	67452301	7bf36ae2	98badcfe	10325476
Round 1	:47ce9389	e708ee33	59d148c0	7bf36ae2	98badcfe
Round 2	:4703107f	47ce9389	f9c23b8c	59d148c0	7bf36ae2
Round 3	:10ab502b	4703107f	51f3a4e2	f9c23b8c	59d148c0
Round 4	:c380f39d	10ab502b	d1c0c41f	51f3a4e2	f9c23b8c
Round 5	:16340da8	c380f39d	c42ad40a	d1c0c41f	51f3a4e2
Round 6	:4338a787	16340da8	70e03ce7	c42ad40a	d1c0c41f
Round 7	:63830b42	4338a787	58d036a	70e03ce7	c42ad40a
Round 8	:c0d6d151	63830b42	d0ce29e1	58d036a	70e03ce7
Round 9	:2acaea20	c0d6d151	98e0c2d0	d0ce29e1	58d036a
Round 10	:4a35a9f8	2acaea20	7035b454	98e0c2d0	d0ce29e1
Round 11	:22268353	4a35a9f8	ab2ba88	7035b454	98e0c2d0
Round 12	:72646359	22268353	128d6a7e	ab2ba88	7035b454
Round 13	:21d8d3f5	72646359	c889a0d4	128d6a7e	ab2ba88
Round 14	:e0d8db3b	21d8d3f5	5c9918d6	c889a0d4	128d6a7e
Round 15	:50c47c7f	e0d8db3b	487634fd	5c9918d6	c889a0d4
Round 16	:26956654	50c47c7f	f83636ce	487634fd	5c9918d6
Round 17	:e1fe91c1	26956654	d4311f1f	f83636ce	487634fd
Round 18	:befdfda0	e1fe91c1	9a55995	d4311f1f	f83636ce
Round 19	:656eda1e	befdfda0	787fa470	9a55995	d4311f1f
Round 20	:c00d4ed1	656eda1e	2fbf7f68	787fa470	9a55995
Round 21	:acd720d4	c00d4ed1	995bb687	2fbf7f68	787fa470
Round 22	:33c9dde6	acd720d4	700353b4	995bb687	2fbf7f68
Round 23	:5d64ede6	33c9dde6	2b35c835	700353b4	995bb687
Round 24	:3b23fc1b	5d64ede6	8cf27779	2b35c835	700353b4
Round 25	:b3456d6a	3b23fc1b	97593b79	8cf27779	2b35c835
Round 26	:23461147	b3456d6a	cec8ff06	97593b79	8cf27779
Round 27	:4f633693	23461147	acd15b5a	cec8ff06	97593b79
Round 28	:1e845ea6	4f633693	c8d18451	acd15b5a	cec8ff06
Round 29	:3992a9f2	1e845ea6	d3d8cda4	c8d18451	acd15b5a
Round 30	:bba1a99a	3992a9f2	87a117a9	d3d8cda4	c8d18451
Round 31	:eee17789	bba1a99a	8e64aa7c	87a117a9	d3d8cda4
Round 32	:f939b8d4	eee17789	ae86a66	8e64aa7c	87a117a9
Round 33	:ec1fdb7c	f939b8d4	7bb85de2	ae86a66	8e64aa7c
Round 34	:57ce55ed	ec1fdb7c	3e4e6e35	7bb85de2	ae86a66
Round 35	:36bc5820	57ce55ed	3b07f6df	3e4e6e35	7bb85de2
Round 36	:b4f54ea5	36bc5820	55f3957b	3b07f6df	3e4e6e35
Round 37	:f86fea97	b4f54ea5	daf1608	55f3957b	3b07f6df
Round 38	:19ce5b59	f86fea97	6d3d53a9	daf1608	55f3957b
Round 39	:9796b435	19ce5b59	fe1bfaa5	6d3d53a9	daf1608
Round 40	:eaaef0dc9	9796b435	467396d6	fe1bfaa5	6d3d53a9
Round 41	:284e8f77	eaaef0dc9	65e5ad0d	467396d6	fe1bfaa5
Round 42	:7f320489	284e8f77	7aabc372	65e5ad0d	467396d6
Round 43	:761578d5	7f320489	ca13a3dd	7aabc372	65e5ad0d
Round 44	:778114a0	761578d5	5fcc8122	ca13a3dd	7aabc372
Round 45	:580016f1	778114a0	5d855e35	5fcc8122	ca13a3dd
Round 46	:90a0aa02	580016f1	1de04528	5d855e35	5fcc8122
Round 47	:818eb9e4	90a0aa02	560005bc	1de04528	5d855e35
Round 48	:d386621d	818eb9e4	a4282a80	560005bc	1de04528
Round 49	:e72877d7	d386621d	2063ae79	a4282a80	560005bc
Round 50	:bea26b34	e72877d7	74e19887	2063ae79	a4282a80
Round 51	:c44c19f	bea26b34	f9ca1df5	74e19887	2063ae79
Round 52	:e09a8994	c44c19f	2fa89acd	f9ca1df5	74e19887
Round 53	:451817dc	e09a8994	c3113067	2fa89acd	f9ca1df5
Round 54	:cda93469	451817dc	3826a265	c3113067	2fa89acd
Round 55	:ca5b5b5c	cda93469	114605f7	3826a265	c3113067
Round 56	:b03f7452	ca5b5b5c	736a4d1a	114605f7	3826a265
Round 57	:76cec37c	b03f7452	3296d6d7	736a4d1a	114605f7
Round 58	:b848be9c	76cec37c	ac0fdd14	3296d6d7	736a4d1a
Round 59	:533b03d3	b848be9c	1db3b0df	ac0fdd14	3296d6d7
Round 60	:495cf42e	533b03d3	2e122fa7	1db3b0df	ac0fdd14
Round 61	:582c49f2	495cf42e	d4cec0f4	2e122fa7	1db3b0df
Round 62	:527d7ca4	582c49f2	92573d0b	d4cec0f4	2e122fa7
Round 63	:bb27839b	527d7ca4	960b127c	92573d0b	d4cec0f4
Round 64	:abb3cc0	bb27839b	149f5f29	960b127c	92573d0b
Round 65	:ede465b0	abb3cc0	eec9e0e6	149f5f29	960b127c
Round 66	:4eb867ff	ede465b0	2aecf30	eec9e0e6	149f5f29
Round 67	:b9e46864	4eb867ff	3b79196c	2aecf30	eec9e0e6
Round 68	:a35913b	b9e46864	d3ae19ff	3b79196c	2aecf30
Round 69	:6557e15e	a35913b	2e791a19	d3ae19ff	3b79196c
Round 70	:921198c2	6557e15e	c28d644e	2e791a19	d3ae19ff
Round 71	:7accf5d1	921198c2	9955f857	c28d644e	2e791a19
Round 72	:a0d37fa1	7accf5d1	a4846630	9955f857	c28d644e
Round 73	:4685ff53	a0d37fa1	5eb33d74	a4846630	9955f857
Round 74	:9132cd8d	4685ff53	6834dfe8	5eb33d74	a4846630
Round 75	:89e4b738	9132cd8d	d1a17fd4	6834dfe8	5eb33d74
Round 76	:ed043ddb	89e4b738	644cb363	d1a17fd4	5eb33d74
Round 77	:101ee2ca	ed043ddb	22792dce	644cb363	d1a17fd4
Round 78	:24cc463c	101ee2ca	fb410f76	22792dce	644cb363
Round 79	:c5708301	24cc463c	8407b8b2	fb410f76	22792dce
Input Message :GTU					
SHA-1 HASH Result :2cb5a6021499f1c51cc295b00b7363ece64c0fbe					

Figure 4.5: All rounds and message digests for SHA-1.

	a	b	c	d	e	f	g	h
Round[0]:	f488a4da	c1059ed8	367cd507	3070dd17	ea26145e	ffc00b31	68581511	64f98fa7
Round[1]:	517c44aa	f488a4da	c1059ed8	367cd507	d479a646	ea26145e	ffc00b31	68581511
Round[2]:	a3d2185	517c44aa	f488a4da	c1059ed8	2f8e3e25	d479a646	ea26145e	ffc00b31
Round[3]:	4f001b06	a3d2185	517c44aa	f488a4da	5a887b4	2f8e3e25	d479a646	ea26145e
Round[4]:	830633ae	4f001b06	a3d2185	517c44aa	e0b46f05	5a887b4	2f8e3e25	d479a646
Round[5]:	1b9d7778	830633ae	4f001b06	a3d2185	3c9a59c6	e0b46f05	5a887b4	2f8e3e25
Round[6]:	24963db8	1b9d7778	830633ae	4f001b06	5ad4cab4	3c9a59c6	e0b46f05	5a887b4
Round[7]:	16a32ab1	24963db8	1b9d7778	830633ae	a5fac2b2	5ad4cab4	3c9a59c6	e0b46f05
Round[8]:	e650fe38	16a32ab1	24963db8	1b9d7778	b635363f	a5fac2b2	5ad4cab4	3c9a59c6
Round[9]:	1201ba66	e650fe38	16a32ab1	24963db8	7b610496	b635363f	a5fac2b2	5ad4cab4
Round[10]:	18bddac3	1201ba66	e650fe38	16a32ab1	d5f8f76f	7b610496	b635363f	a5fac2b2
Round[11]:	16b1e952	18bddac3	1201ba66	e650fe38	33e65a65	d5f8f76f	7b610496	b635363f
Round[12]:	c96543a8	16b1e952	18bddac3	1201ba66	946ac01d	33e65a65	d5f8f76f	7b610496
Round[13]:	617cdb34	c96543a8	16b1e952	18bddac3	a0b710f3	946ac01d	33e65a65	d5f8f76f
Round[14]:	4bcb37a	617cdb34	c96543a8	16b1e952	a973d65f	a0b710f3	946ac01d	33e65a65
Round[15]:	e028df9c	4bcb37a	617cdb34	c96543a8	cd2b148d	a973d65f	a0b710f3	946ac01d
Round[16]:	64e90410	e028df9c	4bcb37a	617cdb34	667e275b	cd2b148d	a973d65f	a0b710f3
Round[17]:	19e88667	64e90410	e028df9c	4bcb37a	7913a2a4	667e275b	cd2b148d	a973d65f
Round[18]:	42cfe013	19e88667	64e90410	e028df9c	8f489abf	7913a2a4	667e275b	cd2b148d
Round[19]:	f8831e44	42cfe013	19e88667	64e90410	a818d65d	8f489abf	7913a2a4	667e275b
Round[20]:	f9036813	f8831e44	42cfe013	19e88667	44a2d571	a818d65d	8f489abf	7913a2a4
Round[21]:	36a86fe8	f9036813	f8831e44	42cfe013	a48e3041	44a2d571	a818d65d	8f489abf
Round[22]:	3fbbcbca	36a86fe8	f9036813	f8831e44	b6b42f5a	a48e3041	44a2d571	a818d65d
Round[23]:	2745c44d	3fbbcbca	36a86fe8	f9036813	a08ccced	b6b42f5a	a48e3041	44a2d571
Round[24]:	bff99319a	2745c44d	3fbbcbca	36a86fe8	4494a25	a08ccced	b6b42f5a	a48e3041
Round[25]:	e7c7475d	bff99319a	2745c44d	3fbbcbca	96e01b2a	4494a25	a08ccced	b6b42f5a
Round[26]:	eb323fd9	e7c7475d	bff99319a	2745c44d	25252df4	96e01b2a	4494a25	a08ccced
Round[27]:	f1c21c02	eb323fd9	e7c7475d	bff99319a	dc77f6ab	25252df4	96e01b2a	4494a25
Round[28]:	842f8d72	f1c21c02	eb323fd9	e7c7475d	bef9c9cb	dc77f6ab	25252df4	96e01b2a
Round[29]:	3b4a7730	842f8d72	f1c21c02	eb323fd9	cd45990b	bef9c9cb	dc77f6ab	25252df4
Round[30]:	e7d26434	3b4a7730	842f8d72	f1c21c02	832aff69	cd45990b	bef9c9cb	dc77f6ab
Round[31]:	388f8f40	e7d26434	3b4a7730	842f8d72	3143d211	832aff69	cd45990b	bef9c9cb
Round[32]:	63c9ec0	388f8f40	e7d26434	3b4a7730	4829db4	3143d211	832aff69	cd45990b
Round[33]:	603c92ad	63c9ec0	388f8f40	e7d26434	6ef46551	4829db4	3143d211	832aff69
Round[34]:	14c112b8	603c92ad	63c9ec0	388f8f40	9d28475d	6ef46551	4829db4	3143d211
Round[35]:	8dc11ade	14c112b8	603c92ad	63c9ec0	2d5a1481	9d28475d	6ef46551	4829db4
Round[36]:	371c43ea	8dc11ade	14c112b8	603c92ad	c6a87d66	2d5a1481	9d28475d	6ef46551
Round[37]:	d35d287	371c43ea	8dc11ade	14c112b8	74185176	c6a87d66	2d5a1481	9d28475d
Round[38]:	45e248d6	d35d287	371c43ea	8dc11ade	cd4f0785	74185176	c6a87d66	2d5a1481
Round[39]:	fc980e83	45e248d6	d35d287	371c43ea	263b026b	cd4f0785	74185176	c6a87d66
Round[40]:	8650f64c	fc980e83	45e248d6	d35d287	84b9071d	263b026b	cd4f0785	74185176
Round[41]:	710929ea	8650f64c	fc980e83	45e248d6	e945709e	84b9071d	263b026b	cd4f0785
Round[42]:	b5df3934	710929ea	8650f64c	fc980e83	ff2e749	e945709e	84b9071d	263b026b
Round[43]:	e92b53aa	b5df3934	710929ea	8650f64c	9833765e	ff2e749	e945709e	84b9071d
Round[44]:	99ae2e86	e92b53aa	b5df3934	710929ea	a4a0b514	9833765e	ff2e749	e945709e
Round[45]:	9369b934	99ae2e86	e92b53aa	b5df3934	dfde4ac2	a4a0b514	9833765e	ff2e749
Round[46]:	fa8c1520	9369b934	99ae2e86	e92b53aa	cba5eb61	dfde4ac2	a4a0b514	9833765e
Round[47]:	5e9f6988	fa8c1520	9369b934	99ae2e86	42e2d4c	cba5eb61	dfde4ac2	a4a0b514
Round[48]:	49a76bff	5e9f6988	fa8c1520	9369b934	e2855182	42e2d4c	cba5eb61	dfde4ac2
Round[49]:	b3112b78	49a76bff	5e9f6988	fa8c1520	dbaf1022	e2855182	42e2d4c	cba5eb61
Round[50]:	c0dc7c4f	b3112b78	49a76bff	5e9f6988	2c24f4dc	dbaf1022	e2855182	42e2d4c
Round[51]:	57ab204f	c0dc7c4f	b3112b78	49a76bff	90fcf765	2c24f4dc	dbaf1022	e2855182
Round[52]:	8915487c	57ab204f	c0dc7c4f	b3112b78	84124418	90fcf765	2c24f4dc	dbaf1022
Round[53]:	caf56b9a	8915487c	57ab204f	c0dc7c4f	87e84632	84124418	90fcf765	2c24f4dc
Round[54]:	9a556eb8	caf56b9a	8915487c	57ab204f	53b71c43	87e84632	84124418	90fcf765
Round[55]:	dd094903	9a556eb8	caf56b9a	8915487c	a373932e	53b71c43	87e84632	84124418
Round[56]:	d1c5e58e	dd094903	9a556eb8	caf56b9a	e60d0cf2	a373932e	53b71c43	87e84632
Round[57]:	fdca19d5	d1c5e58e	dd094903	9a556eb8	5fe84bda	e60d0cf2	a373932e	53b71c43
Round[58]:	33a9c175	fdca19d5	d1c5e58e	dd094903	567ba6d4	5fe84bda	e60d0cf2	a373932e
Round[59]:	573792f7	33a9c175	fdca19d5	d1c5e58e	6230e048	567ba6d4	5fe84bda	e60d0cf2
Round[60]:	1aa48902	573792f7	33a9c175	fdca19d5	d8825c3e	6230e048	567ba6d4	5fe84bda
Round[61]:	2425e032	1aa48902	573792f7	33a9c175	b1ac7982	d8825c3e	6230e048	567ba6d4
Round[62]:	d0a680c8	2425e032	1aa48902	573792f7	cf123058	b1ac7982	d8825c3e	6230e048
Round[63]:	6df62d8d	d0a680c8	2425e032	1aa48902	c1d3a3e	cf123058	b1ac7982	d8825c3e
Input Message	:0TU							
SHA-224 HASH Result	:2efbccc65072355cf5496bd4911b2e23b0bdd456f376a456916a60929							

