

National Chiao Tung University

Department of Computer Science

Dissertation

一些密碼元件之分析與設計

Analysis and Design of Some Cryptographic Primitives

Student: Chen-Yu Lee

Advisor: Prof. Deng-Jyi Chen
Prof. Chu-Hsing Lin

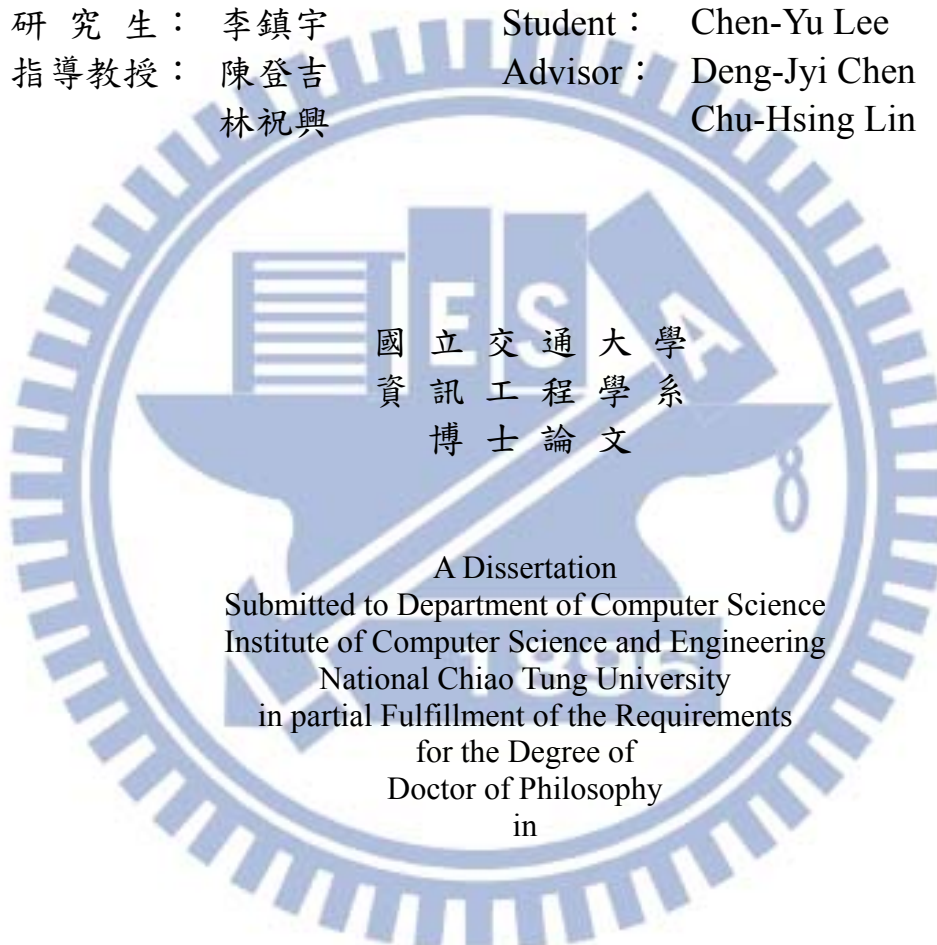
January, 2013

一些密碼元件之分析與設計

Analysis and Design of Some Cryptographic Primitives

研究生：李鎮宇
指導教授：陳登吉
林祝興

Student : Chen-Yu Lee
Advisor : Deng-Jyi Chen
Chu-Hsing Lin



國立交通大學
資訊工程學系
博士論文

A Dissertation
Submitted to Department of Computer Science
Institute of Computer Science and Engineering
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy
in

January 2013

Hsinchu, Taiwan, Republic of China

中華民國一百零二年一月

學生：李鎮宇

指導教授：陳登吉
林祝興

國立交通大學資訊工程與科學研究所博士班

摘 要

網路犯罪伴隨著網路的興起而成長，其核心價值——數位內容正面臨嚴重的威脅。本論文改良網路安全主要元件：對稱式加密演算法、單向雜湊函數以及安全協定的設計以及探討應用於隨意網路上金鑰管理的方法。

本論文替換了進階加密標準(AES)中回合函式的部分運算方法，並改以位元當作運算單位，使得可以抵抗三回合的平方攻擊法，以及線性攻擊法、差分攻擊法，得以證明在許多方面比 AES 優良。本研究也基於安全雜湊演算法(SHA)的設計精神，定義了一般性的 SHA，其接受任意長度訊息輸入，並產生所需要長度的訊息摘要。本研究提出一個新的觀點，以評估 SHA-256-XOR 演算法的安全複雜度，即是計數每個演算方程式中所牽涉的項數，以取代計算碰撞機率的方法。引用基因演算法探究訊息排程中趨近最佳的參數組合，使相對於標準

方法可以提升 1.5 到 4 倍的安全複雜度。最後，本論文改良了秘密分享機制並應用於金鑰管理方法以減少通訊、計算量的花費。

本論文的貢獻將會讓非模加安全雜湊運算的研發者感到興趣，而這樣的運算方式會有利於使用較少邏輯閘的硬體實作。另外，本論文所提出的方法論亦可以應用於所有引用秘密分享機制的設計方法以減少訊息長度而不會降低安全程度。



Analysis and Design of Some Cryptographic Primitives

Student : Chen-Yu Lee

Advisors : Dr. Deng-Jyi Chen
Dr. Chu-Hsing Lin

Institute of Computer Science and Engineering
National Chiao Tung University

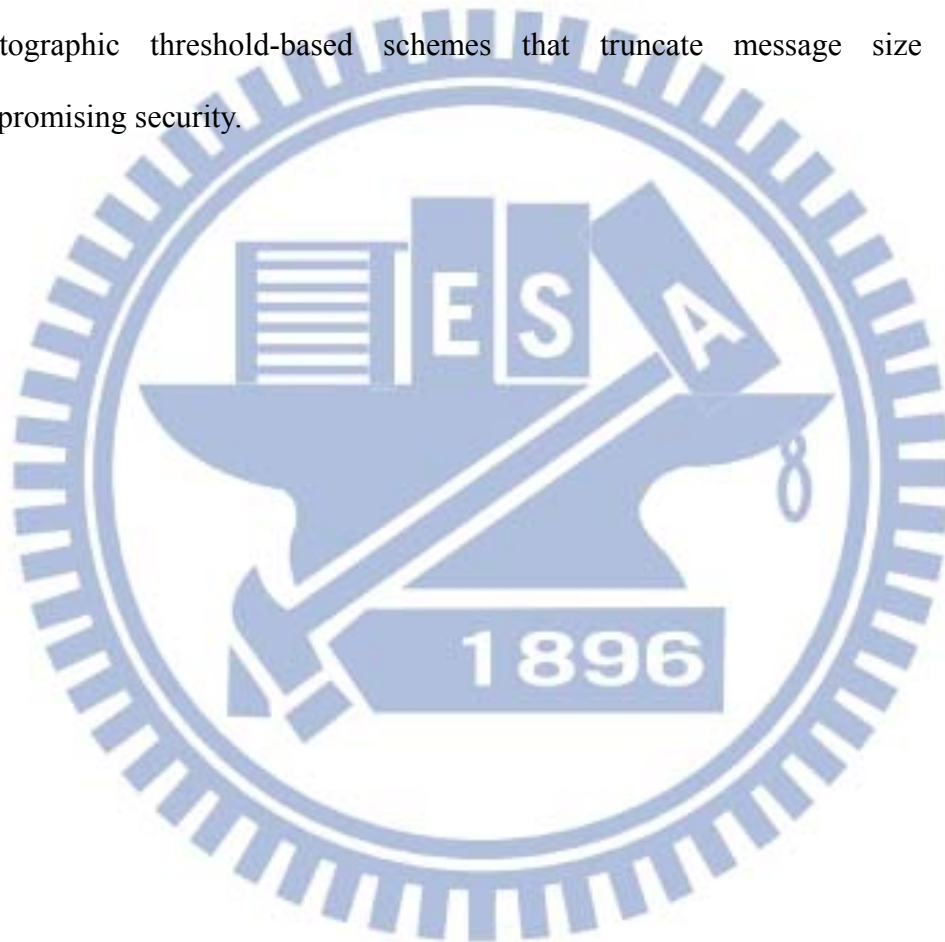
ABSTRACT

Increasing cybercrime activities on the Internet introduces various threats to core values and digital content. This dissertation improves the design of symmetric cipher algorithms and one-way hash functions, and clarifies the functions of key management in mobile ad hoc networks.