Figure 4.6: All rounds and message digests for SHA-224.

	a	b	c	d	e	f	g	h
Round[0]:	435cddcd	6a09e667	bb67ae85	3c6ef372	e01c3822	510e527f	9b05688c	1f83d9ab
Round[1]:	a6a0d85e	435cddcd	6a09e667	bb67ae85	a9f7c473	e01c3822	510e527f	9b05688c
Round[2]:	4658a4a0	a6a0d85e	435cddcd	6a09e667	b77afd4b	a9f7c473	e01c3822	510e527f
Round[3]:	f3700bb8	4658a4a0	a6a0d85e	435cddcd	c11a7a5f	b77afd4b	a9f7c473	e01c3822
Round[4]:	4da5f89	f3700bb8	4658a4a0	a6a0d85e	c09139fb	c11a7a5f	b77afd4b	a9f7c473
Round[5]:	aea322aa	4da5f89	f3700bb8	4658a4a0	3aeb563d	c09139fb	c11a7a5f	b77afd4b
Round[6]:	2b9cc9ca	aea322aa	4da5f89	f3700bb8	988c4bb9	3aeb563d	c09139fb	c11a7a5f
Round[7]:	8270d9de	2b9cc9ca	aea322aa	4da5f89	8fb553d2	988c4bb9	3aeb563d	c09139fb
Round[8]:	93ee20b4	8270d9de	2b9cc9ca	aea322aa	4108567b	8fb553d2	988c4bb9	3aeb563d
Round[9]:	ab14d9f4	93ee20b4	8270d9de	2b9cc9ca	3bdd6dad	4108567b	8fb553d2	988c4bb9
Round[10]:	6b6196e3	ab14d9f4	93ee20b4	8270d9de	5d81ca42	3bdd6dad	4108567b	8fb553d2
Round[11]:	17210819	6b6196e3	ab14d9f4	93ee20b4	2949dea	5d81ca42	3bdd6dad	4108567b
Round[12]:	f8d7ca5e	17210819	6b6196e3	ab14d9f4	e083b2cf	2949dea	5d81ca42	3bdd6dad
Round[13]:	31314be7	f8d7ca5e	17210819	6b6196e3	acf94bb6	e083b2cf	2949dea	5d81ca42
Round[14]:	f4329b1e	31314be7	f8d7ca5e	17210819	d8089feb	acf94bb6	e083b2cf	2949dea
Round[15]:	62dbd97d	f4329b1e	31314be7	f8d7ca5e	da3199b5	d8089feb	acf94bb6	e083b2cf
Round[16]:	6ec798f4	62dbd97d	f4329b1e	31314be7	fb3372c9	da3199b5	d8089feb	acf94bb6
Round[17]:	87afd6bd	6ec798f4	62dbd97d	f4329b1e	8f9c4c1e	fb3372c9	da3199b5	d8089feb
Round[18]:	7e1f2e5	87afd6bd	6ec798f4	62dbd97d	8f5a36cb	8f9c4c1e	fb3372c9	da3199b5
Round[19]:	10d31c39	7e1f2e5	87afd6bd	6ec798f4	1aab4328	8f5a36cb	8f9c4c1e	fb3372c9
Round[20]:	fb3c1644	10d31c39	7e1f2e5	87afd6bd	78932ef6	1aab4328	8f5a36cb	8f9c4c1e
Round[21]:	5abcead5	fb3c1644	10d31c39	7e1f2e5	51c9e040	78932ef6	1aab4328	8f5a36cb
Round[22]:	b46fc5ab	5abcead5	fb3c1644	10d31c39	6de6e003	51c9e040	78932ef6	1aab4328
Round[23]:	24835af8	b46fc5ab	5abcead5	fb3c1644	bbc8b3a7	6de6e003	51c9e040	78932ef6
Round[24]:	719bfe54	24835af8	b46fc5ab	5abcead5	649e3769	bbc8b3a7	6de6e003	51c9e040
Round[25]:	4df881af	719bfe54	24835af8	b46fc5ab	f2ed6b00	649e3769	bbc8b3a7	6de6e003
Round[26]:	d9d6645d	4df881af	719bfe54	24835af8	eca75c74	f2ed6b00	649e3769	bbc8b3a7
Round[27]:	25a2a62c	d9d6645d	4df881af	719bfe54	e340fc04	eca75c74	f2ed6b00	649e3769
Round[28]:	88b5c6f3	25a2a62c	d9d6645d	4df881af	f9cded12	e340fc04	eca75c74	f2ed6b00
Round[29]:	79d6a2dd	88b5c6f3	25a2a62c	d9d6645d	3b6d43df	f9cded12	e340fc04	eca75c74
Round[30]:	8047a365	79d6a2dd	88b5c6f3	25a2a62c	1e524ce0	3b6d43df	f9cded12	e340fc04
Round[31]:	fd49c89	8047a365	79d6a2dd	88b5c6f3	350f22db	1e524ce0	3b6d43df	f9cded12
Round[32]:	bbe1aba	fd49c89	8047a365	79d6a2dd	515c4f6f	350f22db	1e524ce0	3b6d43df
Round[33]:	4dad2475	bbe1aba	fd49c89	8047a365	8e37f939	515c4f6f	350f22db	1e524ce0
Round[34]:	c373d1a4	4dad2475	bbe1aba	fd49c89	31ae670e	8e37f939	515c4f6f	350f22db
Round[35]:	ef334f9e	c373d1a4	4dad2475	bbe1aba	2eac5e79	31ae670e	8e37f939	515c4f6f
Round[36]:	7ba7fe6c	ef334f9e	c373d1a4	4dad2475	ae2d01b1	2eac5e79	31ae670e	8e37f939
Round[37]:	357df7a5	7ba7fe6c	ef334f9e	c373d1a4	2583a924	ae2d01b1	2eac5e79	31ae670e
Round[38]:	2f672bdd	357df7a5	7ba7fe6c	ef334f9e	6bfabb02	2583a924	ae2d01b1	2eac5e79
Round[39]:	1662e4a6	2f672bdd	357df7a5	7ba7fe6c	3c8e6ee4	6bfabb02	2583a924	ae2d01b1
Round[40]:	7c09d4e3	1662e4a6	2f672bdd	357df7a5	950f5943	3c8e6ee4	6bfabb02	2583a924
Round[41]:	35111a2f	7c09d4e3	1662e4a6	2f672bdd	cce11467	950f5943	3c8e6ee4	6bfabb02
Round[42]:	bb67ab31	35111a2f	7c09d4e3	1662e4a6	5e77af90	cce11467	950f5943	3c8e6ee4
Round[43]:	3141a0d8	bb67ab31	35111a2f	7c09d4e3	60a9f43f	5e77af90	cce11467	950f5943
Round[44]:	55714410	3141a0d8	bb67ab31	35111a2f	9426ebbc	60a9f43f	5e77af90	cce11467
Round[45]:	29fbb314	55714410	3141a0d8	bb67ab31	3cdc7158	9426ebbc	60a9f43f	5e77af90
Round[46]:	4b20222e	29fbb314	55714410	3141a0d8	59023990	3cdc7158	9426ebbc	60a9f43f
Round[47]:	954f83bd	4b20222e	29fbb314	55714410	79ed19db	59023990	3cdc7158	9426ebbc
Round[48]:	1b14907a	954f83bd	4b20222e	29fbb314	20687488	79ed19db	59023990	3cdc7158
Round[49]:	9422e6fe	1b14907a	954f83bd	4b20222e	4bc602fe	20687488	79ed19db	59023990
Round[50]:	c470b73b	9422e6fe	1b14907a	954f83bd	61227373	4bc602fe	20687488	79ed19db
Round[51]:	7692a310	c470b73b	9422e6fe	1b14907a	ed928ef9	61227373	4bc602fe	20687488
Round[52]:	a18c3426	7692a310	c470b73b	9422e6fe	98c2b6db	ed928ef9	61227373	4bc602fe
Round[53]:	b0a07335	a18c3426	7692a310	c470b73b	26ac0713	98c2b6db	ed928ef9	61227373
Round[54]:	22b0e22d	b0a07335	a18c3426	7692a310	c2581728	26ac0713	98c2b6db	ed928ef9
Round[55]:	81a6b5db	22b0e22d	b0a07335	a18c3426	bd4b4d40	c2581728	26ac0713	98c2b6db
Round[56]:	a90a14f	81a6b5db	22b0e22d	b0a07335	371913f3	bd4b4d40	c2581728	26ac0713
Round[57]:	ed476aa6	a90a14f	81a6b5db	22b0e22d	10ddf9b0	371913f3	bd4b4d40	c2581728
Round[58]:	f5c47221	ed476aa6	a90a14f	81a6b5db	9b228758	10ddf9b0	371913f3	bd4b4d40
Round[59]:	2a6367f7	f5c47221	ed476aa6	a90a14f	c08f862f	9b228758	10ddf9b0	371913f3
Round[60]:	fa6d20c9	2a6367f7	f5c47221	ed476aa6	9ef80922	c08f862f	9b228758	10ddf9b0
Round[61]:	fb3b7abb	fa6d20c9	2a6367f7	f5c47221	21c5c5ce	9ef80922	c08f862f	9b228758
Round[62]:	6a54962	fb3b7abb	fa6d20c9	2a6367f7	3b026fef	21c5c5ce	9ef80922	c08f862f
Round[63]:	608586b8	6a54962	fb3b7abb	fa6d20c9	311e965c	3b026fef	21c5c5ce	9ef80922
Input Message :GTU								
SHA-256 HASH Result :ca8f6d1fc20cf7e737aa6e2d9fbd1603822ce8dbd607d87b41499f79fad8d63b								

Figure 4.7: All rounds and message digests for SHA-256.

	a	b	c	d	e	f	g	h
Round[0]:	2cfd86ad 30873f88	c500d5d5 c589e0d8	429a2926 347c0d87	9159015a 30700327	a2f5e924 b46875f9	67132667 ffc00033	8d5446d7 68515131	0d0c200d 46f98f87
Round[1]:	5c1202e2 a0300f8f	2cf706ad 30973f88	c500d5d5 c589e0d8	429a2926 347c0d87	645659f0 090c0049	a2f5e924 b46875f9	67132667 ffc00033	8d5446d7 68515131
Round[2]:	5f471000 6718a999	5c202e83 a0300f8f	2cf706ad 30973f88	c500d5d5 c589e0d8	a5a9f329 f2f40d29	645659f0 090c0049	a2f5e924 b46875f9	67132667 ffc00033
Round[3]:	c031a0d0 8401c1d3	5f471000 6718a999	5c202e83 a0300f8f	2cf706ad 30973f88	506f31c0 46771d25	a5a9f329 f2f40d29	645659f0 090c0049	a2f5e924 b46875f9
Round[4]:	15a0e09e 52c4f188	c031a0d0 8401c1d3	5f471000 6718a999	5c202e83 a0300f8f	03a1a039 230003a5	506f31c0 46771d25	a5a9f329 f2f40d29	645659f0 090c0049
Round[5]:	a02f08c9 0882c7f0	15a0e09e 52c4f188	c031a0d0 8401c1d3	5f471000 6718a999	620a4705 a0629027	03a1a039 230003a5	506f31c0 46771d25	a5a9f329 f2f40d29
Round[6]:	a0f3cf0a 85c5a03a	a02f08c9 0882c7f0	15a0e09e 52c4f188	c031a0d0 8401c1d3	c076a40e 07040323	620a4705 a0629027	03a1a039 230003a5	506f31c0 46771d25
Round[7]:	94c4a548 67340c33	a0f3cf0a 85c5a03a	a02f08c9 0882c7f0	15a0e09e 52c4f188	027a9232 09a970a5	c076a40e 07040323	620a4705 a0629027	03a1a039 230003a5
Round[8]:	a610c055 18240ee	94c4a548 67340c33	a0f3cf0a 85c5a03a	a02f08c9 0882c7f0	a0055a05 211517e0	a2792322 09a970a5	c076a40e 07040323	620a4705 a0629027
Round[9]:	f0309c39 c600959	a610c055 18240ee	94c4a548 67340c33	a0f3cf0a 85c5a03a	4f3c5f3b aec4a00e	a0055a05 211517e0	a2792322 09a970a5	c076a40e 07040323
Round[10]:	2a22f3c9 3feaf0a5	f0309c39 c600959	a610c055 18240ee	94c4a548 67340c33	09c0a00a a4a0053a	4f3c5f3b aec4a00e	a0055a05 211517e0	a2792322 09a970a5
Round[11]:	7c000770 2050522c	2a22f3c9 3feaf0a5	f0309c39 c600959	a610c055 18240ee	99700e5b 05d47913	09c0a00a a4a0053a	4f3c5f3b aec4a00e	a0055a05 211517e0
Round[12]:	709c4f0d ead9a031	7c000770 2050522c	2a22f3c9 3feaf0a5	f0309c39 c600959	00044012 3726c2a0	99700e5b 05d47913	09c0a00a a4a0053a	4f3c5f3b aec4a00e
Round[13]:	3c0b1340 47c4c7a6	709c4f0d ead9a031	7c000770 2050522c	2a22f3c9 3feaf0a5	57c0b09f 13005044	00044012 3726c2a0	99700e5b 05d47913	09c0a00a a4a0053a
Round[14]:	f90ec0d3 9047f7d3	3c0b1340 47c4c7a6	709c4f0d ead9a031	7c000770 2050522c	70c4f031 70100874	57c0b09f 13005044	00044012 3726c2a0	99700e5b 05d47913
Round[15]:	b43295a2 b0000c3c	f90ec0d3 9047f7d3	3c0b1340 47c4c7a6	709c4f0d ead9a031	f3879742 49a530c5	70c4f031 70100874	57c0b09f 13005044	00044012 3726c2a0
Round[16]:	f42959fd faa0d28	b43295a2 b0000c3c	f90ec0d3 9047f7d3	3c0b1340 47c4c7a6	c5c0a031 6c97700d	f3879742 49a530c5	70c4f031 70100874	57c0b09f 13005044
Round[17]:	b312a02d c20c19b0	f42959fd faa0d28	b43295a2 b0000c3c	f90ec0d3 9047f7d3	a2964013 641f90cf	c5c0a031 6c97700d	f3879742 49a530c5	70c4f031 70100874
Round[18]:	f1a210f1 90f83184	b312a02d c20c19b0	f42959fd faa0d28	b43295a2 b0000c3c	36a0a00e 03f91d62	a2964013 641f90cf	c5c0a031 6c97700d	f3879742 49a530c5
Round[19]:	d000b0a2 d072f106	f1a210f1 90f83184	b312a02d c20c19b0	f42959fd faa0d28	105dc0c2 000c4a0e	36a0a00e 03f91d62	a2964013 641f90cf	c5c0a031 6c97700d
Round[20]:	2a6f416c 16a0030c	d000b0a2 d072f106	f1a210f1 90f83184	b312a02d c20c19b0	0871a009 64105f29	105dc0c2 000c4a0e	36a0a00e 03f91d62	a2964013 641f90cf
Round[21]:	3000c413 77277f46	2a6f416c 16a0030c	d000b0a2 d072f106	f1a210f1 90f83184	16331c0a c007a0b1	0871a009 64105f29	105dc0c2 000c4a0e	36a0a00e 03f91d62
Round[22]:	a042730d 4e250a0f	3000c413 77277f46	2a6f416c 16a0030c	d000b0a2 d072f106	d738531 1a44a716	16331c0a c007a0b1	0871a009 64105f29	105dc0c2 000c4a0e
Round[23]:	cc6f416c c7270a00	a042730d 4e250a0f	3000c413 77277f46	2a6f416c 16a0030c	30031e0a 00c40c02	d738531 1a44a716	16331c0a c007a0b1	0871a009 64105f29
Round[24]:	940c0372 7360a0a7	cc6f416c c7270a00	a042730d 4e250a0f	3000c413 77277f46	579a340f 6c3233a0	30031e0a 00c40c02	d738531 1a44a716	16331c0a c007a0b1
Round[25]:	50ef8446 70d40095	940c0372 7360a0a7	cc6f416c c7270a00	a042730d 4e250a0f	40514167 1591a0f7	579a340f 6c3233a0	30031e0a 00c40c02	d738531 1a44a716
Round[26]:	f0787628 00010041	50ef8446 70d40095	940c0372 7360a0a7	cc6f416c c7270a00	003a4a00 8000f044	40514167 1591a0f7	579a340f 6c3233a0	30031e0a 00c40c02
Round[27]:	5c42c7f3 9047f6a9	f0787628 00010041	50ef8446 70d40095	940c0372 7360a0a7	6021a728 4209a0a0	003a4a00 8000f044	40514167 1591a0f7	579a340f 6c3233a0
Round[28]:	340b1e00 05030104	5c42c7f3 9047f6a9	f0787628 00010041	50ef8446 70d40095	0209c1e1 412ff99c	6021a728 4209a0a0	003a4a00 8000f044	40514167 1591a0f7
Round[29]:	60261a44 810c301e	340b1e00 05030104	5c42c7f3 9047f6a9	f0787628 00010041	97570720 00030043	0209c1e1 412ff99c	6021a728 4209a0a0	003a4a00 8000f044
Round[30]:	60040406 74035214	60261a44 810c301e	340b1e00 05030104	5c42c7f3 9047f6a9	5c725970 30f0e70a	97570720 00030043	0209c1e1 412ff99c	6021a728 4209a0a0
Round[31]:	59a53f90 29300f30	60040406 74035214	60261a44 810c301e	340b1e00 05030104	02055505 5a091f96	5c725970 30f0e70a	97570720 00030043	0209c1e1 412ff99c
Round[32]:	1d103700 835099ef	59a53f90 29300f30	60040406 74035214	60261a44 810c301e	20a5505e 43f00052	02055505 5a091f96	5c725970 30f0e70a	97570720 00030043
Round[33]:	5a111a04 efa31875	1d103700 835099ef	59a53f90 29300f30	60040406 74035214	c8c0050e a0430046	20a5505e 43f00052	02055505 5a091f96	5c725970 30f0e70a
Round[34]:	c9001042 3f070f20	5a111a04 efa31875	1d103700 835099ef	59a53f90 29300f30	31620910 c07323cf	c8c0050e a0430046	20a5505e 43f00052	02055505 5a091f96
Round[35]:	7501a091 1cc0a45f	c9001042 3f070f20	5a111a04 efa31875	1d103700 835099ef	0c4cc0f6 5a0735a0	31620910 c07323cf	c8c0050e a0430046	20a5505e 43f00052
Round[36]:	40a470a3 09957702	7501a091 1cc0a45f	c9001042 3f070f20	5a111a04 efa31875	60a0c0f6 20c0f676	0c4cc0f6 5a0735a0	31620910 c07323cf	c8c0050e a0430046
Round[37]:	30fca7a0 00000000	40a470a3 09957702	7501a091 1cc0a45f	c9001042 3f070f20	37021a0f 0072c04d	60a0c0f6 20c0f676	0c4cc0f6 5a0735a0	31620910 c07323cf
Round[38]:	a01000 04020c5f	30fca7a0 00000000	40a470a3 09957702	7501a091 1cc0a45f	000c047c 4e3972a2	37021a0f 0072c04d	60a0c0f6 20c0f676	0c4cc0f6 5a0735a0
Round[39]:	200c0050 e09f2000	a01000 04020c5f	30fca7a0 00000000	40a470a3 09957702	00f160a0 c40020a8	000c047c 4e3972a2	37021a0f 0072c04d	60a0c0f6 20c0f676
Round[40]:	3c1070f7 91561a13	200c0050 e09f2000	a01000 04020c5f	30fca7a0 00000000	000c047c 4e3972a2	00f160a0 c40020a8	000c047c 4e3972a2	37021a0f 0072c04d
Round[41]:	213073a1 95631c70	3c1070f7 91561a13	200c0050 e09f2000	a01000 04020c5f	2e033001 0e31e1a0	000c047c 4e3972a2	00f160a0 c40020a8	000c047c 4e3972a2
Round[42]:	00f40555 02982007	213073a1 95631c70	3c1070f7 91561a13	200c0050 e09f2000	f1f0a200 09000000	2e033001 0e31e1a0	000c047c 4e3972a2	00f160a0 c40020a8
Round[43]:	1a030100 6470a034	00f40555 02982007	213073a1 95631c70	3c1070f7 91561a13	3c1070f7 91561a13	f1f0a200 09000000	2e033001 0e31e1a0	000c047c 4e3972a2
Round[44]:	35c121e7 81740a00	1a030100 6470a034	00f40555 02982007	213073a1 95631c70	04a4a04c 6a100150	3c1070f7 91561a13	f1f0a200 09000000	2e033001 0e31e1a0
Round[45]:	e000c0c4 07709a00	35c121e7 81740a00	00f40555 02982007	213073a1 95631c70	c2647415 a00030a0	04a4a04c 6a100150	3c1070f7 91561a13	f1f0a200 09000000
Round[46]:	c220c100 30091579	e000c0c4 07709a00	35c121e7 81740a00	00f40555 02982007	f155f101 0001c816	c2647415 a00030a0	04a4a04c 6a100150	3c1070f7 91561a13
Round[47]:	9c30a200 09001e32	c220c100 30091579	e000c0c4 07709a00	35c121e7 81740a00	f0470081 55c00a0e	f155f101 0001c816	c2647415 a00030a0	04a4a04c 6a100150
Round[48]:	c0a25025 0f000c12	9c30a200 09001e32	c220c100 30091579	e000c0c4 07709a00	00001003 02009f16	f0470081 55c00a0e	f155f101 0001c816	c2647415 a00030a0
Round[49]:	a72e9300 c0c0975e	c0a25025 0f000c12	9c30a200 09001e32	c220c100 30091579	63ef32c0 c1509415	00001003 02009f16	f0470081 55c00a0e	f155f101 0001c816
Round[50]:	b031081c 43c7051d	a72e9300 c0c0975e	c0a25025 0f000c12	9c30a200 09001e32	c03a07f7 00002f0d	63ef32c0 c1509415	00001003 02009f16	f0470081 55c00a0e
Round[51]:	77a17c01 403c1f0d	b031081c 43c7051d	a72e9300 c0c0975e	c0a25025 0f000c12	8150a009 1172030d	c03a07f7 00002f0d	63ef32c0 c1509415	00001003 02009f16
Round[52]:	66c0737e 4e23f17a	77a17c01 403c1f0d	b031081c 43c7051d	a72e9300 c0c0975e	58a01503 c00f0d79	8150a009 1172030d	c03a07f7 00002f0d	63ef32c0 c1509415
Round[53]:	f0c1a0c0 30d0a0a7	66c0737e 4e23f17a	77a17c01 403c1f0d	b031081c 43c7051d	53941f0c 03f0f0d0	58a01503 c00f0d79	8150a009 1172030d	c03a07f7 00002f0d
Round[54]:	f9099f0a 902000a0	f0c1a0c0 30d0a0a7	66c0737e 4e23f17a	77a17c01 403c1f0d	c50a0000 070401c4	53941f0c 03f0f0d0	58a01503 c00f0d79	8150a009 1172030d
Round[55]:	0000b0d0 16023a00	f9099f0a 902000a0	f0c1a0c0 30d0a0a7	66c0737e 4e23f17a	200c0000 000a0f00	c50a0000 070401c4	53941f0c 03f0f0d0	58a01503 c00f0d79
Round[56]:	64a0c0a2 2776370a	0000b0d0 16023a00	f9099f0a 902000a0	f0c1a0c0 30d0a0a7	1000f000 000a0f00	200c0000 000a0f00	c50a0000 070401c4	53941f0c 03f0f0d0
Round[57]:	1c000700 09055a0d	64a0c0a2 2776370a	0000b0d0 16023a00	f9099f0a 902000a0	c2f0a002 9310a330	1000f000 000a0f00	200c0000 000a0f00	c50a0000 070401c4
Round[58]:	007c053c 3096f316	1c000700 09055a0d	64a0c0a2 2776370a	0000b0d0 16023a00	05a01a15 03a0f27c	c2f0a002 9310a330	1000f000 000a0f00	200c0000 000a0f00
Round[59]:	c03a0009 e030a000	007c053c 3096f316	1c000700 09055a0d	64a0c0a2 2776370a	0a02f930 5000a052	05a01a15 03a0f27c	c2f0a002 9310a330	1000f000 000a0f00
Round[60]:	46f91000 c29a00f0	c03a0009 e030a000	007c053c 3096f316	1c000700 09055a0d	123370a2 7302a02f	0a02f930 5000a052	05a01a15 03a0f27c	c2f0a002 9310a330
Round[61]:	7a3a732e 39032770	46f91000 c29a00f0	c03a0009 e030a000	007c053c 3096f316	f9c0a072 0a000a01	123370a2 7302a02f	0a02f930 5000a052	05a01a15 03a0f27c
Round[62]:	0350e000 07c0000f	7a3a732e 39032770	46f91000 c29a00f0	c03a0009 e030a000	a471a20a 5000c413	f9c0a072 0a000a01	123370a2 7302a02f	0a02f930 5000a052
Round[63]:	2f053ee 3094a013	0350e000 07c0000f	7a3a732e 39032770	46f91000 c29a00f0	300007f9 c000e000	a471a20a 5000c413	f9c0a072 0a000a01	123370a2 7302a02f
Round[64]:	501f7009 000c0000	2f053ee 3094a013	0350e000 07c0000f	7a3a732e 39032770	01070f0c 50000000	300007f9 c000e000	a471a20a 5000c413	f9c0a072 0a000a01
Round[65]:	0022c0c0 a0a0000f	501f7009 000c0000	2f053ee 3094a013	0350e000 07c0000f	501f7009 000c0000	01070f0c 50000000	300007f9 c000e000	a471a20a 5000c413
Round[66]:	501f7009 000c0000	0022c0c0 a						