We replace some procedures in the round function of the advanced encryption standard (AES) and use bits as the operation unit to foil the 3-round square attack. Moreover, we apply linear cryptanalysis and differential cryptanalysis to the proposed cipher, which is superior to AES. Our study defines a generalized secure hash algorithm (SHA) algorithm based on SHA family rules. The algorithm accepts arbitrary length messages as inputs that generate message digests with the required length. We propose a new perspective of complexity for SHA-256-XOR functions by counting the terms involved in each equation, instead of analyzing the probability of finding collisions within SHA-256-XOR hash functions. We apply genetic algorithms to find the near-optimal message schedule parameter sets that enhance the complexity 4 times for SHA-1 and 1.5 times for SHA-256-XOR, when compared to their original

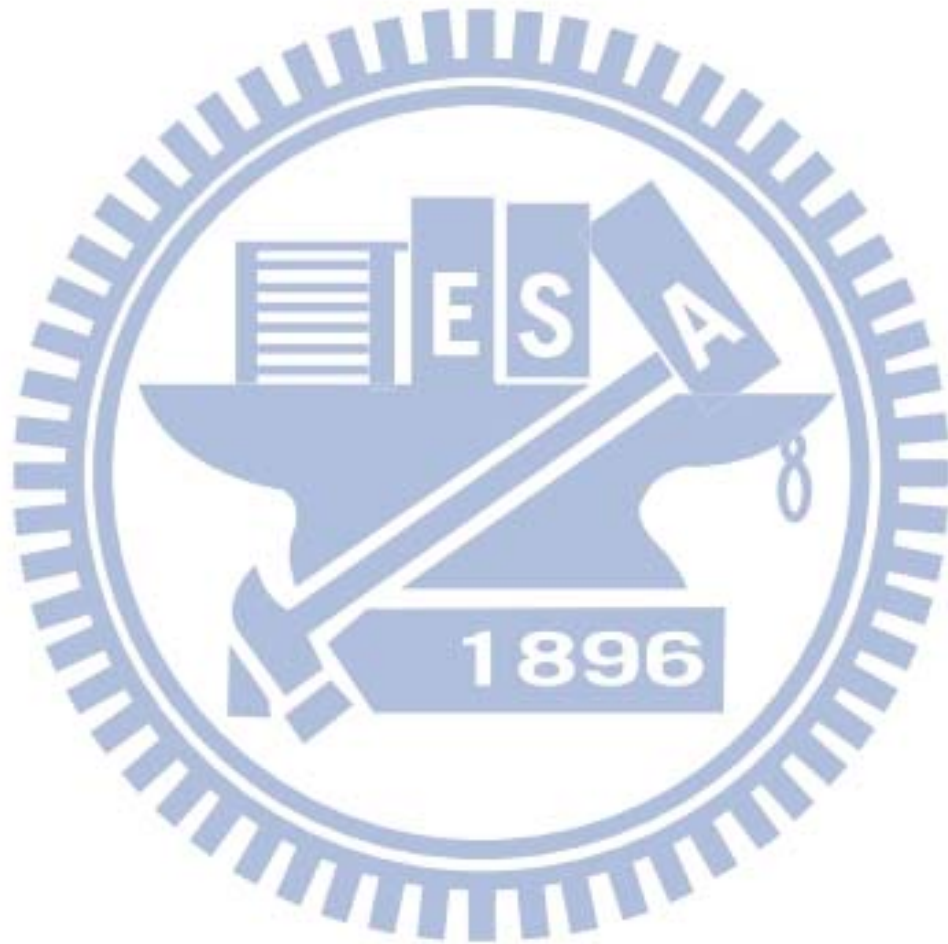
SHA-1 and SHA-256-XOR functions. Finally, we modify the secret sharing scheme and apply it to autonomous key management (AKM) for reducing communication and computation costs.

Our results are useful when designing security for modular-addition-free hash functions, simplifying hardware implementation and allowing a smaller gate count, and when designing symmetric ciphers. The proposed methodology applies to all cryptographic threshold-based schemes that truncate message size without compromising security.



Dedication

To my parents, Heng-Hsing Lee and Li-Hua Mao, and my wife, Yi-Ting Chen,
for their unwavering support and encouragement over the years.



Acknowledgement

During the past 3700 days, I have received encouragement and support from several people. Now, upon the completion of the dissertation for my doctoral study, I would like to offer my gratitude. First, I must thank my supervisor, Dr. Yi-Shiung Yeh, who dedicatedly guided me through my study, up to the time of his passing. His inspiration triggered my passion for Cryptology. Even though he will not be able to see the result of my study, I have embraced his foresight. In addition, I would like to thank my other supervisor, Dr. Deng Jyi-Chen, who helped me after Dr. Yeh passed away. He accepted me and offered me guidance wholeheartedly, leading me into the domain of software development and medical information and enabling me to broaden my perspective. Furthermore, I am most grateful to Dr. Chu-Hsing Lin for offering guidance and care since the beginning of my master program. He has not only led me through the study of information security and steganography but has also taught me much about interpersonal interactions. Most of all, I tender my gratitude to the committee of the doctoral dissertation, Professor Chien-Chao Tseng, Lein Harn, Jinn-Ke Jan, Shih-Kun Huang and Chorng-Shiuh Koong, as they have shared precious suggestions and opinions.

Finally, I would like to thank Dr. Min-Chih Kao, Dr. I-Te Yiter Chen, Dr. Wei-Shen Lai, Dr. Chia-Yin Lee, Dr. Tzer-Long Chen, Mr. Ting-Yu Huang, Dr. Ching-Wen Cheng, Dr. Lin-Chuan Wu, and my classmates for the encouragement and discussions that helped me complete my dissertation.

Contents

摘 要.....	i
ABSTRACT.....	iii
Dedication.....	v
Acknowledgement.....	vi
Contents.....	vii
Table of Contents.....	x
List of Figures.....	xi
1. Introduction.....	1
2. Definitions.....	9
2.1 Terms and Acronyms.....	9
2.2 Algorithm parameters, symbols and terms.....	9
2.3 Symbols and operations.....	11
3. Related Works.....	12
3.1 Related Works on AES.....	12
3.1.1 Function SubBytes.....	13
3.1.2 Function ShiftRows.....	16
3.1.3 Function MixColumns.....	16
3.1.4 Function AddRoundKey.....	17
3.1.5 Function Key Expansion.....	17
3.1.6 Inverse Cipher.....	18
3.1.7 Function InvShiftRows.....	19
3.1.8 Function InvShiftRows.....	19
3.1.9 Function InvMixColumns.....	20

3.2 Related Works on SHA Family	21
3.2.1 Overview of SHA-1, SHA-224 and SHA-256 Algorithms.....	22
3.2.2 Overview of SHA-384 and SHA-512 Algorithms	26
3.3 Genetic Algorithm.....	30
3.4 Secret Sharing Scheme	32
3.4.1 The Shamir (t, n) -Threshold Scheme in \mathbb{Z}_p	32
3.4.2 Share Distribution	32
3.4.3 Proactive Security	33
3.5 Autonomous Key Management (AKM).....	33
4. Our Proposed Schemes	35
4.1 A Transpositional Advanced Encryption Standard (AES) Resists 3-round Square Attack	35
4.1.1 Cipher Structure	35
4.1.2 Our Proposed AES_Plus	37
4.2 Generalized Secure Hash Algorithm: SHA-X.....	43
4.2.1 Generalized Secure Hash Algorithm.....	43
4.2.2 SHA(x) Family.....	51
4.3 Finding Near-Optimum Message Scheduling Settings for SHA-256 Variants Using Genetic Algorithms	57
4.3.1 SHA Message Scheduling Evaluation Criterion.....	57
4.3.2 Improving SHA-256-XOR Via Genetic Algorithms.....	61
4.4 Modified Autonomous Key Management.....	65
4.4.1 Function Update.....	66
4.4.2 Function Join.....	67
4.4.3 Function Leave.....	68

4.4.4 Function Merge	68
4.4.5 Function Partition.....	69
4.4.6 Function Expansion	71
4.4.7 Function Contraction	71
5. Discussion and Analyses.....	73
5.1 Cryptanalysis of Transpositional AES	73
5.1.1 Linear Cryptanalysis	73
5.1.2 Differential Cryptanalysis	76
5.1.3 Square Attack.....	79
5.2 Experiment results of SHA-256-XOR.....	82
5.3 Performance analysis of Modified Autonomous Key Management.....	84
6. Conclusions.....	86
References.....	89

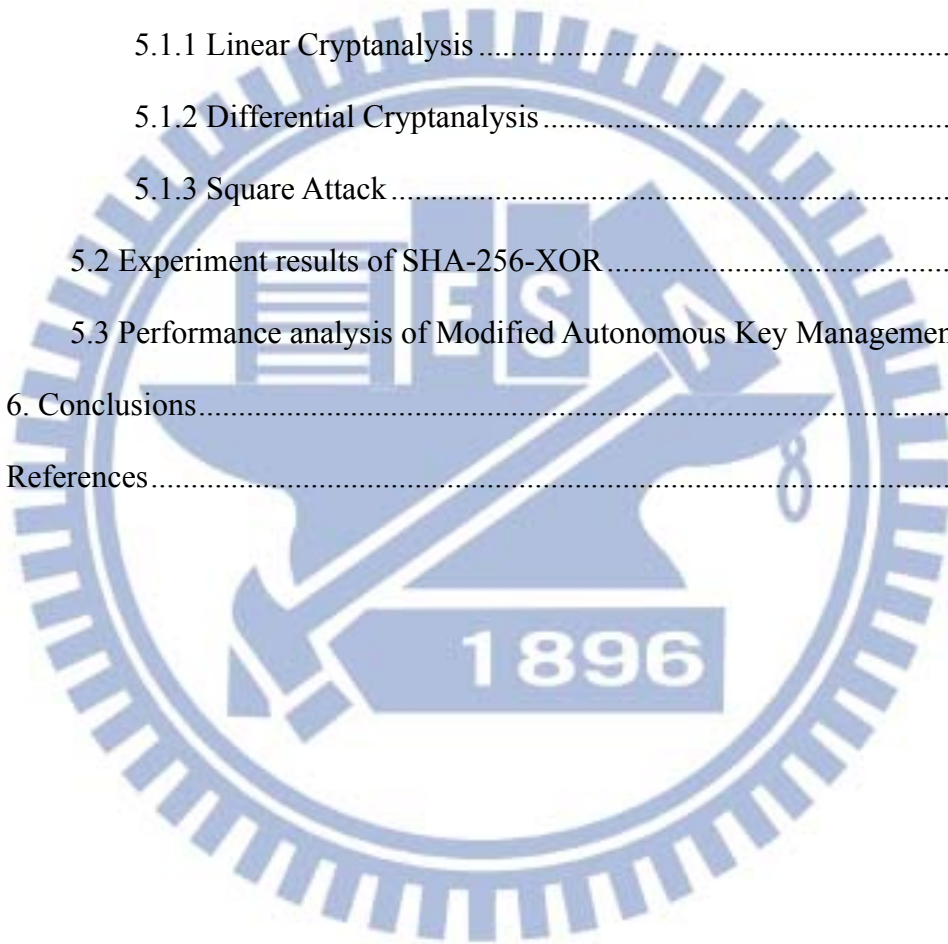


Table of Contents

Table 1 SHA algorithms.....	4
Table 2 Key-Block-Round Combinations.....	12
Table 3 Boolean function and constants used in SHA-1.....	23
Table 4 The initial hash value, $H^{(0)}$ in SHA-1.....	23
Table 5 Boolean function used in SHA-224 and SHA-256.....	25
Table 6 Constants in SHA-224 and SHA-256 (From left to right, up to down).....	26
Table 7 The initial hash value, $H^{(0)}$ in SHA-224 and SHA-256.....	26
Table 8 Boolean function used in SHA-384 and SHA-512.....	28
Table 9 Constants in SHA-385 and SHA-512 (From left to right, up to down).....	29
Table 10 The initial hash value, $H^{(0)}$ in SHA-384 and SHA-512.....	30
Table 11 Values of m and n for SHA family.....	44
Table 12 SHA(2).....	52
Table 13 SHA-0, SHA-1, and SHA-256-XOR equations.....	58
Table 14 Genetic algorithm parameters.....	64
Table 15 Differences between AES_Plus and AES.....	73
Table 16 Largest bias of state bits at the end of each round (up to the tenth round)	76
Table 17 Highest probabilities of differential trails for two ciphers up to some rounds.....	79
Table 18 The last 10 generations of the simulation.....	82
Table 19 Message length comparison.....	84
Table 20 Operand length comparison.....	85

List of Figures

Figure 1 S-box of AES.....	15
Figure 2 <i>ShiftRows()</i> in AES.....	16
Figure 3 <i>InvShiftRows()</i> in AES.....	19
Figure 4 Inverse S-box of AES.....	20
Figure 5 Flowchart of genetic algorithm.....	31
Figure 6 An example of AKM.....	34
Figure 7 The structure overview of the proposed cipher where \oplus denotes the round key addition.....	37
Figure 8 <i>Round_Plus</i> of <i>AES_Plus</i>	39
Figure 9 <i>FinalRound_Plus</i> of <i>AES_Plus</i>	39
Figure 10 <i>TransByte</i>	40
Figure 11 Sub-Block XOR.....	41
Figure 12 Encryption and decryption of <i>AES_Plus</i>	42
Figure 13 Initial values of standard SHA family.....	46
Figure 14 Initial values of SHA-192 and SHA-448.....	47
Figure 15 Initial values of SHA- <i>mn</i>	48
Figure 16 Comparison of the number of terms involved in each W_i in message scheduling for SHA-0, SHA-1 and SHA-256.....	60
Figure 17 Comparison of the number of terms involved in each W_i in message scheduling for SHA-1 and SHA-1-OPT.....	61
Figure 18 Function Merge – merges S_i into S_j and S_k	69
Figure 19 Function Partition – partition of S_i into S_j and S_{m+1}	70
Figure 20 Function Expansion.....	71
Figure 21 Example of differential trails into the S-box.....	77
Figure 22 Example of joining of differential trails in XOR Operation.....	78
Figure 23 The influence of active byte Λ_l in 1 st round.....	80
Figure 24 The influence of active byte Λ_l in 2 nd round.....	80
Figure 25 The influence of active byte Λ_l in 3 rd round.....	81
Figure 26 The influence of active byte Λ_r in 1 st round.....	81
Figure 27 The influence of active byte Λ_r in 2 nd round.....	81
Figure 28 The influence of active byte Λ_r in 3 rd round.....	81
Figure 29 Comparison of the number of terms involved in each W_i in message scheduling for SHA-256-XOR and optSHA-256-XOR.....	83
Figure 30 Comparison of the running time in W_i between genetic algorithm and brute force.....	83

1. Introduction

According to The Cluster of European Research projects on the Internet of Things (CERP-IoT) in 2010 [1], “Over the next 10 to 15 years, the Internet of Things is likely to develop fast and shape a newer ‘information society’ and ‘knowledge economy’.” The common feature of the terms “knowledge economy” and “Internet of Things” is digital content. The former considers knowledge (digital content) to be the most important economic resource, basic production factor, and the main driver of development [2], and the latter allows connected sensors to promote interactions for ubiquitous access to digital content.

However, digital content and Internet users remain prone to various security threats. It is necessary to establish a security framework covering various scenarios, e.g., supply chains and air travel, with interrelated factors including safety, privacy, and economy [3]. Without a secure framework, losses due to attacks will outweigh any benefits. Security frameworks require optimal cryptography mechanisms, key management systems, and security protocols. Possible mechanisms include symmetric algorithms, asymmetric algorithms, one-way hash functions, and random number generators.

Symmetric algorithm

On October 2, 2000, the National Institute of Standards and Technology (NIST) announced that Rijndael had been selected as the proposed Advanced Encryption Standard (AES) and began the process of making it the official standard. On November 26, 2001, NIST announced the AES as Federal Information Processing Standards Publication (FIPS PUB) 197. The National Security Agency (NSA) stated

all AES finalists, including Rijndael, were secure enough for US government non-classified data. In June 2003, the US government announced that AES should be used for classified information.

AES suffers from many attacks such as linear cryptanalysis [4], differential analysis [5] [6] and square attack [7] [8]. Impossible differential attacks [5] use differential probability to eliminate the key material for finding the right key candidate for AES. The 4-round impossible differential cryptanalysis of AES were proposed in [9] [10] [11]. In 2000, E. Biham and N. Keller presented an impossible differential attack on 5-round AES-128 in [6]. Later in Cheon et al. improved the attack to 6-round AES-128 in [12]. In 2004 Phan [13], and Chen Jie et al. [14] gave attacks on 7-round AES-192 and AES-256 exploiting weaknesses in the key schedule. In 2007 Wentao Zhang et al [15] enhanced the attack on 8-round AES-192 and AES-256. In [16], E. Biham et al also successfully attacked 8-round AES-192 by related-key impossible differential attack. The square attack on AES was presented by an AES designer in [7] [8].

[17] describes the properties of cryptographically robust S-boxes as high nonlinearity, balanced output, immunity against linear cryptanalysis, robustness against differential cryptanalysis, avalanche effect and high algebraic degree of its output Boolean functions. The above cryptanalysis seems to focus on the design of the S-box to increase the complexity of the algebraic expression of the AES S-box to render it capable of resisting the known powerful differential cryptanalysis from 2005. In [18], A. Grochowska-Czurylo and J. Stoklosa found a deterministic algorithm to construct bent functions for random generation S-boxes. In 2007, D. Bhattacharya et al proposed a cellular automata-based structure S-box design which showed itself strongly resistant to linear cryptanalysis, differential cryptanalysis, algebraic attack and power attack in [19]. L. Cui and Y. Cao proposed an Affine-Power-Affine (APA)

S-box structure that increases the complexity of algebraic expression from 9 to 255 [20].

Instead of improving the S-box design, the focus of previous research for defending against linear cryptanalysis and differential cryptanalysis, this research varies the cipher structure in AES to resist square attack while keeping basic security. Due to the byte-oriented structure of AES, the square attack can be applied effectively. This work replaces some functions in the round transformation of AES and takes the bit as the operation unit to avoid 3-round square attacks. Applying linear cryptanalysis and differential cryptanalysis to our proposed block cipher, the results show our proposed cipher can resist these attacks in five and four rounds, respectively.

One-way hash functions

Cryptographic hash functions play an important role in modern cryptography. They are widely used in a variety of applications such as password protection, secure protocols, digital signatures, and more. The hash function uses a string of arbitrary length as its input and creates a fixed-length string as its output. A hash value is often called a data fingerprint or message digest. The following sections provide some definitions of collision-free hash functions.

Secure Hash Algorithm (SHA) is a series of cryptographic hash functions published by the National Institute of Standards and Technology (NIST). NIST published SHA as FIPS PUB 180-4 [21] consisting of seven algorithms: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224 and SHA-512/256.

Recent studies have proposed extensions based on SHA. For example, RAR-SHA-256 [22] is composed of the SHA-256 compression function, and is faster

than SHA-256 when implemented in parallel. SHACAL and SHACAL-2 [23] [24] are block ciphers that are based on SHA-1 and SHA-256, respectively, and which were submitted to the New European Schemes for Signatures, Integrity, and Encryption project (NESSIE) in 2003. Yoshida and Biryukov replaced all arithmetic additions with XOR operations in SHA-256, naming it SHA-256-XOR, and found that SHA-2-XOR has a pseudo-collision resistance weakness up to 34 rounds [25].

A birthday attack [11] [26] is a type of cryptographic attack based on the birthday problem in probability theory. Given a function f , the attack attempts to find two different inputs x_1, x_2 such that $f(x_1) = f(x_2)$. Such a pair (x_1, x_2) is called a collision input. The birthday attack on a message digest of size n produces a collision after trying $1.2 \times \sqrt{2^n} \approx 2^{n/2}$ input values. Under the birthday attack, the security of SHA-1, SHA-192, SHA-224, SHA-256, SHA-384, SHA-448, and SHA-512 are approximately 2^{80} , 2^{96} , 2^{112} , 2^{128} , 2^{192} , 2^{224} , and 2^{256} , respectively, and are listed in Table 1. Many researchers have tried to develop a cryptanalytic method with a lower complexity than the birthday attack.

Table 1 SHA algorithms

Algorithm	Message Size (bits)	Block Size (bits)	Word Size (bits)	Message Digest Size (bits)	Security
SHA-1	$< 2^{64}$	512	32	160	2^{80}
SHA-224	$< 2^{64}$	512	32	224	2^{112}
SHA-256	$< 2^{64}$	512	32	256	2^{128}
SHA-384	$< 2^{128}$	1024	64	384	2^{192}
SHA-512	$< 2^{128}$	1024	64	512	2^{256}

SHA-512/224	$< 2^{128}$	1024	64	224	2^{112}
SHA-512/256	$< 2^{128}$	1024	64	256	2^{128}
The term security in this table means that a birthday attack on a message digest of size n produces a collision with a factor of approximately $2^{n/2}$.					