	A	B	C	D	E	F	G	H
Round[0]:	dca1c08d bcfcd0f5	6a99e667 f30cc908	b067ae05 84caa73b	3c6ef372 fe94f82b	3ebcf434 7a051f91	518e527f a0e682d1	90b6e80c 253e6c1f	1f83d9db f043d0db
Round[1]:	d135f8d0 79ad3537	dca1c08d bcfcd0f5	6a99e667 f30cc908	b067ae05 84caa73b	81d55a3f 6c8b2805	3ebcf434 7a051f91	518e527f a0e682d1	90b6e80c 253e6c1f
Round[2]:	d52fabd0 c2c94611	d135f8d0 79ad3537	dca1c08d bcfcd0f5	6a99e667 f30cc908	126d9e8a c2ff5cbd	81d55a3f 6c8b2805	3ebcf434 7a051f91	518e527f a0e682d1
Round[3]:	d52fabd0 c2c94611	d135f8d0 79ad3537	dca1c08d bcfcd0f5	6a99e667 f30cc908	d25e014e de05c027	126d9e8a c2ff5cbd	81d55a3f 6c8b2805	3ebcf434 7a051f91
Round[4]:	944686ca caaf795c	d52fabd0 c2c94611	d135f8d0 79ad3537	4075289a a4a9fe11	d25e014e de05c027	126d9e8a c2ff5cbd	81d55a3f 6c8b2805	3ebcf434 7a051f91
Round[5]:	40ef0b47 e025cf53	944686ca caaf795c	6c65df95 3877401e	d52fabd0 c2c94611	8cc6655a c43a7e1d	4075289a a4a9fe11	d25e014e de05c027	126d9e8a c2ff5cbd
Round[6]:	28c67c00 11413418	40ef0b47 e025cf53	944686ca caaf795c	6c65df95 3877401e	44a0a9f7 2106f942	8cc6655a c43a7e1d	4075289a a4a9fe11	d25e014e de05c027
Round[7]:	70c0e1f4 a5c976d0	28c67c00 11413418	40ef0b47 e025cf53	944686ca caaf795c	6f8463d8 7a9f70ce	44a0a9f7 2106f942	8cc6655a c43a7e1d	4075289a a4a9fe11
Round[8]:	987ac0da f8fed0fc	70c0e1f4 a5c976d0	28c67c00 11413418	40ef0b47 e025cf53	e3400913 a6a7856c	6f8463d8 7a9f70ce	44a0a9f7 2106f942	8cc6655a c43a7e1d
Round[9]:	b090a408 f4145372	987ac0da f8fed0fc	70c0e1f4 a5c976d0	28c67c00 11413418	4798a4c8 d6ee0265	e3400913 a6a7856c	6f8463d8 7a9f70ce	44a0a9f7 2106f942
Round[10]:	879881a0 52b20e0d	b090a408 f4145372	987ac0da f8fed0fc	70c0e1f4 a5c976d0	ebe02805 b096f415	4798a4c8 d6ee0265	e3400913 a6a7856c	6f8463d8 7a9f70ce
Round[11]:	879881a0 52b20e0d	b090a408 f4145372	987ac0da f8fed0fc	70c0e1f4 a5c976d0	e375e2a8 2a40fc63	ebe02805 b096f415	4798a4c8 d6ee0265	e3400913 a6a7856c
Round[12]:	b506021a 3cd2e4ae	879881a0 52b20e0d	b090a408 f4145372	2cf11719 4a091728	f575e2a8 2a40fc63	ebe02805 b096f415	4798a4c8 d6ee0265	e3400913 a6a7856c
Round[13]:	d345905b 36fb6cda	b506021a 3cd2e4ae	879881a0 52b20e0d	fe57b000 40d0e17e	2cf11719 4a091728	f575e2a8 2a40fc63	ebe02805 b096f415	4798a4c8 d6ee0265
Round[14]:	4e3b2437 27683898	d345905b 36fb6cda	b506021a 3cd2e4ae	3a4a0b31 272a1238	fe57b000 40d0e17e	2cf11719 4a091728	f575e2a8 2a40fc63	ebe02805 b096f415
Round[15]:	634374cd d8ff824f	4e3b2437 27683898	d345905b 36fb6cda	b506021a 3cd2e4ae	d6e5132a 10fe056f	3a4a0b31 272a1238	fe57b000 40d0e17e	2cf11719 4a091728
Round[16]:	56b7498b 9781c41f	634374cd d8ff824f	4e3b2437 27683898	d345905b 36fb6cda	d6e5132a 10fe056f	3a4a0b31 272a1238	fe57b000 40d0e17e	2cf11719 4a091728
Round[17]:	56b7498b 9781c41f	634374cd d8ff824f	4e3b2437 27683898	d345905b 36fb6cda	d6e5132a 10fe056f	3a4a0b31 272a1238	fe57b000 40d0e17e	2cf11719 4a091728
Round[18]:	8d07a0fd a77b08a9	56b7498b 9781c41f	634374cd d8ff824f	42e0c800 f05cfff1d	d0f7652c a3b7a29	d6e5132a 10fe056f	3a4a0b31 272a1238	fe57b000 40d0e17e
Round[19]:	64b0f93a 2c43065d	8d07a0fd a77b08a9	56b7498b 9781c41f	94325f20 fc09caef	42e0c800 f05cfff1d	d0f7652c a3b7a29	d6e5132a 10fe056f	3a4a0b31 272a1238
Round[20]:	64b0f93a 2c43065d	8d07a0fd a77b08a9	56b7498b 9781c41f	94325f20 fc09caef	42e0c800 f05cfff1d	d0f7652c a3b7a29	d6e5132a 10fe056f	3a4a0b31 272a1238
Round[21]:	3283a016 d6a93a4a	64b0f93a 2c43065d	8d07a0fd a77b08a9	5743d00e 153d3096	e4036953 ac1a4fe	94325f20 fc09caef	42e0c800 f05cfff1d	d0f7652c a3b7a29
Round[22]:	00020913 a08c2508	3283a016 d6a93a4a	64b0f93a 2c43065d	7a230e24 1233165e	5743d00e 153d3096	e4036953 ac1a4fe	94325f20 fc09caef	42e0c800 f05cfff1d
Round[23]:	00020913 a08c2508	3283a016 d6a93a4a	64b0f93a 2c43065d	7a230e24 1233165e	5743d00e 153d3096	e4036953 ac1a4fe	94325f20 fc09caef	42e0c800 f05cfff1d
Round[24]:	230c0e7e f70f757e	c6230e27 d812a045	00020913 a08c2508	3283a016 d6a93a4a	124cf0f39 06c51b06	10a09e77 64f05c4e	7a230e24 1233165e	5743d00e 153d3096
Round[25]:	14f54f0e d603f81	230c0e7e f70f757e	c6230e27 d812a045	00020913 a08c2508	520c2064 d088570b	124cf0f39 06c51b06	10a09e77 64f05c4e	7a230e24 1233165e
Round[26]:	14f54f0e d603f81	230c0e7e f70f757e	c6230e27 d812a045	00020913 a08c2508	520c2064 d088570b	124cf0f39 06c51b06	10a09e77 64f05c4e	7a230e24 1233165e
Round[27]:	14f54f0e d603f81	230c0e7e f70f757e	c6230e27 d812a045	00020913 a08c2508	520c2064 d088570b	124cf0f39 06c51b06	10a09e77 64f05c4e	7a230e24 1233165e
Round[28]:	49a916d3 80e0156b	14f54f0e d603f81	230c0e7e f70f757e	c6230e27 d812a045	00020913 a08c2508	520c2064 d088570b	124cf0f39 06c51b06	10a09e77 64f05c4e
Round[29]:	b090a408 f4145372	49a916d3 80e0156b	14f54f0e d603f81	230c0e7e f70f757e	c6230e27 d812a045	00020913 a08c2508	520c2064 d088570b	124cf0f39 06c51b06
Round[30]:	eef7257f e9945a1e	b090a408 f4145372	49a916d3 80e0156b	14f54f0e d603f81	230c0e7e f70f757e	c6230e27 d812a045	00020913 a08c2508	520c2064 d088570b
Round[31]:	9c14ed13 301d0d59	eef7257f e9945a1e	b090a408 f4145372	49a916d3 80e0156b	14f54f0e d603f81	230c0e7e f70f757e	c6230e27 d812a045	00020913 a08c2508
Round[32]:	83000591 c6a280cc	9c14ed13 301d0d59	eef7257f e9945a1e	b090a408 f4145372	49a916d3 80e0156b	14f54f0e d603f81	230c0e7e f70f757e	c6230e27 d812a045
Round[33]:	6028ef04 1050e043	83000591 c6a280cc	9c14ed13 301d0d59	eef7257f e9945a1e	b090a408 f4145372	49a916d3 80e0156b	14f54f0e d603f81	230c0e7e f70f757e
Round[34]:	a095028a 3fb63aad	6028ef04 1050e043	83000591 c6a280cc	9c14ed13 301d0d59	eef7257f e9945a1e	b090a408 f4145372	49a916d3 80e0156b	14f54f0e d603f81
Round[35]:	6018014 50ec786c	a095028a 3fb63aad	6028ef04 1050e043	83000591 c6a280cc	9c14ed13 301d0d59	eef7257f e9945a1e	b090a408 f4145372	49a916d3 80e0156b
Round[36]:	10421488 487a38b9	6018014 50ec786c	a095028a 3fb63aad	6028ef04 1050e043	83000591 c6a280cc	9c14ed13 301d0d59	eef7257f e9945a1e	b090a408 f4145372
Round[37]:	45000f72 97a678ff	10421488 487a38b9	6018014 50ec786c	a095028a 3fb63aad	6028ef04 1050e043	83000591 c6a280cc	9c14ed13 301d0d59	eef7257f e9945a1e
Round[38]:	13ee9f58 30e311a6	45000f72 97a678ff	10421488 487a38b9	6018014 50ec786c	a095028a 3fb63aad	6028ef04 1050e043	83000591 c6a280cc	9c14ed13 301d0d59
Round[39]:	1070006f 31534008	13ee9f58 30e311a6	45000f72 97a678ff	10421488 487a38b9	6018014 50ec786c	a095028a 3fb63aad	6028ef04 1050e043	83000591 c6a280cc
Round[40]:	1e0d00f2 59061f8b	1070006f 31534008	13ee9f58 30e311a6	45000f72 97a678ff	10421488 487a38b9	6018014 50ec786c	a095028a 3fb63aad	6028ef04 1050e043
Round[41]:	40ef0b47 e025cf53	1e0d00f2 59061f8b	1070006f 31534008	13ee9f58 30e311a6	45000f72 97a678ff	10421488 487a38b9	6018014 50ec786c	a095028a 3fb63aad
Round[42]:	b28a4100 e0f0357a	40ef0b47 e025cf53	1e0d00f2 59061f8b	1070006f 31534008	13ee9f58 30e311a6	45000f72 97a678ff	10421488 487a38b9	6018014 50ec786c
Round[43]:	7669350e aa37458a	b28a4100 e0f0357a	40ef0b47 e025cf53	1e0d00f2 59061f8b	1070006f 31534008	13ee9f58 30e311a6	45000f72 97a678ff	10421488 487a38b9
Round[44]:	40ca285e 30554a04	7669350e aa37458a	b28a4100 e0f0357a	40ef0b47 e025cf53	1e0d00f2 59061f8b	1070006f 31534008	13ee9f58 30e311a6	45000f72 97a678ff
Round[45]:	70e0ac95 9e3a42fc	40ca285e 30554a04	7669350e aa37458a	b28a4100 e0f0357a	40ef0b47 e025cf53	1e0d00f2 59061f8b	1070006f 31534008	13ee9f58 30e311a6
Round[46]:	8f62c7c7 163441ff	70e0ac95 9e3a42fc	40ca285e 30554a04	7669350e aa37458a	b28a4100 e0f0357a	40ef0b47 e025cf53	1e0d00f2 59061f8b	1070006f 31534008
Round[47]:	8f62c7c7 163441ff	70e0ac95 9e3a42fc	40ca285e 30554a04	7669350e aa37458a	b28a4100 e0f0357a	40ef0b47 e025cf53	1e0d00f2 59061f8b	1070006f 31534008
Round[48]:	48165fff 93c0d452	8f62c7c7 163441ff	70e0ac95 9e3a42fc	40ca285e 30554a04	7669350e aa37458a	b28a4100 e0f0357a	40ef0b47 e025cf53	1e0d00f2 59061f8b
Round[49]:	7801f49 1700a408	48165fff 93c0d452	8f62c7c7 163441ff	70e0ac95 9e3a42fc	40ca285e 30554a04	7669350e aa37458a	b28a4100 e0f0357a	40ef0b47 e025cf53
Round[50]:	f8b01f49 1700a408	7801f49 1700a408	48165fff 93c0d452	8f62c7c7 163441ff	70e0ac95 9e3a42fc	40ca285e 30554a04	7669350e aa37458a	b28a4100 e0f0357a
Round[51]:	a438a08d a39ffef3	f8b01f49 1700a408	7801f49 1700a408	48165fff 93c0d452	8f62c7c7 163441ff	70e0ac95 9e3a42fc	40ca285e 30554a04	7669350e aa37458a
Round[52]:	c641851c c9037a0e	a438a08d a39ffef3	f8b01f49 1700a408	7801f49 1700a408	48165fff 93c0d452	8f62c7c7 163441ff	70e0ac95 9e3a42fc	40ca285e 30554a04
Round[53]:	d5960d9f 30890364	c641851c c9037a0e	a438a08d a39ffef3	f8b01f49 1700a408	7801f49 1700a408	48165fff 93c0d452	8f62c7c7 163441ff	70e0ac95 9e3a42fc
Round[54]:	7218c5d3 40a9e727	d5960d9f 30890364	c641851c c9037a0e	a438a08d a39ffef3	f8b01f49 1700a408	7801f49 1700a408	48165fff 93c0d452	8f62c7c7 163441ff
Round[55]:	59e90d9f 30890364	7218c5d3 40a9e727	d5960d9f 30890364	c641851c c9037a0e	a438a08d a39ffef3	f8b01f49 1700a408	7801f49 1700a408	48165fff 93c0d452
Round[56]:	827479a7 1ff2c872	59e90d9f 30890364	7218c5d3 40a9e727	d5960d9f 30890364	c641851c c9037a0e	a438a08d a39ffef3	f8b01f49 1700a408	7801f49 1700a408
Round[57]:	827479a7 1ff2c872	59e90d9f 30890364	7218c5d3 40a9e727	d5960d9f 30890364	c641851c c9037a0e	a438a08d a39ffef3	f8b01f49 1700a408	7801f49 1700a408
Round[58]:	59e90d9f 30890364	827479a7 1ff2c872	d5960d9f 30890364	c641851c c9037a0e	a438a08d a39ffef3	f8b01f49 1700a408	7801f49 1700a408	48165fff 93c0d452
Round[59]:	10950f98 5c7e4e01	59e90d9f 30890364	827479a7 1ff2c872	d5960d9f 30890364	c641851c c9037a0e	a438a08d a39ffef3	f8b01f49 1700a408	7801f49 1700a408
Round[60]:	4e3279a4 74a89209	10950f98 5c7e4e01	59e90d9f 30890364	827479a7 1ff2c872	d5960d9f 30890364	c641851c c9037a0e	a438a08d a39ffef3	f8b01f49 1700a408
Round[61]:	84ae5028 cf302946	4e3279a4 74a89209	10950f98 5c7e4e01	59e90d9f 30890364	827479a7 1ff2c872	d5960d9f 30890364	c641851c c9037a0e	a438a08d a39ffef3
Round[62]:	c76f2185 3054330e	84ae5028 cf302946	e43279a4 74a89209	10950f98 5c7e4e01	59e90d9f 30890364	827479a7 1ff2c872	d5960d9f 30890364	c641851c c9037a0e
Round[63]:	40f979ac 152976fc	c76f2185 3054330e	84ae5028 cf302946	e43279a4 74a89209	10950f98 5c7e4e01	59e90d9f 30890364	827479a7 1ff2c872	d5960d9f 30890364
Round[64]:	c80e0904 3265e111	40f979ac 152976fc	c76f2185 3054330e	84ae5028 cf302946	e43279a4 74a89209	10950f98 5c7e4e01	59e90d9f 30890364	827479a7 1ff2c872
Round[65]:	3300c43c 41399c0c	c80e0904 3265e111	40f979ac 152976fc	c76f2185 3054330e	84ae5028 cf302946	e43279a4 74a89209	10950f98 5c7e4e01	59e90d9f 30890364
Round[66]:	b7ecf304 ac47e126	3300c43c 41399c0c	c80e0904 3265e111	40f979ac 152976fc	c76f2185 3054330e	84ae5028 cf302946	e43279a4 74a89209	10950f98 5c7e4e01
Round[67]:	9180ed91 fee083	b7ecf304 ac47e126	3300c43c 41399c0c	c80e0904 3265e111	40f979ac 152976fc	c76f2185 3054330e	84ae5028 cf302946	e43279a4 74a89209
Round[68]:	231c8163 82028506	9180ed91 fee083	b7ecf304 ac47e126	3300c43c 41399c0c	c80e0904 3265e111	40		

	a	b	c	d	e	f	g	h
Round[0]:	85789909 c85a8a02	8c3d37c8 19544a82	73e19966 890cda06	1d0fa79e 321ff9c82	c0f8c504 f5742383	f6d2d69 70d44d08	77c3b473 4c48942	3f9d8508 8a22d3c8
Round[1]:	47488360 92471716	85789909 c85a8a02	8c3d37c8 19544a82	73e19966 890cda06	34f84e17 d797d746	c0f8c504 f5742383	f6d2d69 70d44d08	77c3b473 4c48942
Round[2]:	71d0f952c 880c9522	47488360 92471716	85789909 c85a8a02	8c3d37c8 19544a82	48a5580c 6fc7f8a1	24f914e7 d797d746	34f84e17 d797d746	c0f8c504 f5742383
Round[3]:	19w9221c 7a750dc7	71d0f952c 880c9522	47488360 92471716	85789909 c85a8a02	6af3d03e 3e1d400a	48a5580c 6fc7f8a1	24f914e7 d797d746	c0f8c504 f5742383
Round[4]:	08a8f131 82f64306	19w9221c 7a750dc7	71d0f952c 880c9522	47488360 92471716	7a21ac30 6a0f8a7f	6af3d03e 3e1d400a	48a5580c 6fc7f8a1	24f914e7 d797d746
Round[5]:	018c6d0 8a752858	08a8f131 82f64306	19w9221c 7a750dc7	71d0f952c 880c9522	4c6a8c30 3c0fa36a	7a21ac30 6a0f8a7f	6af3d03e 3e1d400a	48a5580c 6fc7f8a1
Round[6]:	9a095231 9238a8a6	018c6d0 8a752858	08a8f131 82f64306	19w9221c 7a750dc7	22e70f30 716da4e2	4c6a8c30 3c0fa36a	7a21ac30 6a0f8a7f	6af3d03e 3e1d400a
Round[7]:	15c13089a 7f471056	9a095231 9238a8a6	018c6d0 8a752858	08a8f131 82f64306	70083919 61090236	22e70f30 716da4e2	4c6a8c30 3c0fa36a	7a21ac30 6a0f8a7f
Round[8]:	28512446 49372931	15c13089a 7f471056	9a095231 9238a8a6	018c6d0 8a752858	6396a73a 68745085	70083919 61090236	22e70f30 716da4e2	4c6a8c30 3c0fa36a
Round[9]:	1975f4fb fccca771	28512446 49372931	15c13089a 7f471056	9a095231 9238a8a6	678952c7 e2bc1e09	6396a73a 68745085	70083919 61090236	22e70f30 716da4e2
Round[10]:	d73a306d 903d04f5	1975f4fb fccca771	28512446 49372931	15c13089a 7f471056	77c79330 fcd78c21	678952c7 e2bc1e09	6396a73a 68745085	70083919 61090236
Round[11]:	08a8f131 82f64306	d73a306d 903d04f5	1975f4fb fccca771	28512446 49372931	8d3d635a 60f8cfee	77c79330 fcd78c21	678952c7 e2bc1e09	6396a73a 68745085
Round[12]:	4a28380c 392a4962	08a8f131 82f64306	d73a306d 903d04f5	1975f4fb fccca771	7a051e17 476dcf3	8d3d635a 60f8cfee	77c79330 fcd78c21	678952c7 e2bc1e09
Round[13]:	d786c126 002281c3	4a28380c 392a4962	08a8f131 82f64306	d73a306d 903d04f5	95902544 3a077e26	7a051e17 476dcf3	8d3d635a 60f8cfee	77c79330 fcd78c21
Round[14]:	0786c126 002281c3	d786c126 002281c3	4a28380c 392a4962	08a8f131 82f64306	7d742f3f 9a312a08	95902544 3a077e26	7a051e17 476dcf3	8d3d635a 60f8cfee
Round[15]:	4345789c 07a00d4e	0786c126 002281c3	d786c126 002281c3	4a28380c 392a4962	d7892f2c 4f5a2019	7d742f3f 9a312a08	95902544 3a077e26	7a051e17 476dcf3
Round[16]:	f173308a 8326531e	4345789c 07a00d4e	0786c126 002281c3	d786c126 002281c3	8374ff3a cfb59986	d7892f2c 4f5a2019	7d742f3f 9a312a08	95902544 3a077e26
Round[17]:	d73a308a 8326531e	f173308a 8326531e	4345789c 07a00d4e	0786c126 002281c3	60a0080c 4a0f8928	8374ff3a cfb59986	d7892f2c 4f5a2019	7d742f3f 9a312a08
Round[18]:	5fc2771c 9a0f28f9	d73a308a 8326531e	f173308a 8326531e	4345789c 07a00d4e	6207542b 9a4f97a1	60a0080c 4a0f8928	8374ff3a cfb59986	d7892f2c 4f5a2019
Round[19]:	9e4e78a5 e77f9599	5fc2771c 9a0f28f9	d73a308a 8326531e	f173308a 8326531e	af5a6d8a 2a5ef3a	6207542b 9a4f97a1	60a0080c 4a0f8928	8374ff3a cfb59986
Round[20]:	bcf34972 3f29c018	9e4e78a5 e77f9599	5fc2771c 9a0f28f9	d73a308a 8326531e	82c0a596 9a0f7e08	af5a6d8a 2a5ef3a	6207542b 9a4f97a1	60a0080c 4a0f8928
Round[21]:	8a0f8793 7550a418	bcf34972 3f29c018	9e4e78a5 e77f9599	5fc2771c 9a0f28f9	325a844c 3e72e057	82c0a596 9a0f7e08	af5a6d8a 2a5ef3a	6207542b 9a4f97a1
Round[22]:	e0e40741 5020a45f	8a0f8793 7550a418	bcf34972 3f29c018	9e4e78a5 e77f9599	f539c315 8a70a05a	325a844c 3e72e057	82c0a596 9a0f7e08	af5a6d8a 2a5ef3a
Round[23]:	bcf34972 3f29c018	e0e40741 5020a45f	8a0f8793 7550a418	bcf34972 3f29c018	293c4d79 1c1f0e02	f539c315 8a70a05a	325a844c 3e72e057	82c0a596 9a0f7e08
Round[24]:	25120184 7e95780	bcf34972 3f29c018	e0e40741 5020a45f	8a0f8793 7550a418	09001c3a f9358f48	293c4d79 1c1f0e02	f539c315 8a70a05a	325a844c 3e72e057
Round[25]:	3a02a25c 11283029	25120184 7e95780	bcf34972 3f29c018	e0e40741 5020a45f	cc7e53d3 a00c73a1	09001c3a f9358f48	293c4d79 1c1f0e02	f539c315 8a70a05a
Round[26]:	d7c0b50c 570059f9	3a02a25c 11283029	25120184 7e95780	bcf34972 3f29c018	553b1369 a03a0507	cc7e53d3 a00c73a1	09001c3a f9358f48	293c4d79 1c1f0e02
Round[27]:	0d00a0ff 531c4ff3	d7c0b50c 570059f9	3a02a25c 11283029	25120184 7e95780	4a000308 09705d09	553b1369 a03a0507	cc7e53d3 a00c73a1	09001c3a f9358f48
Round[28]:	5a5f9f19 3a0a0f3c	0d00a0ff 531c4ff3	d7c0b50c 570059f9	3a02a25c 11283029	7fc06001 e9f02319	4a000308 09705d09	553b1369 a03a0507	cc7e53d3 a00c73a1
Round[29]:	d31cc8ff 4093a2a8	5a5f9f19 3a0a0f3c	0d00a0ff 531c4ff3	d7c0b50c 570059f9	19f02057 f050a0c5	7fc06001 e9f02319	4a000308 09705d09	553b1369 a03a0507
Round[30]:	23077763 79071c08	d31cc8ff 4093a2a8	5a5f9f19 3a0a0f3c	0d00a0ff 531c4ff3	7319231a f81182f9	19f02057 f050a0c5	7fc06001 e9f02319	4a000308 09705d09
Round[31]:	c5a34001 93f2a240	23077763 79071c08	d31cc8ff 4093a2a8	5a5f9f19 3a0a0f3c	816f14fe c130a511	7319231a f81182f9	19f02057 f050a0c5	7fc06001 e9f02319
Round[32]:	5073390d 8a281c0a	c5a34001 93f2a240	23077763 79071c08	d31cc8ff 4093a2a8	33a0edff 6912a319	816f14fe c130a511	7319231a f81182f9	19f02057 f050a0c5
Round[33]:	8a03c131 9a43a018	5073390d 8a281c0a	c5a34001 93f2a240	23077763 79071c08	0e4c1390 2d33a0f9	33a0edff 6912a319	816f14fe c130a511	7319231a f81182f9
Round[34]:	60e0b30c 39a0a406	8a03c131 9a43a018	5073390d 8a281c0a	c5a34001 93f2a240	c57805e7 3e1a784d	0e4c1390 2d33a0f9	33a0edff 6912a319	816f14fe c130a511
Round[35]:	08b3a02c 39a0a406	60e0b30c 39a0a406	8a03c131 9a43a018	5073390d 8a281c0a	053287a0 03f4c39d	c57805e7 3e1a784d	0e4c1390 2d33a0f9	33a0edff 6912a319
Round[36]:	f95e3e0c 4a40d0d9	08b3a02c 39a0a406	60e0b30c 39a0a406	8a03c131 9a43a018	6903387a 833954f6	053287a0 03f4c39d	c57805e7 3e1a784d	0e4c1390 2d33a0f9
Round[37]:	60e0b30c 39a0a406	f95e3e0c 4a40d0d9	08b3a02c 39a0a406	8a03c131 9a43a018	8308704d eef7e08a	6903387a 833954f6	053287a0 03f4c39d	c57805e7 3e1a784d
Round[38]:	a09404f3 e022a0f4	60e0b30c 39a0a406	f95e3e0c 4a40d0d9	08b3a02c 39a0a406	ef8c7c0a 0a0e582a	8308704d eef7e08a	6903387a 833954f6	053287a0 03f4c39d
Round[39]:	402f0c73 2a330a03	a09404f3 e022a0f4	60e0b30c 39a0a406	f95e3e0c 4a40d0d9	af8f1a07 d09a0082	ef8c7c0a 0a0e582a	8308704d eef7e08a	6903387a 833954f6
Round[40]:	402f0c73 2a330a03	402f0c73 2a330a03	a09404f3 e022a0f4	60e0b30c 39a0a406	0a03a000 9c49a00e	af8f1a07 d09a0082	ef8c7c0a 0a0e582a	8308704d eef7e08a
Round[41]:	8a01c130 40f05ccf	402f0c73 2a330a03	402f0c73 2a330a03	a09404f3 e022a0f4	775044cf e3711a00	0a03a000 9c49a00e	af8f1a07 d09a0082	ef8c7c0a 0a0e582a
Round[42]:	82099530 041255a1	8a01c130 40f05ccf	402f0c73 2a330a03	a09404f3 e022a0f4	0a03a000 9c49a00e	775044cf e3711a00	af8f1a07 d09a0082	ef8c7c0a 0a0e582a
Round[43]:	2a330a03 041255a1	82099530 041255a1	8a01c130 40f05ccf	402f0c73 2a330a03	21b0ca93 d042a79	82099530 041255a1	8a01c130 40f05ccf	402f0c73 2a330a03
Round[44]:	4f244517 1a4c6e70	2a330a03 041255a1	82099530 041255a1	8a01c130 40f05ccf	3e93d00d 09e0c054	4f244517 1a4c6e70	2a330a03 041255a1	82099530 041255a1
Round[45]:	081c0e08 875a47d0	4f244517 1a4c6e70	2a330a03 041255a1	8a01c130 40f05ccf	0a03a000 9c49a00e	3e93d00d 09e0c054	4f244517 1a4c6e70	2a330a03 041255a1
Round[46]:	7a51c13a d1d7f052	081c0e08 875a47d0	4f244517 1a4c6e70	2a330a03 041255a1	3e93d00d 09e0c054	0a03a000 9c49a00e	af8f1a07 d09a0082	ef8c7c0a 0a0e582a
Round[47]:	4a0f0a0f 07e04341	7a51c13a d1d7f052	081c0e08 875a47d0	4f244517 1a4c6e70	0a03a000 9c49a00e	3e93d00d 09e0c054	4f244517 1a4c6e70	2a330a03 041255a1
Round[48]:	081c0e08 875a47d0	4a0f0a0f 07e04341	7a51c13a d1d7f052	081c0e08 875a47d0	856c455c 15a0c0b3	0a03a000 9c49a00e	af8f1a07 d09a0082	ef8c7c0a 0a0e582a
Round[49]:	8c0a0931 5c9a0917	081c0e08 875a47d0	4a0f0a0f 07e04341	7a51c13a d1d7f052	d20f0c0a 590c490d	856c455c 15a0c0b3	0a03a000 9c49a00e	af8f1a07 d09a0082
Round[50]:	2c78930a 1a4a730a	8c0a0931 5c9a0917	081c0e08 875a47d0	4a0f0a0f 07e04341	f0087492 2c005418	d20f0c0a 590c490d	856c455c 15a0c0b3	0a03a000 9c49a00e
Round[51]:	6a0f0a0f 07e04341	2c78930a 1a4a730a	8c0a0931 5c9a0917	081c0e08 875a47d0	0a03a000 9c49a00e	f0087492 2c005418	d20f0c0a 590c490d	856c455c 15a0c0b3
Round[52]:	73a0a0f0 8c0a0931	6a0f0a0f 07e04341	2c78930a 1a4a730a	8c0a0931 5c9a0917	2a72680d 2a058390	0a03a000 9c49a00e	af8f1a07 d09a0082	ef8c7c0a 0a0e582a
Round[53]:	9a030c14 22a021f7	73a0a0f0 8c0a0931	6a0f0a0f 07e04341	2c78930a 1a4a730a	3e93d00d 09e0c054	2a72680d 2a058390	0a03a000 9c49a00e	af8f1a07 d09a0082
Round[54]:	3a0a0a0f 7e01919e	9a030c14 22a021f7	73a0a0f0 8c0a0931	6a0f0a0f 07e04341	0a03a000 9c49a00e	3e93d00d 09e0c054	4f244517 1a4c6e70	2a330a03 041255a1
Round[55]:	9a0a0a0f 7e01919e	3a0a0a0f 7e01919e	9a030c14 22a021f7	73a0a0f0 8c0a0931	6a0f0a0f 07e04341	0a03a000 9c49a00e	af8f1a07 d09a0082	ef8c7c0a 0a0e582a
Round[56]:	4a0f0a0f 07e04341	9a0a0a0f 7e01919e	3a0a0a0f 7e01919e	9a030c14 22a021f7	af8f0a0f 9b5a0a0c	4a0f0a0f 07e04341	9a0a0a0f 7e01919e	3a0a0a0f 7e01919e
Round[57]:	c7502a24 e5c672ff	4a0f0a0f 07e04341	9a0a0a0f 7e01919e	3a0a0a0f 7e01919e	7a6e775f f97892a1	c7502a24 e5c672ff	4a0f0a0f 07e04341	9a0a0a0f 7e01919e
Round[58]:	0a0a0a0f 07e04341	c7502a24 e5c672ff	4a0f0a0f 07e04341	9a0a0a0f 7e01919e	af8f0a0f 9b5a0a0c	0a0a0a0f 07e04341	c7502a24 e5c672ff	4a0f0a0f 07e04341
Round[59]:	3a0a0a0f 7e01919e	0a0a0a0f 07e04341	c7502a24 e5c672ff	4a0f0a0f 07e04341	c26a35c0 21911a7a	3a0a0a0f 7e01919e	0a0a0a0f 07e04341	c7502a24 e5c672ff
Round[60]:	7a470a0d 3a0f6a0c	3a0a0a0f 7e01919e	0a0a0a0f 07e04341	c7502a24 e5c672ff	d0a0c1a0 8a050a0c	7a470a0d 3a0f6a0c	3a0a0a0f 7e01919e	0a0a0a0f 07e04341
Round[61]:	1a402a22 1a0c1258	7a470a0d 3a0f6a0c	3a0a0a0f 7e01919e	0a0a0a0f 07e04341	7c0921a0 d3082189	1a402a22 1a0c1258	7a470a0d 3a0f6a0c	3a0a0a0f 7e01919e
Round[62]:	3a0a0a0f 7e01919e	1a402a22 1a0c1258	7a470a0d 3a0f6a0c	3a0a0a0f 7e01919e	678a5a0a 8a0c911f	3a0a0a0f 7e01919e	1a402a22 1a0c1258	7a470a0d 3a0f6a0c
Round[63]:	8a01c130 40f05ccf	3a0a0a0f 7e01919e	1a402a22 1a0c1258	7a470a0d 3a0f6a0c	9c13721a 8a75803a	8a01c130 40f05ccf	3a0a0a0f 7e01919e	1a402a22 1a0c1258
Round[64]:	0a01c130 40f05ccf	8a01c130 40f05ccf	3a0a0a0f 7e01919e	1a402a22 1a0c1258	d2200598 d02743a8	0a01c130 40f05ccf	3a0a0a0f 7e01919e	1a402a22 1a0c1258
Round[65]:	0a01c130 40f05ccf	0a01c130 40f05ccf	3a0a0a0f 7e01919e	1a402a22 1a0c1258	fca70599 810b0009	0a01c130 40f05ccf	3a0a0a0f 7e01919e	1a402a22 1a0c1258