In 1998, Chabaud and Joux announced a method for finding the SHA-0 collisions [27]. They reduced this complexity to 261 using a differential cryptanalysis technique, but they could not successfully apply it to SHA-1. This result implied that SHA-1 is more secure than SHA-0. In early 2005, Rijmen and Oswald applied the same method to find collisions in SHA-1 [28]. They examined message scheduling in SHA-0 and SHA-1, and proved that the complexity associated with finding collisions in a reduced version of SHA-1 (with 53 rounds instead of 80 rounds) was less than 2^{80} . Wang, Yin, and Yu found collisions with a complexity of 2^{69} in the full 80-step SHA-1 [29]. In 2010, Grechnikov announced the practical collision attack on the 73-step SHA-1 based on an automated approach [30]. NIST announced that SHA-1 will be used until 2010, at which time it will be replaced by SHA-2.

Since 2004, several authors have reported on collisions for SHA-256. Gilbert and Handschuh reported a 9-round local collision with a complexity of 2^{66} using differential path analysis [31]. Mendel et al. later reduced this complexity to 2^{39} [32]. Nikolić and Biryukov realized 21-step collisions for SHA-256 using a nonlinear differential path analysis with a complexity of 2^{19} [33]. In 2008, Sanadhya and Sarkar found a local collision with 24-step SHA-256 and SHA-512 with $2^{28.5}$ and $2^{32.5}$ calls, respectively [34], and this was the first time that a colliding message pair for 24-step SHA-512 was provided. In 2009, Indestege et al. found collisions on the 24-step SHA-256 and SHA-512 with $2^{28.5}$ calls and 2^{53} calls, respectively, and a local collision on 31-step SHA-256 with 2^{32} [35]. Also in 2009, Aoki et al. presented full

preimage attacks on up to 43-step SHA-256 and SHA-512 with the time complexities of $2^{254.9}$ and $2^{511.5}$ compression function operations for full preimages, respectively [36]. Since 2011, Mendel et al. have presented a collision on 27-step SHA-256 and a semi-free-start collision on 32-step SHA-256 with practical complexity [37]. Biryukov et al. presented a second-order differential collision for the SHA-256 compression function on 47 out of 64 steps, which have practical complexity based on a rectangle/boomerang approach [38].

Almost all of the currently known cryptanalyses of SHA have attempted to find collisions on a differential path. However, the design of each component such as algorithms for message scheduling and hash loop body and the function parameters, affects the possibility that a path for collisions (using differential path cryptanalysis) will be found. A fairly large body of literature exists regarding methods of improving hash algorithms. However, there is a surprising lack of information regarding the design and selection of function parameters. This paper addresses this deficiency.

The purpose of the research presented in this dissertation is to examine the relationship between the security of a hash function and its function parameters. In this regard, two issues that need to be resolved are (a) how to assess the security fitness of a given set of function parameters, and (b) how to find the optimal function parameter set. Specifically, this paper proposes a novel view of complexity (hence security fitness) of SHA-2-XOR functions proposed in [25], by counting the terms involved in each equation, instead of analyzing the probability of finding collisions within an SHA-256-XOR hash function. Our experiments have shown that the parameter set in each equation of a message schedule plays an important role in security fitness, but it is very hard to find the optimum parameter values. We apply genetic algorithms to find the optimal message schedule parameter sets that enhance the complexity 4 times for SHA-1 and 1.5 times for SHA-256-XOR, when compared

to original SHA-1 and SHA-256-XOR functions. The analysis results would be interesting for designers who are interested in the security of modular addition free hash functions, which are good for hardware implementation with lower gate counts. Moreover, the found message schedule parameter sets would be a good reference for further improvement of SHA functions.

The dissertation also defined a *generalized SHA* algorithm based on SHA family rules. The algorithm contains the initial values, constant values, padding, parsing, as well as the main body, and accepts arbitrary length message as input to generate message digest with required length. Further, the study solved *Length-of-the-Hash-Value (LHV)* problem that occurs when $SHA-r$ cannot be expressed as $r = mn$ uniquely.

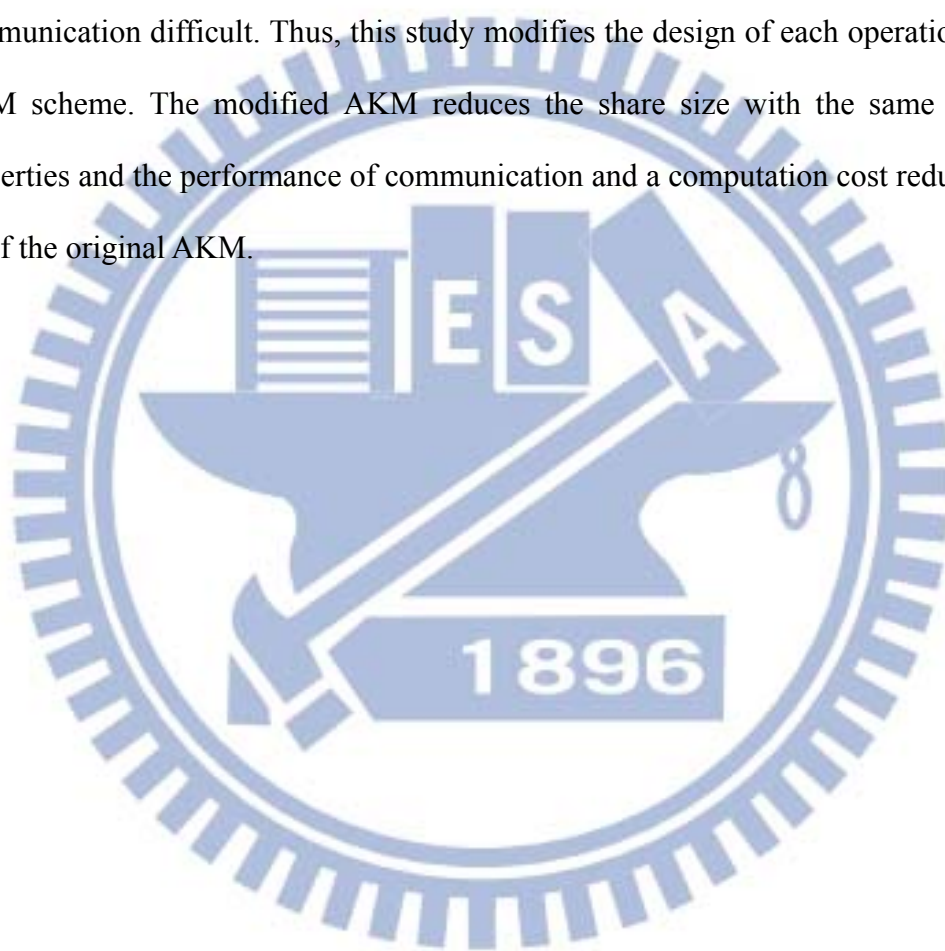
Secure protocols

Key management within a *Mobile Ad hoc Network (MANET)* is a security issue that cannot be ignored. Many researchers have dedicated themselves to this field since 1999. Some schemes are suitable for a limited number of nodes and are inefficient, insecure, or unreliable when the nodes increase [39] [40] [41] [42] [43] [44] [45] [46] [47] [48] [49] [50]. Nodes may join the MANET and leave later normally. Thus, the key management scheme in MANET must be dynamic. The main challenge of MANET is that each node handles the joining or leaving of nodes with the limited resources, such as CPU computation, storage, and the power consumption [51]. The mobility of a MANET increases its unreliability and limits the bandwidth of wireless environment due to frequent topology changes.

B. Zhu et al. proposed a key management scheme [52] using the secret sharing method [53] [54] [55] [56] to construct an Autonomous Key Management (AKM)

hierarchy structure with flexibility and adaptivity. This scheme needs no central party to control the key structure, and each node cooperates to create virtual nodes in building the key hierarchy. The method proposed in [57] dynamic group key management schemes with forward secrecy and backward secrecy based on elliptic curve cryptosystem (ECC) [58], forming a self-certified public key cryptosystem [59].

However, a message of 2048 bits would make computing or calculating AKM communication difficult. Thus, this study modifies the design of each operation in the AKM scheme. The modified AKM reduces the share size with the same security properties and the performance of communication and a computation cost reduction to $1/t$ of the original AKM.



2. Definitions

2.1 Terms and Acronyms

Λ -set	A set of 256 states that differ in active bytes and are equal in passive bytes.
AES	Advanced Encryption Standard.
AKM	Autonomous Key Management.
APA	Affine-Power-Affine S-box structure.
Bit	A binary digital having a value of 0 or 1.
Byte	A group of eight bits.
CRL	Certificate Revoking List.
ECC	Elliptic Curve Cryptography.
GTC	The AKM sets a Global Trust Coefficient as a lower bound of all the RTC.
LHV	Length-of-the-Hash-Value problem.
MANET	Mobile Ad Hoc Networks.
NESSIE	New European Schemes for Signatures, Integrity, and Encryption project.
NIST	National Institute of Standards and Technology.
ORS	Overall Region Size is the number of the nodes that know the secret of region.
RTC	Regional Trust Coefficient is the ratio of the threshold to ORS.
SHA	Secure Hash Algorithm.
Word	A group of either 32 bits (4 bytes) or 64 bits (8 bytes), depending on the secure hash algorithm.