	A	B	C	D	E	F	G	H
Round[0]:	80215f15 339488fe	22321394 f020f72c	9f555f43 c84c84c2	23938808 4f938315	16f4c19f bcf4f312	96283462 8080ff43	06561429 33843992	20819f96 2c000808
Round[1]:	a4185800 96c41443	80215f15 339488fe	22321394 f020f72c	9f555f43 c84c84c2	0b406173 62606c48	16f4c19f bcf4f312	96283462 8080ff43	06561429 33843992
Round[2]:	12622a85 70a0408d	a4185800 96c41443	80215f15 339488fe	22321394 f020f72c	9f555f43 c84c84c2	0b406173 62606c48	16f4c19f bcf4f312	96283462 8080ff43
Round[3]:	04240410c 42016168	12622a85 70a0408d	a4185800 96c41443	80215f15 339488fe	c4074f6a 8339f926	81c18076 26c4e064	0b406173 62606c48	16f4c19f bcf4f312
Round[4]:	367812cc c84930e2	04240410c 42016168	12622a85 70a0408d	a4185800 96c41443	96cf4062 5360203a	c4074f6a 8339f926	81c18076 26c4e064	0b406173 62606c48
Round[5]:	f808ff5e 2f8af3ef	367812cc c84930e2	04240410c 42016168	12622a85 70a0408d	3a2145f7 4a037054	96cf4062 5360203a	c4074f6a 8339f926	81c18076 26c4e064
Round[6]:	406a0604 49c5af61	f808ff5e 2f8af3ef	367812cc c84930e2	04240410c 42016168	08006710 562f2a0d	3a2145f7 4a037054	96cf4062 5360203a	c4074f6a 8339f926
Round[7]:	76787078 08fae5a3	406a0604 49c5af61	f808ff5e 2f8af3ef	367812cc c84930e2	3a7e787e 12a4305c	08006710 562f2a0d	3a2145f7 4a037054	96cf4062 5360203a
Round[8]:	1c02f070 4774ca28	76787078 08fae5a3	406a0604 49c5af61	f808ff5e 2f8af3ef	80218005 67203746	3a7e787e 12a4305c	08006710 562f2a0d	3a2145f7 4a037054
Round[9]:	45830df9 91f6a028	1c02f070 4774ca28	76787078 08fae5a3	406a0604 49c5af61	924ac018 6751a609	80218005 67203746	3a7e787e 12a4305c	08006710 562f2a0d
Round[10]:	c92640f7 31c3a22c	45830df9 91f6a028	1c02f070 4774ca28	76787078 08fae5a3	0f813700 6537084a	924ac018 6751a609	80218005 67203746	3a7e787e 12a4305c
Round[11]:	08c40091 2743c0f2	c92640f7 31c3a22c	45830df9 91f6a028	1c02f070 4774ca28	809002f9 4226246d	0f813700 6537084a	924ac018 6751a609	80218005 67203746
Round[12]:	4359471f 95c5f32c	08c40091 2743c0f2	c92640f7 31c3a22c	45830df9 91f6a028	f808ff5e 2f8af3ef	809002f9 4226246d	0f813700 6537084a	924ac018 6751a609
Round[13]:	70446008 4c400c04	4359471f 95c5f32c	08c40091 2743c0f2	c92640f7 31c3a22c	70446008 4c400c04	f808ff5e 2f8af3ef	809002f9 4226246d	0f813700 6537084a
Round[14]:	41300091 4200c004	70446008 4c400c04	4359471f 95c5f32c	08c40091 2743c0f2	9546c24c 0392b461	70446008 4c400c04	f808ff5e 2f8af3ef	809002f9 4226246d
Round[15]:	3271a556 4739a384	41300091 4200c004	70446008 4c400c04	4359471f 95c5f32c	4a8a04a2 15085c43	9546c24c 0392b461	70446008 4c400c04	f808ff5e 2f8af3ef
Round[16]:	0a0af0c 3140a0c8	3271a556 4739a384	41300091 4200c004	70446008 4c400c04	0a0af0c 3140a0c8	4a8a04a2 15085c43	9546c24c 0392b461	70446008 4c400c04
Round[17]:	7823a00f 67c23157	0a0af0c 3140a0c8	3271a556 4739a384	41300091 4200c004	1004920d 5a910c03	0a0af0c 3140a0c8	4a8a04a2 15085c43	9546c24c 0392b461
Round[18]:	0b04af72 40f230b4	7823a00f 67c23157	0a0af0c 3140a0c8	3271a556 4739a384	8a0e9560 795d209e	0b04af72 40f230b4	7823a00f 67c23157	0a0af0c 3140a0c8
Round[19]:	19c0a0f8 41cf0ff4b	0b04af72 40f230b4	7823a00f 67c23157	0a0af0c 3140a0c8	979c0ff0 41240958	19c0a0f8 41cf0ff4b	0b04af72 40f230b4	7823a00f 67c23157
Round[20]:	4f237827 0a0c1c02	19c0a0f8 41cf0ff4b	0b04af72 40f230b4	7823a00f 67c23157	048a061f 3790d050	4f237827 0a0c1c02	19c0a0f8 41cf0ff4b	0b04af72 40f230b4
Round[21]:	45a00f16 f0001c7f	4f237827 0a0c1c02	19c0a0f8 41cf0ff4b	0b04af72 40f230b4	d100732 8a7ff704	45a00f16 f0001c7f	4f237827 0a0c1c02	19c0a0f8 41cf0ff4b
Round[22]:	42f02043 31770c03	45a00f16 f0001c7f	4f237827 0a0c1c02	19c0a0f8 41cf0ff4b	37c2f000 554279a9	42f02043 31770c03	45a00f16 f0001c7f	4f237827 0a0c1c02
Round[23]:	0f5f3007 f2009042	42f02043 31770c03	45a00f16 f0001c7f	4f237827 0a0c1c02	20c8a008 47402236	0f5f3007 f2009042	42f02043 31770c03	45a00f16 f0001c7f
Round[24]:	c5c6f011 71a07415	0f5f3007 f2009042	42f02043 31770c03	45a00f16 f0001c7f	00932000 414f9438	c5c6f011 71a07415	0f5f3007 f2009042	42f02043 31770c03
Round[25]:	12000753 470ef4c7	c5c6f011 71a07415	0f5f3007 f2009042	42f02043 31770c03	004c0f06 603f3448	12000753 470ef4c7	c5c6f011 71a07415	0f5f3007 f2009042
Round[26]:	132f3888 470ef4c7	12000753 470ef4c7	c5c6f011 71a07415	0f5f3007 f2009042	0f813700 6537084a	132f3888 470ef4c7	12000753 470ef4c7	c5c6f011 71a07415
Round[27]:	1462c0f 8a079a01	132f3888 470ef4c7	c5c6f011 71a07415	0f5f3007 f2009042	0ff40461 312000ff	1462c0f 8a079a01	132f3888 470ef4c7	c5c6f011 71a07415
Round[28]:	4146c095 31029c14	1462c0f 8a079a01	132f3888 470ef4c7	c5c6f011 71a07415	0f813700 6537084a	4146c095 31029c14	1462c0f 8a079a01	132f3888 470ef4c7
Round[29]:	1f6a180a 470ef4c7	4146c095 31029c14	1462c0f 8a079a01	132f3888 470ef4c7	0f813700 6537084a	1f6a180a 470ef4c7	4146c095 31029c14	1462c0f 8a079a01
Round[30]:	4072940c 08f60005	1f6a180a 470ef4c7	4146c095 31029c14	1462c0f 8a079a01	6773504f 03425604	4072940c 08f60005	1f6a180a 470ef4c7	4146c095 31029c14
Round[31]:	21080e4a d030004	4072940c 08f60005	1f6a180a 470ef4c7	4146c095 31029c14	31cf6f70 5a00d119	21080e4a d030004	4072940c 08f60005	1f6a180a 470ef4c7
Round[32]:	081f000a 083cf40d	21080e4a d030004	4072940c 08f60005	1f6a180a 470ef4c7	90975493 42630001	081f000a 083cf40d	21080e4a d030004	4072940c 08f60005
Round[33]:	7529a126 4a0c0104	081f000a 083cf40d	21080e4a d030004	4072940c 08f60005	0d7030a0 02041413	7529a126 4a0c0104	081f000a 083cf40d	21080e4a d030004
Round[34]:	9014970a 80921404	7529a126 4a0c0104	081f000a 083cf40d	21080e4a d030004	0d130130 4420502c	9014970a 80921404	7529a126 4a0c0104	081f000a 083cf40d
Round[35]:	0a19951f 0e0a0000	9014970a 80921404	7529a126 4a0c0104	081f000a 083cf40d	0d130130 4420502c	0a19951f 0e0a0000	9014970a 80921404	7529a126 4a0c0104
Round[36]:	0a19951f 0e0a0000	0a19951f 0e0a0000	9014970a 80921404	7529a126 4a0c0104	0d130130 4420502c	0a19951f 0e0a0000	0a19951f 0e0a0000	9014970a 80921404
Round[37]:	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	9014970a 80921404	0d130130 4420502c	0a19951f 0e0a0000	0a19951f 0e0a0000	9014970a 80921404
Round[38]:	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	9014970a 80921404	0a19951f 0e0a0000	0a19951f 0e0a0000	9014970a 80921404
Round[39]:	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	9014970a 80921404
Round[40]:	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	9014970a 80921404
Round[41]:	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	9014970a 80921404
Round[42]:	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	9014970a 80921404
Round[43]:	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	9014970a 80921404
Round[44]:	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	9014970a 80921404
Round[45]:	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	9014970a 80921404
Round[46]:	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	9014970a 80921404
Round[47]:	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	9014970a 80921404
Round[48]:	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	9014970a 80921404
Round[49]:	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	9014970a 80921404
Round[50]:	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	9014970a 80921404
Round[51]:	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	9014970a 80921404
Round[52]:	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	9014970a 80921404
Round[53]:	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	9014970a 80921404
Round[54]:	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	9014970a 80921404
Round[55]:	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	9014970a 80921404
Round[56]:	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	9014970a 80921404
Round[57]:	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	9014970a 80921404
Round[58]:	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	9014970a 80921404
Round[59]:	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	9014970a 80921404
Round[60]:	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	9014970a 80921404
Round[61]:	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	9014970a 80921404
Round[62]:	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	9014970a 80921404
Round[63]:	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	9014970a 80921404
Round[64]:	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	0a19951f 0e0a0000	9014970a 80921404
Round[65]:	0a19951f 0e0a0000</							