2.2 Algorithm parameters, symbols and terms

Λ_l	Λ -set at a byte of left 64 bits.
Λ_r	Λ -set at a byte of right 64 bits.
(t, n) -threshold	A secret key K can be recovered by t out of total n shares.
$\{t, A, B, C, D\}$	The parameter set of W_t equation in message scheduling.
$b_{i,j}$	The element with i^{th} row and j^{th} column of a matrix.

D	The dealer of secret sharing scheme.
$F(t)$	The number of different terms involve in W_t equation.
g	Random number generator.
$H(M)$	The hash function $H()$ with input M .
IV	Initial value of a hash function.
K	Secret key for symmetric cipher in a cryptosystem.
$K_t^{\{mn\}}$	The constant value to be used for the iteration t of the SHA- mn hash function.
l	Length of the message, M , in bits, $l = M $.
M	Message with arbitrary length as the input of a hash function.
m	The number of words in a message digest.
$M^{(i)}$	Message block i .
MD	Message digest which is the output of a hash function with fixed length.
$M_j^{(i)}$	The j^{th} word of the i^{th} message block, where $M_0^{(i)}$ is the left-most word of message block i .
M_j^n	The message block M_j with n -bitwise left rotation.
n	The number of bits in a word.
Nr	The total number of encryption rounds.
P_i	The i^{th} participant of secret sharing scheme.
PK_i	The public key of node i for asymmetric cipher in cryptosystem.
r	The value of $m \times n$, $r = mn$.
$S_{(i,j),k}$	The share of region $S_{(i,j)}$.
$SHA(x)$	Generalized SHA family
$SHA-r$	SHA with r bits digest output.
S_i	A region with a three with node i as root.
$S_{i,j}$	The share is distributed from node i to node j .
SK_i	The secret key of node i for asymmetric cipher in cryptosystem.
W_j^n	The message word W_j with n -bitwise left rotation.
W_t	The t^{th} word of the message schedule.

2.3 Symbols and operations

\cap	Set INTERSENTION operation.
\wedge	Bitwise AND operation
\vee	Bitwise OR operation
\oplus	Bitwise XOR operation
\neg	Bitwise complement operation
$ROTL^{i}(x)$	Rotate left operation by i bits.
$SHR^{i}(x)$	Shift right operation by i bits.



3. Related Works

3.1 Related Works on AES

AES algorithm is specified with a fixed block size of 128 bits ($Nb = 4$), a key size of 128, 192, or 256 bits ($Nk = 4, 6, 8$), and referred as AES-128, AES-192, AES-256. It is capable of using any key and block size for all multiples of 32 bits. The key is expanded using Rijndael's key schedule. Most AES computations are done in a special finite field. AES operates on a 4×4 array of bytes called the *state*. The number of rounds (Nr) to be performed during the execution of the algorithm is dependent on the key size.

Table 2 Key-Block-Round Combinations

	Key Length (Nk words)	Block Size (Nb words)	Number of Rounds (Nr)
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

For encryption, each round of AES (except for the last round, which omits the *MixColumns()* stage) consists of four stages.

The four stages of AES are explained as follows:

- *SubBytes()*: a non-linear substitution step where each byte is replaced with another according to a lookup table.
- *ShiftRows()*: a transposition step where each row of the state is shifted cyclically by a certain number of offsets.
- *MixColumns()*: a mixing operation that operates on the columns of the state and combines the four bytes in each column using a linear transformation.

- *AddRoundKey()*: each byte of the state is combined with the round key; each round key is derived from the cipher key using a key schedule algorithm.

Algorithm 1 AESCipher (byte in [$4 \cdot Nb$], byte out [$4 \cdot Nb$], word w [$Nb \cdot (Nr+1)$])

```

1: byte state[4, Nb]
2: state = in
3: AddRoundKey(state, w[0, Nb-1])
4: FOR round = 1 to (Nr-1)
5:   SubBytes(state)
6:   ShiftRows(state)
7:   MixColumns(state)
8:   AddRoundKey(state, w[round * Nb, (round+1) * Nb-1])
9: End FOR
10: SubBytes(state)
11: ShiftRows(state)
12: AddRoundKey(state, w[Nr * Nb, (Nr + 1) * Nb - 1])
13: out = state

```

3.1.1 Function SubBytes

The *SubBytes()* is an invertible non-linear byte substitution operating on the *state* using a substitution table (S-box) which is constructed by composing two transformations:

- Take the multiplicative inverse in the finite field $GF(2^8)$ and the element $\{00\}$ is mapped to itself.
- Apply the following affine transformation over $GF(2)$:

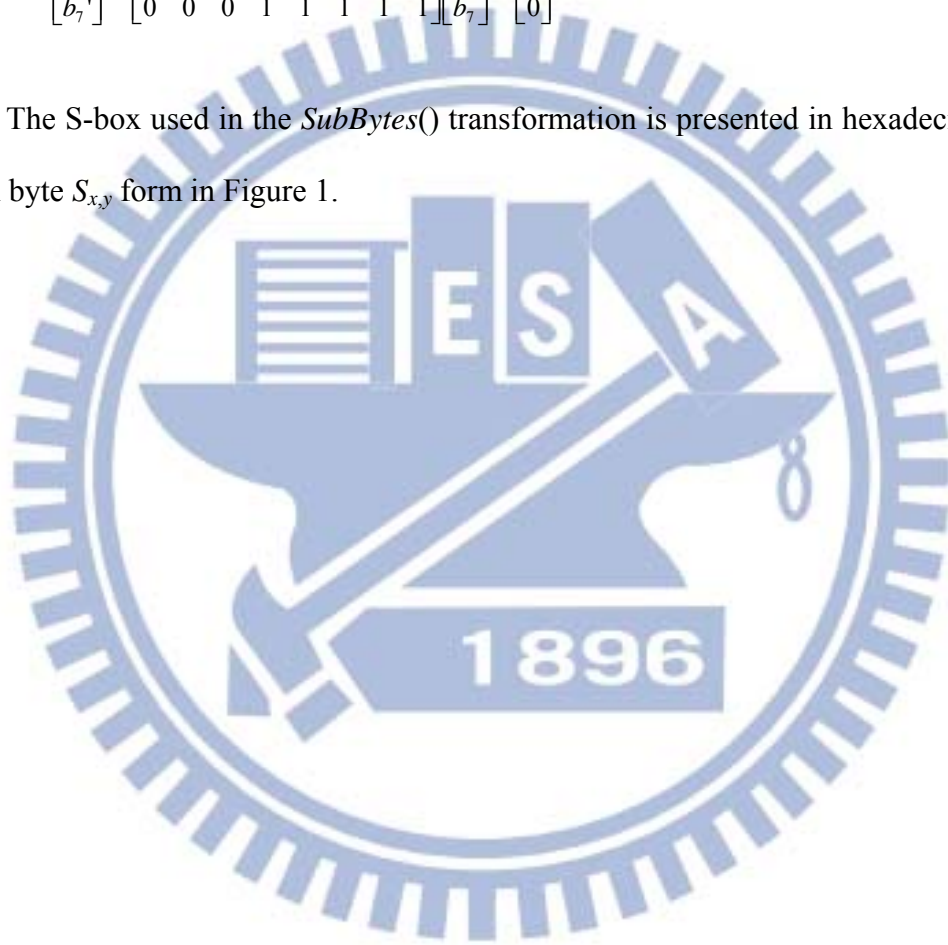
$$b'_i = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i \quad \text{Eq 1}$$

for $0 \leq i < 8$, where b_i is the i^{th} bit of the byte, and c_i is the i^{th} bit of a byte c with

the value {63} or {01100011}. In matrix form, the affine transformation element of the S-box can be expressed as:

$$\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \quad \text{Eq 2}$$

The S-box used in the *SubBytes()* transformation is presented in hexadecimal for each byte $S_{x,y}$ form in Figure 1.



		Y															
X		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df

Figure 1 S-box of AES

3.1.2 Function ShiftRows

In the *ShiftRows()*, the bytes in the last three rows of the *state* are left-rotated over different numbers of bytes. *ShiftRows()* is formed as:

$$S'_{r,c} = S_{r,(c+LRotate(r,Nb)) \bmod Nb} \text{ for } 0 < r < 4 \text{ and } 0 \leq c < Nb \quad \text{Eq 3}$$

where the rotation left $LRotate(0, 4) = 0$, $LRotate(1, 4) = 1$, $LRotate(2, 4) = 2$, $LRotate(3, 4) = 3$.

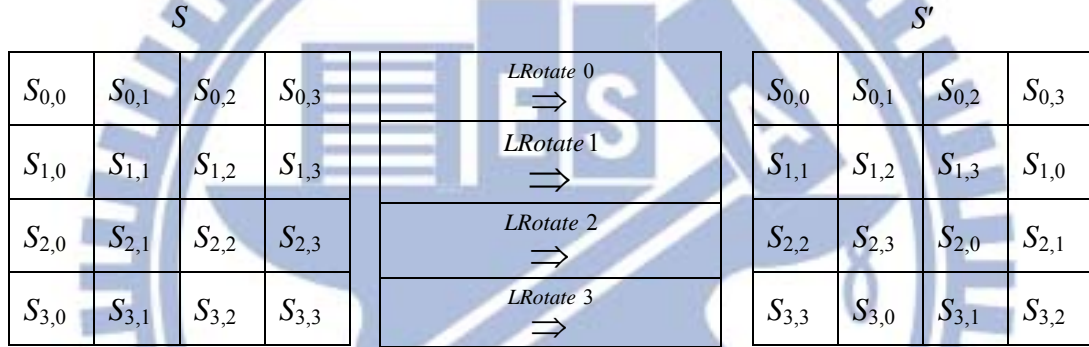


Figure 2 *ShiftRows()* in AES

3.1.3 Function MixColumns

The *MixColumns()* operates on the *state* column-by-column. The columns are considered as polynomials over $GF(2^8)$ and multiplied modulo $x^4 + 1$ with a fixed polynomial $a(x)$:

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \quad \text{Eq 4}$$

MixColumns() can be formed as a matrix multiplication $s'(x) = a(x) \otimes s(x)$:

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} \text{ for } 0 \leq c \leq Nb \quad \text{Eq 5}$$

3.1.4 Function AddRoundKey

A round Key is added to the *state* by a simple bitwise XOR operation in the *AddRoundKey()*. Each round key consists of *Nb* words which are each added into the columns of the *state*, such that

$$[S'_{0,c}, S'_{1,c}, S'_{2,c}, S'_{3,c}] = [S_{0,c}, S_{1,c}, S_{2,c}, S_{3,c}] \oplus [w_{round * Nb + c}] \text{ for } 0 \leq c \leq Nb \quad \text{Eq 6}$$

where $[w_i]$ are the key schedule words and $0 \leq round \leq Nr$.

3.1.5 Function Key Expansion

The key expansion generates a total of *Nb* (*Nr* + 1) words and consists of a linear array of 4-byte words, denoted $[w_i]$, $0 \leq i < Nb(Nr + 1)$.

Algorithm 2 KeyExpansion(byte key[4 * *Nk*]), word $w[Nb * (Nr + 1)]$, *Nk*)

```
1: word temp
2: i = 0
3: WHILE (i < Nk)
4:    $w[i] = \text{word}(\text{key}[4 * i], \text{key}[4 * i + 1], \text{key}[4 * i + 2], \text{key}[4 * i + 3])$ 
5:   i = i + 1
6: END WHILE
7: i = Nk
8: WHILE (i < Nb * (Nr + 1))
9:   temp =  $w[i - 1]$ 
10:  IF (i mod Nk = 0)
11:    temp =  $\text{SubWord}(\text{RotWord}(\text{temp})) \text{ xor } Rcon[i / Nk]$ 
12:  ELSE-IF (Nk > 6 and i mod Nk = 4)
13:    temp =  $\text{SubWord}(\text{temp})$ 
14:  END IF
15:   $w[i] = w[i - Nk] \text{ xor } \text{temp}$ 
16:  i = i + 1
17: END WHILE
```

SubWord() applies the S-box to each of the four-byte input word to produce an output word. The function *RotWord()* takes a word $[a_0, a_1, a_2, a_3]$ as input to perform a rotation left as the word $[a_1, a_2, a_3, a_0]$. *Rcon* $[i]$ is a round constant word array which contains the values $[x^{i-1}, \{00\}, \{00\}, \{00\}]$, where x is denoted as $\{02\}$ in the field $GF(2^8)$.

3.1.6 Inverse Cipher

The AES can be inverted by the implementation of *InvShiftRows()*, *InvSubBytes()*, *InvMixColumns()*, and *AddRoundKey()* on the *state* in reverse order.

Algorithm 3 InvAESCipher (byte in $[4 * Nb]$, byte out $[4 * Nb]$, word $w[Nb * (Nr + 1)]$)

```

1: byte state[4, Nb]
2: state = in
3: AddRoundKey(state, w[Nr * Nb, (Nr + 1) * Nb - 1])
4: FOR round = Nr - 1 downto 1
5:   InverShiftRows(state)
6:   InverSubByte(state)
7:   AddRoundKey(state, w[round * Nb, (round + 1) * Nb - 1])
8:   InverMixColumns(state)
9: END FOR
10: InverShiftRows(state)
11: InverSubByte(state)
12: AddRoundKey(state, w[round * Nb, (round + 1) * Nb - 1])
13: out = state

```

3.1.7 Function *InvShiftRows*

InvShiftRows() is the inverse of the *ShiftRows()* transformation.

$$S'_{r,(c+RRotate(r,Nb)\bmod Nb)} = S_{r,c} \text{ for } 0 < r < 4 \text{ and } 0 \leq c < Nb \quad \text{Eq 7}$$

where the rotation left $RRotate(0, 4) = 0$, $RRotate(1, 4) = 1$, $RRotate(2, 4) = 2$, $RRotate(3, 4) = 3$.



Figure 3 *InvShiftRows()* in AES

3.1.8 Function *InvShiftRows*

InvSubBytes() is the inverse of the *SubBytes()*. The inverse S-box used in the *InvSubBytes()* is:

		Y															
X		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	af	93	c9	9c	ef
	e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61

Figure 4 Inverse S-box of AES.

3.1.9 Function InvMixColumns

InvMixColumns() is the inverse of the *MixColumns()* transformation.

InvMixColumns() can be formed as a matrix multiplication $s' = a^{-1}(x) \otimes s(x)$,

where

$$a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\} \tag{Eq 8}$$

Such that

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} \text{ for } 0 \leq c \leq Nb \quad \text{Eq 9}$$

3.2 Related Works on SHA Family

Cryptographic hash functions play an important role in modern cryptography. They are widely used in a variety of applications such as password protection, secure protocols, and digital signatures. The hash function uses a string of arbitrary length as its input, and creates a fixed-length string as its output. A hash value is often called a data *fingerprint* or *message digest*.

Definition 1 [60]: (Collision-free hash function) A *collision-free hash function* H uses a message M of arbitrary length as its input, and produces a fixed-length message digest when it satisfies the following conditions:

- The description of $H(M)$ is publicly known and it is easy to implement.
- Pre-image resistant: Given message digest y , it is difficult to find a message M such that $H(M) = y$.
- Second pre-image resistant: Given M and its image $H(M)$, it is difficult to find another M' such that $H(M) = H(M')$.
- (Strong) Collision Resistance: It is difficult to find two distinct messages M and M' such that $H(M) = H(M')$.

The Secure Hash Algorithm (SHA) is a series of cryptographic hash functions published by the US National Institute of Standards and Technology (NIST). NIST proposed the SHA-0 as a Federal Information Processing Standard Publication (FIPS PUB) 180 in 1993 [61]. In 1995, NIST announced a revised version, the SHA-1, in FIPS PUB 180-1 [62] as a standard to replace the SHA-0. Since 2002, the NIST

published SHA-2 as FIPS PUB 180-2 [63], which consisted of four algorithms: SHA-1, SHA-256, SHA-384, and SHA-512 and then added SHA-224, SHA-512/224 and SHA-512/256 into FIPS PUB 180-3 [64] in 2008 and into 180-4 [21] in 2012. Table 1 lists the characteristics of the seven SHA algorithms.

3.2.1 Overview of SHA-1, SHA-224 and SHA-256 Algorithms

SHA-1, SHA-224 and SHA-256 [21] take a message M with a length of l bits, where $0 \leq l < 2^{64}$, as the input, and outputs 160-bit, 224-bit, and 256-bit hash values. The hash function parses the padded message into 512-bit blocks and each block passes an 80-round and 64-round compression functions.

SHA-1 processing involves the following 4 steps:

Step 1: Padding message: pad the input message making it a multiple of 512 bits.

Step 2: Parsing the padded message: parse the padded message into N 512-bit blocks, $M^{(1)}, M^{(2)}, \dots, M^{(N)}$. Each block $M^{(i)}$ is divided into sixteen 32-bit words, $M_0^{(i)}, M_1^{(i)}, \dots, M_{15}^{(i)}$.

Step 3: Computing hash values for each message block $M^{(i)}$.

- The message schedule, $\{W_t\}$:

$$W_t = \begin{cases} M_t^{(i)} & , 0 \leq t \leq 15 \\ ROTL^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) & , 16 \leq t \leq 79 \end{cases} \quad \text{Eq. 10}$$

where $ROTL^i(x)$ indicates left rotation operation by i bits.

- Message expansions are performed for 80 rounds. Algorithm 4 defines these steps in detail.
 - Table 3 summarizes the Boolean function f_t that appeared in the SHA-1 step function.

Step 4: Resulting message digest of the message, M , is

$$H_0^{(N)} \parallel H_1^{(N)} \parallel H_2^{(N)} \parallel H_3^{(N)} \parallel H_4^{(N)}$$

Eq 11

Algorithm 4 SHA-1 step function

- 1: $a = H_0^{(i-1)}, b = H_1^{(i-1)}, c = H_2^{(i-1)}, d = H_3^{(i-1)}, e = H_4^{(i-1)}$
 - 2: **FOR** $t = 1$ to 80
 - 3: $e_t = d_{t-1}$
 - 4: $d_t = c_{t-1}$
 - 5: $c_t = ROTL^{30}(b_{t-1})$
 - 6: $b_t = a_{t-1}$
 - 7: $a_t = ROTL^5(a_{t-1}) + f_i(b_{t-1}, c_{t-1}, d_{t-1}) + e_{t-1} + K_t + W_{t-1}$
 - 8: **End FOR**
 - 9: $H_0^{(i)} = a + H_0^{(i-1)}, H_1^{(i)} = b + H_1^{(i-1)}, H_2^{(i)} = c + H_2^{(i-1)}, H_3^{(i)} = d + H_3^{(i-1)}, H_4^{(i)} = e + H_4^{(i-1)}$
-

Table 3 Boolean function and constants used in SHA-1

Round t	Boolean function $f_i(x, y, z)$	K_t
$01 \leq t \leq 20$	$(x \wedge y) \vee (\neg x \wedge z)$	5a827999
$21 \leq t \leq 40$	$x \oplus y \oplus z$	6ed9eba1
$41 \leq t \leq 60$	$(x \wedge y) \vee (x \wedge z) \vee (y \wedge z)$	8f1bbcdc
$61 \leq t \leq 80$	$x \oplus y \oplus z$	ca62c1d6

Table 4 The initial hash value, $H^{(0)}$ in SHA-1

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

SHA-224 and SHA-256 take a message M with a length of l bits, where $0 \leq l < 2^{64}$, as the input, and output 224-bit and 256-bit hash value. The hash function parses the padded message into 512-bit blocks and each block passes a 64-round compression functions.