4.2.2. Comparison of SHA-1, SHA-2 and SHA-3

Each secure hash algorithm is compiled for each input message that has data size shown on table 4.1 and table 4.3. After compiling process timer value is sent from ARM microprocessor to personal computer via USART2 by using terminal program that is demonstrated at figure 4.12. All values are represented as milisecond at below tables.

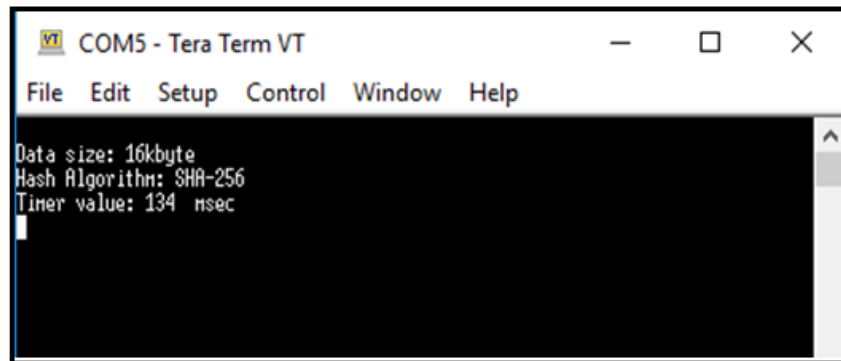


Figure 4.12: Demonstration of terminal program.

When table 4.1 and table 4.2 are examined, how fast SHA-1 algorithm is easily observed. Although SHA-1 consists of 80 rounds, it is very fast since the message digest length is 160 bits.

When SHA-224 and SHA-256 are considered, it is observed that the performance results are the same. Although SHA-224 message digest consists of 224 bits, the processes performed are exactly the same as SHA-256. At final process omitting of the last 32 bits of SHA-224 does not make it faster than SHA-256.

Considering the SHA-384, SHA-512, SHA-512/224 and SHA-512/256 algorithms, it is seen that process times are much longer than SHA-1, SHA-224 and SHA-256. Its processes take almost five times of SHA-1, and four times of SHA-224 and SHA-256. The reason for this situation can be stated as follows: Although SHA-1 consists of 80 rounds, the message digest length is 160 bits and 32 bit words are used. However, in SHA-384, SHA-512, SHA-512/224 and SHA-512/256 algorithms 64-bit words are used. This situation increases the process time very much. SHA-224 and SHA-256 are much faster than SHA-384, SHA-512, SHA-512/224 and SHA-512/256

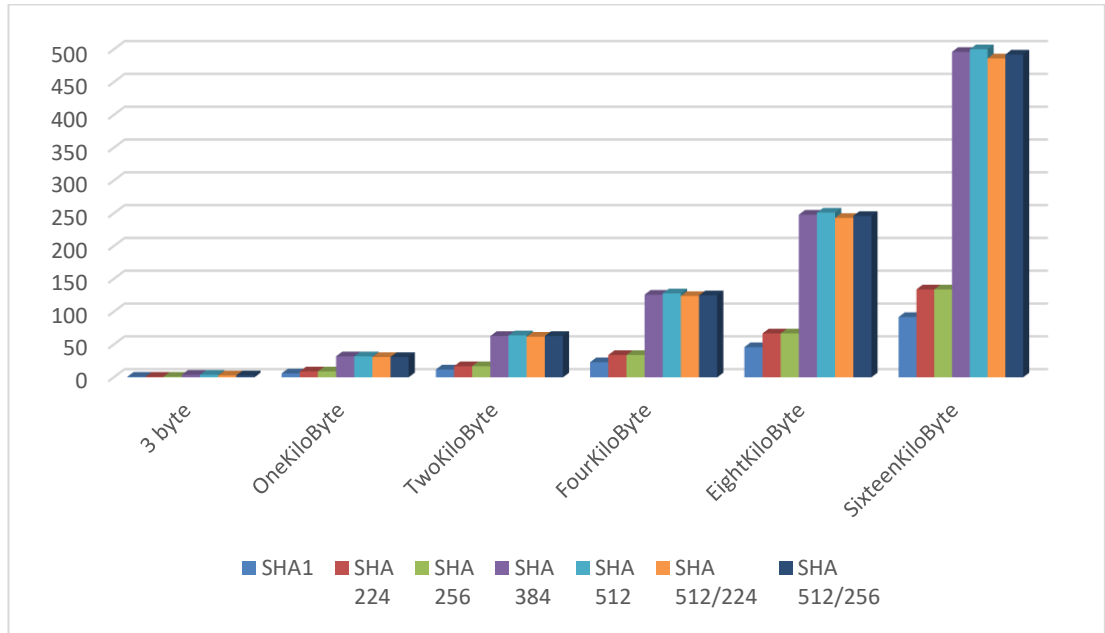
algorithms because again 32-bit words are used in SHA-224 and SHA-256 and also consist of 64 rounds that is less than the others.

There are a few differences that can be negligible between SHA-384, SHA-512, SHA-512/224 and SHA-512/256 algorithms in terms of time comparison.

Table 4.1: SHA-1 and SHA2 algorithms comparison(msec).

Data size / Algorithm	SHA1	SHA 224	SHA 256	SHA 384	SHA 512	SHA 512/224	SHA 512/256
3 byte	1	1	1	4	4	3	3
OneKiloByte	6	9	9	32	32	31	31
TwoKiloByte	12	17	17	63	64	62	63
FourKiloByte	23	34	34	126	128	124	125
EightKiloByte	46	67	67	248	251	243	246
SixteenKiloByte	92	134	134	496	502	486	492

Table 4.2: Graphic of SHA-1 and SHA2 algorithms comparison(msec).

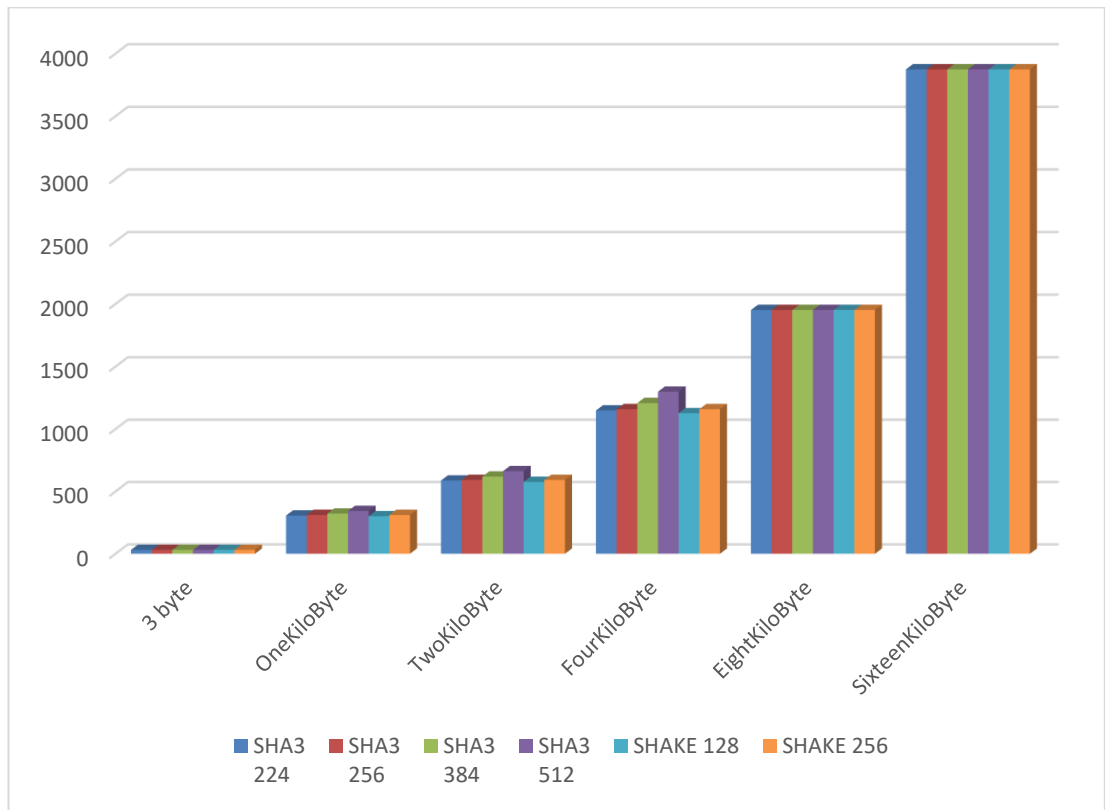


When time processes are examined on table 4.3 and table 4.4 it is observed that SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE128 and SHAKE256 algorithms take longer time than SHA-1 and SHA-2 algorithms. This is because the structure of SHA3 algorithm which is different from SHA-1 and SHA-2. Sponge construction structure that is used in SHA3 consists of many processes and this increases completion time of the algorithm compared to SHA-1 and SHA-2.

Table 4.3: SHA-3 and SHAKE algorithms comparison(msec).

Data size / Algorithm	SHA3 224	SHA3 256	SHA3 384	SHA3 512	SHAKE 128	SHAKE 256
3 byte	31	31	31	31	31	31
OneKiloByte	306	311	322	343	301	311
TwoKiloByte	587	592	619	662	576	592
FourKiloByte	1149	1159	1208	1299	1127	1159
EightKiloByte	1952	1952	1952	1952	1952	1952
SixteenKiloByte	3874	3874	3874	3874	3874	3874

Table 4.4: Graphic of SHA-3 and SHAKE algorithms comparison(msec).



There are a few differences that can be negligible between SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE128 and SHAKE256 algorithms in terms of time comparison.

A study conducted by Kuznetsov et al. to examine the performance of cryptographic hash algorithms in various microprocessors matches the data obtained in this thesis[15]. Ratio of obtained time for SHA-256 and SHA-512 algorithms in this thesis is approximately 3.74, while in the study conducted by Kuznetsov et al., this ratio is obtained as 3.43, 2.99, 4.19, 4.16, 4.34 on five different microprocessors[15].

Similarly, in this thesis ratio of obtained time for SHA3-256 and SHA3-512 algorithms is approximately 1,118, while at the study of Alexandr et al., this ratio is obtained as 1.004, 1.008, 1.019, 1.020, 1.017 in five different microprocessors[15].

In this thesis, step mapping which is used in all SHA3 algorithms increases the timing value approximately 22% while creating message digest. With a future work it is possible to speed up the step mapping process. As a result of that SHA3 algorithms can be used more effectively.

5. DISCUSSION AND CONCLUSION

In this thesis, on ARM microprocessor, message digests were obtained using SHA-1, SHA-2 and SHA-3 secure hash algorithms for input messages with a length range from three bytes to sixteen kilobytes. Thirteen secure hash algorithms SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256, SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE128 and SHAKE256 were examined in detail and tested on STM32F407VG ARM microprocessor using various length input messages.

Considering these thirteen algorithms: Since 32-bit words are used at SHA-1 algorithm and the message digest length is the shortest, the message digest is obtained faster than others. Moreover at SHA-224 and SHA256 algorithms, 32-bit words are used and message digest has been created at the same time period. When SHA-384 and SHA-512 are considered, the process time takes about five times more than SHA-1 and 4 times more than SHA-224 and SHA256. Because 64-bit words are used at these algorithms and the number of rounds are higher than SHA-224 and SHA256.

Since SHA-3 algorithms use sponge construction, it is observed that the process time takes much longer compared to SHA-1 and SHA-2 algorithms.

While all these algorithms are run on the STM32F407VG ARM microprocessor, it is determined how long the algorithm creates a message digest by using 32-bit timer on interrupt mode which is a feature of the microprocessor. This obtained time value is sent to a personal computer through the terminal program through USART2 communication which is also a feature of the microprocessor. Based on the obtained data, the performances of the secure hash algorithms were compared.

REFERENCES

- [1] Schneier B., (1996), “Applied Crptography”, 2nd Edition, Wiley & Sons Inc.
- [2] Stallings W., (2011), ” Cryptography and Network Security”, 5th edition, Prentice Hall.
- [3] Web 1, (2015), <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf>, (Access Date: 05/08/2015).
- [4] Web 2, (2015), <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf>, (Access Date: 05/08/2015).
- [5] Kim D., Lee S., Shin K., (2017), ”A Hardware Implementation of SHA3 Hash Processor using Cortex-M0”, International Conference on Electronics, Information, and Communication, Auckland, New Zealand, 22-25 January.
- [6] Wu X., Li S., (2017), “High Throughput Design and Implementation of SHA-3 Hash Algorithm”, International Conference on Electron Devices and Solid-State Circuits, 100-101, Hsinchu, Taiwan, 18-20 October.
- [7] Sideris A., Sanida T., Dasygenis M., (2019), “Hardware Acceleration of SHA-256 Algorithm using NIOS-II Processor”, 8th International Conference on Modern Circuits and Systems Technologies, Thessaloniki, Greece,13-15 May.
- [8] Gueron S., (2012), “Speeding up SHA-1, SHA-256 and SHA-512 on the 2nd Generation Intel® Core™ Processors”, Ninth International Conference on Information Technology - New Generations, 824-826, Las Vegas, NV, USA, 16-18 April.
- [9] Sobti R., Geetha G., Anand S., (2012), “Performance comparison of Grøstl, JH and BLAKE – SHA-3 Final Round Candidate Algorithms on ARM Cortex M3 Processor”, International Conference on Computing Sciences, 220-224, Phagwara, India, 14-15 Sept.
- [10] Laantansa, Saputra R., Noranita B., (2019), “Analysis of GPGPU-Based Brute-Force and Dictionary Attack on SHA-1 Password Hash”, 3rd International Conference on Informatics and Computational Sciences, Semarang, Indonesia, 29-30 October.
- [11] Chen S., Jin C., (2018), “Preimage Attacks on Some Hashing Modes Instantiating Reduced-Round Lblock”, IEEE Access, Volume: 6, 44659 – 44665, 10 August.

- [12] Web 3, (2019), https://www.st.com/resource/en/reference_manual/dm00031020-stm32f405-415-stm32f407-417-stm32f427-437-and-stm32f429-439-advanced-arm-based-32-bit-mcus-stmicroelectronics.pdf, (Access Date: 05/02/2019).
- [13] Web 4, (2017), https://www.st.com/resource/en/user_manual/dm00039084-discovery-kit-with-stm32f407vg-mcu-stmicroelectronics.pdf, (Access Date: 05/05/2017).
- [14] Web 5, (2011), <https://keccak.team/files/Keccak-reference-3.0.pdf>, (Access Date: 14/01/2011).
- [15] Kuznetsov A., Shekhanin K., Kolhatin A., Kovalchuk D., Babenko V., Perevozova I., (2019), "Performance of Hash Algorithms on GPUs for Use in Blockchain", IEEE International Conference on Advanced Trends in Information Theory, 166-170, Kyiv, Ukraine, 18-20 December.

BIOGRAPHY

Turgut ORHAN was born in 1987 at Zonguldak. He graduated from Istanbul Kultur University, Faculty of Engineering and Architecture, Department of Electronic Engineering at 2010, where he started at 2005. After graduation worked at Turkey Petroleum A.O. Batman Regional Directorate for one year, at EUAŞ Ermenek HEPP Operation Directorate for six years. He has been working as a researcher at the Information Technologies Institute TUBITAK since 2018. He successfully completed his master's education at 2020, which he started at Gebze Technical University at 2018. He has knowledge about embedded software, circuit design, system design. Besides English, he has a high level of Arabic and Persian languages.