The SHA-224 and SHA-256 contain steps that are similar to SHA-1, except that it sets different initial values and constants, and uses different functions. The following is a description of the message block processing step.

Step 3: Message scheduling for each message block $M^{(i)}$.

- The message schedule, $\{W_t\}$:

$$W_t = \begin{cases} M_t^{(i)}, & 0 \leq t \leq 15 \\ \sigma_0^{\{256\}}(W_{t-2}) + W_{t-7} + \sigma_1^{\{256\}}(W_{t-15}) + W_{t-16}, & 16 \leq t \leq 63 \end{cases} \quad \text{Eq. 12}$$

$$\begin{aligned} \sigma_0^{\{256\}}(x) &= ROTL^7(x) \oplus ROTL^{18}(x) \oplus SHR^3(x) \\ \sigma_1^{\{256\}}(x) &= ROTL^{17}(x) \oplus ROTL^{19}(x) \oplus SHR^{10}(x) \end{aligned} \quad \text{Eq. 13}$$

where $SHR^{\{i\}}(x)$ indicates right shift operation by i bits.

Message expansions are performed for 64 rounds. Algorithm 5 defines these steps in detail.

- Table 5 summarizes the Boolean function f_i used in each round.

Step 4: Resulting final message digests

- The 224-bit message digest of the message, M , is

$$H_0^{(N)} \parallel H_1^{(N)} \parallel H_2^{(N)} \parallel H_3^{(N)} \parallel H_4^{(N)} \parallel H_5^{(N)} \parallel H_6^{(N)} \quad \text{Eq 14}$$

- The 256-bit message digest of the message, M , is

$$H_0^{(N)} \parallel H_1^{(N)} \parallel H_2^{(N)} \parallel H_3^{(N)} \parallel H_4^{(N)} \parallel H_5^{(N)} \parallel H_6^{(N)} \parallel H_7^{(N)} \quad \text{Eq 15}$$

Algorithm 5 SHA-224 and SHA-256 step function

1: $a = H_0^{(i-1)}$, $b = H_1^{(i-1)}$, $c = H_2^{(i-1)}$, $d = H_3^{(i-1)}$, $e = H_4^{(i-1)}$, $f = H_5^{(i-1)}$, $g = H_6^{(i-1)}$,
 $h = H_7^{(i-1)}$

1: **FOR** $t = 1$ to 64

2: $T_1 = h_{t-1} + f_1(e_{t-1}) + f_3(e_{t-1}, f_{t-1}, g_{t-1}) + K_t + W_{t-1}$

3: $T_2 = f_2(a_{t-1}) + f_4(a_{t-1}, b_{t-1}, c_{t-1})$

4: $h_t = g_{t-1}$

5: $g_t = f_{t-1}$

6: $f_t = e_{t-1}$

7: $e_t = d_t + T_1$

8: $d_t = c_{t-1}$

9: $d_t = c_{t-1}$

10: $c_t = b_{t-1}$

11: $b_t = a_{t-1}$

12: $a_t = T_1 + T_2$

13: **End FOR**

15: $H_0^{(i)} = a + H_0^{(i-1)}$, $H_1^{(i)} = b + H_1^{(i-1)}$, $H_2^{(i)} = c + H_2^{(i-1)}$, $H_3^{(i)} = d + H_3^{(i-1)}$,
 $H_4^{(i)} = e + H_4^{(i-1)}$, $H_5^{(i)} = f + H_5^{(i-1)}$, $H_6^{(i)} = g + H_6^{(i-1)}$, $H_7^{(i)} = h + H_7^{(i-1)}$

Table 5 Boolean function used in SHA-224 and SHA-256

Boolean function f_t
$f_1(x) = ROTL^{(2)}(x) \oplus ROTL^{(13)}(x) \oplus ROTL^{(22)}(x)$
$f_2(x) = ROTL^{(6)}(x) \oplus ROTL^{(11)}(x) \oplus ROTL^{(25)}(x)$
$f_3(x) = (x \wedge y) \oplus (\neg x \wedge z)$
$f_4(x) = (x \wedge y) \oplus (x \wedge z) \vee (y \wedge z)$

Table 6 Constants in SHA-224 and SHA-256 (From left to right, up to down)

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

Table 7 The initial hash value, $H^{(0)}$ in SHA-224 and SHA-256

$H^{(0)}$	SHA-224	SHA-256
$H_0^{(0)}$	c1059ed8	6a09e667
$H_1^{(0)}$	367cd507	bb67ae85
$H_2^{(0)}$	3070dd17	3c6ef372
$H_3^{(0)}$	f70e5939	a54ff53a
$H_4^{(0)}$	ffc00b31	510e527f
$H_5^{(0)}$	68581511	9b05688c
$H_6^{(0)}$	64f98fa7	1f83d9ab
$H_7^{(0)}$	befa4fa4	5be0cd19

3.2.2 Overview of SHA-384 and SHA-512 Algorithms

SHA-384 and SHA-512 take a message M with a length of l bits, where $0 \leq l < 2^{512}$, as the input, and outputs 384-bit, and 512-bit hash values. The hash function

parses the padded message into 1024-bit blocks and each block passes an 80-round and 80-round compression functions.

SHA-384 and SHA-512 processing involve the following 4 steps:

Step 1: Padding message: pad the input message making it a multiple of 1024 bits.

Step 2: Parsing the padded message: parse the padded message into N 1024-bit blocks, $M^{(1)}, M^{(2)}, \dots, M^{(N)}$. Each block $M^{(i)}$ is divided into sixteen 64-bit words, $M_0^{(i)}, M_1^{(i)}, \dots, M_{15}^{(i)}$.

Step 3: Computing hash values for each message block $M^{(i)}$.

- The message schedule, $\{W_t\}$:

$$W_t = \begin{cases} M_t^{(i)} & , 0 \leq t \leq 15 \\ \sigma_0^{\{512\}}(W_{t-2}) + W_{t-7} + \sigma_1^{\{512\}}(W_{t-15}) + W_{t-16} & , 16 \leq t \leq 63 \end{cases} \quad \text{Eq 16}$$

$$\begin{aligned} \sigma_0^{\{512\}}(x) &= ROTL^1(x) \oplus ROTL^8(x) \oplus SHR^7(x) \\ \sigma_1^{\{512\}}(x) &= ROTL^{19}(x) \oplus ROTL^{61}(x) \oplus SHR^6(x) \end{aligned} \quad \text{Eq 17}$$

- Message expansions are performed for 80 rounds.
 - defines these steps in detail. Table 8 summarizes the Boolean function f_t used in each round.

Step 4: Resulting final message digests

- The 384-bit message digest of the message, M , is

$$H_0^{(N)} \parallel H_1^{(N)} \parallel H_2^{(N)} \parallel H_3^{(N)} \parallel H_4^{(N)} \parallel H_5^{(N)} \quad \text{Eq 18}$$

- The 512-bit message digest of the message, M , is

$$H_0^{(N)} \parallel H_1^{(N)} \parallel H_2^{(N)} \parallel H_3^{(N)} \parallel H_4^{(N)} \parallel H_5^{(N)} \parallel H_6^{(N)} \parallel H_7^{(N)} \quad \text{Eq 19}$$

Algorithm 6 SHA-385 and SHA-512 step function

1: $a = H_0^{(i-1)}$, $b = H_1^{(i-1)}$, $c = H_2^{(i-1)}$, $d = H_3^{(i-1)}$, $e = H_4^{(i-1)}$, $f = H_5^{(i-1)}$, $g = H_6^{(i-1)}$,
 $h = H_7^{(i-1)}$

1: **FOR** $t = 1$ to 80

2: $T_1 = h_{t-1} + f_1(e_{t-1}) + f_3(e_{t-1}, f_{t-1}, g_{t-1}) + K_t + W_{t-1}$

3: $T_2 = f_2(a_{t-1}) + f_4(a_{t-1}, b_{t-1}, c_{t-1})$

4: $h_t = g_{t-1}$

5: $g_t = f_{t-1}$

6: $f_t = e_{t-1}$

7: $e_t = d_t + T_1$

8: $d_t = c_{t-1}$

9: $d_t = c_{t-1}$

10: $c_t = b_{t-1}$

11: $b_t = a_{t-1}$

12: $a_t = T_1 + T_2$

13: **End FOR**

15: $H_0^{(i)} = a + H_0^{(i-1)}$, $H_1^{(i)} = b + H_1^{(i-1)}$, $H_2^{(i)} = c + H_2^{(i-1)}$, $H_3^{(i)} = d + H_3^{(i-1)}$,
 $H_4^{(i)} = e + H_4^{(i-1)}$, $H_5^{(i)} = f + H_5^{(i-1)}$, $H_6^{(i)} = g + H_6^{(i-1)}$, $H_7^{(i)} = h + H_7^{(i-1)}$

Table 8 Boolean function used in SHA-384 and SHA-512

Boolean function f_t
$f_1(x) = ROTL^{(28)}(x) \oplus ROTL^{(34)}(x) \oplus ROTL^{(39)}(x)$
$f_2(x) = ROTL^{(14)}(x) \oplus ROTL^{(18)}(x) \oplus ROTL^{(41)}(x)$
$f_3(x) = (x \wedge y) \oplus (\neg x \wedge z)$
$f_4(x) = (x \wedge y) \oplus (x \wedge z) \vee (y \wedge z)$

Table 9 Constants in SHA-385 and SHA-512 (From left to right, up to down)

428a2f98d728ae22	7137449123ef65cd	b5c0fbcfec4d3b2f	e9b5dba58189dbbc
3956c25bf348b538	59f111f1b605d019	923f82a4af194f9b	ab1c5ed5da6d8118
d807aa98a3030242	12835b0145706fbe	243185be4ee4b28c	550c7dc3d5ffb4e2
72be5d74f27b896f	80deb1fe3b1696b1	9bdc06a725c71235	c19bf174cf692694
e49b69c19ef14ad2	efbe4786384f25e3	0fc19dc68b8cd5b5	240ca1cc77ac9c65
2de92c6f592b0275	4a7484aa6ea6e483	5cb0a9dcdbd41fbd4	76f988da831153b5
983e5152ee66dfab	a831c66d2db43210	b00327c898fb213f	bf597fc7beef0ee4
c6e00bf33da88fc2	d5a79147930aa725	06ca6351e003826f	142929670a0e6e70
27b70a8546d22ffc	2e1b21385c26c926	4d2c6dfc5ac42aed	53380d139d95b3df
650a73548baf63de	766a0abb3c77b2a8	81c2c92e47edae6	92722c851482353b
a2bfe8a14cf10364	a81a664bbc423001	c24b8b70d0f89791	c76c51a30654be30
d192e819d6ef5218	d69906245565a910	f40e35855771202a	106aa07032bbd1b8
19a4c116b8d2d0c8	1e376c085141ab53	2748774cdf8eeb99	34b0bcb5e19b48a8
391c0cb3c5c95a63	4ed8aa4ae3418acb	5b9cca4f7763e373	682e6ff3d6b2b8a3
748f82ee5defb2fc	78a5636f43172f60	84c87814a1f0ab72	8cc702081a6439ec
90befffa23631e28	a4506cebbe82bde9	bef9a3f7b2c67915	c67178f2e372532b
ca273ecee26619c	d186b8c721c0c207	eada7dd6cde0eb1e	f57d4f7fee6ed178
06f067aa72176fba	0a637dc5a2c898a6	113f9804bef90dae	1b710b35131c471b
28db77f523047d84	32caab7b40c72493	3c9ebe0a15c9bebc	431d67c49c100d4c
4cc5d4becb3e42b6	597f299cfc657e2a	5fcb6fab3ad6faec	6c44198c4a475817

Table 10 The initial hash value, $H^{(0)}$ in SHA-384 and SHA-512

$H^{(0)}$	SHA-385	SHA-512
$H_0^{(0)}$	cbbb9d5dc1059ed8	6a09e667f3bcc908
$H_1^{(0)}$	629a292a367cd507	bb67ae8584caa73b
$H_2^{(0)}$	9159015a3070dd17	3c6ef372fe94f82b
$H_3^{(0)}$	152fec8f70e5939	a54ff53a5f1d36f1
$H_4^{(0)}$	67332667ffc00b31	510e527fade682d1
$H_5^{(0)}$	8eb44a8768581511	9b05688c2b3e6c1f
$H_6^{(0)}$	db0c2e0d64f98fa7	1f83d9abfb41bd6b
$H_7^{(0)}$	47b5481dbefa4fa4	5be0cd19137e2179

3.3 Genetic Algorithm

The genetic algorithm is the most popular type of evolutionary algorithm that use techniques inspired by evolutionary biology. As stated by John H. Holland in 1975, “The genetic algorithm has a wide scope of applications, including economics, engineering, machine learning, genome biology, game theory, neural networks, and etc. [65]. A genetic algorithm provides a highly efficient method for ensuring convergence to near-optimal or optimal solutions.

Figure 5 shows the steps of the genetic algorithm, which are described as follows:

- (1) Initialization of population.

- (2) Choice of a fitness function and evaluation of the fitness value of each individual in the population.
- (3) Selection of better ranked part to be reproduced.
- (4) Breeding new generation's population by crossover and mutation.
- (5) Replacement of the worst ranked part of the population with the new generation's population.
- (6) Repeating this generational process until the termination condition has been reached.

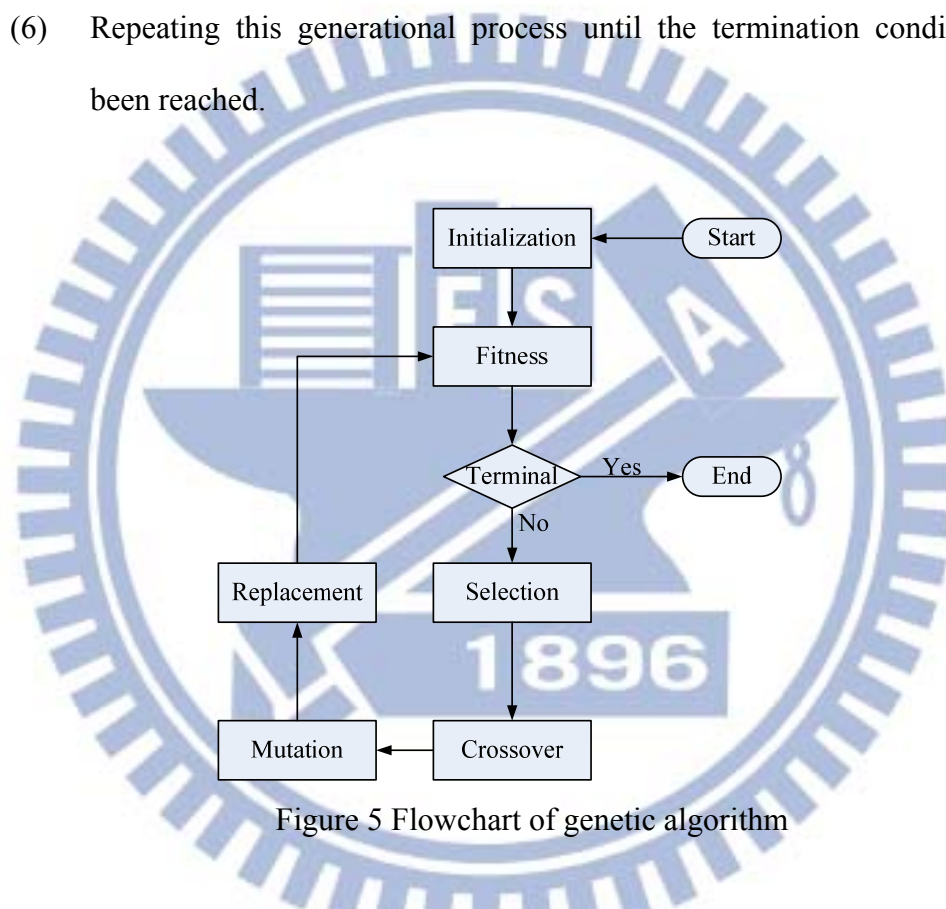


Figure 5 Flowchart of genetic algorithm

The Genetic Algorithm Utility Library (GAUL) developed by AI Foundry [66] is a flexible programming library designed to aid in the development of applications that use genetic or evolutionary algorithms. It provides data structures and functions for handling and manipulating the data required for serial and parallel evolutionary algorithms.

GAUL is an open-source programming library, which was released under the GNU General Public License. It is designed to assist in the development of code that

requires evolutionary algorithms.

3.4 Secret Sharing Scheme

Let t, n be positive integers, $t \leq n$. Shamir proposed a (t, n) -threshold scheme in 1979 [53]. His scheme is a method of sharing a key K among a set of n participants in such a way that any t participants can compute the value of key K , but no group of $(t - 1)$ participants can do so.

3.4.1 The Shamir (t, n) -Threshold Scheme in \mathbb{Z}_p

D (the dealer) chooses n distinct, nonzero elements of \mathbb{Z}_p , denoted $x_i, 1 \leq i \leq n$, where $p > n$ is a large prime. D gives the values x_i to participant P_i , and each value x_i is public.

3.4.2 Share Distribution

1. Suppose D wants to share a key $K \in \mathbb{Z}_p$. D secretly chooses (independently and randomly) $(t - 1)$ elements of $\mathbb{Z}_p, a_1, \dots, a_{t-1}$.
2. For $1 \leq i \leq n$, D computes $y_i = a(x_i)$, where

$$a(x) = K + \sum_{j=1}^{t-1} a_j x^j \pmod{p} \quad \text{Eq. 20}$$

Thus

$$y_i = a(x_i) = K + \sum_{j=1}^{t-1} a_j x_i^j \pmod{p} \quad \text{Eq. 21}$$

3. For $1 \leq i \leq n$, D gives the share y_i to P_i .

3.4.3 Proactive Security

It is difficult to compromise the secret key K under (t, n) -threshold scheme unless the adversary collects at least t shares. In practice, since each share exists in a machine, the risk of the secret key being compromised depends on the security of machine. For security concerns, it is necessary to update each share for a period of time. A proactive threshold scheme allows users to refresh shares without disclosing the secret key.

1. Let

$$y_i = a(x_i) = K + \sum_{j=1}^{t-1} a_j x_i^j \text{ mod } p \quad \text{Eq. 21}$$

be the original share of key K for P_i .

2. The dealer D then computes

$$y'_i = a(x'_i) = \sum_{j=1}^{t-1} a_j x'_i{}^j \text{ mod } p \quad \text{Eq. 22}$$

3. For $1 \leq i \leq n$, D gives the share y'_i to P_i .
4. For $1 \leq i \leq n$, P_i computes $(y_i + y'_i)$ as a new share.

3.5 Autonomous Key Management (AKM)

Autonomous key management (AKM) for a mobile ad hoc network (MANET) with a large number of nodes is based on a hierarchical structure to provide flexibility and adaptivity. Every leaf node in the logical tree structure is a real ad hoc device, and the other nodes are virtual nodes. The root node holds the global secret key, and AKM distributes key shares to its children recursively from the root down to the leaves using Shamir's secret sharing scheme.

Every node except the AKM root node must store its own public key pair and its

parent node secret share. The secret share each virtual branch node holds is as the secret key, and the public key can be generated using any asymmetric cryptographic scheme, such as RSA. Additionally, every real node has its PKI key pair before joining AKM.

A tree with node A as its root is called region A. AKM includes seven node-based/region-based operations from node joining, region partitioning, to node leaving. AKM runs dynamically with continuous node joining/leaves. Figure 6 is an example of AKM.

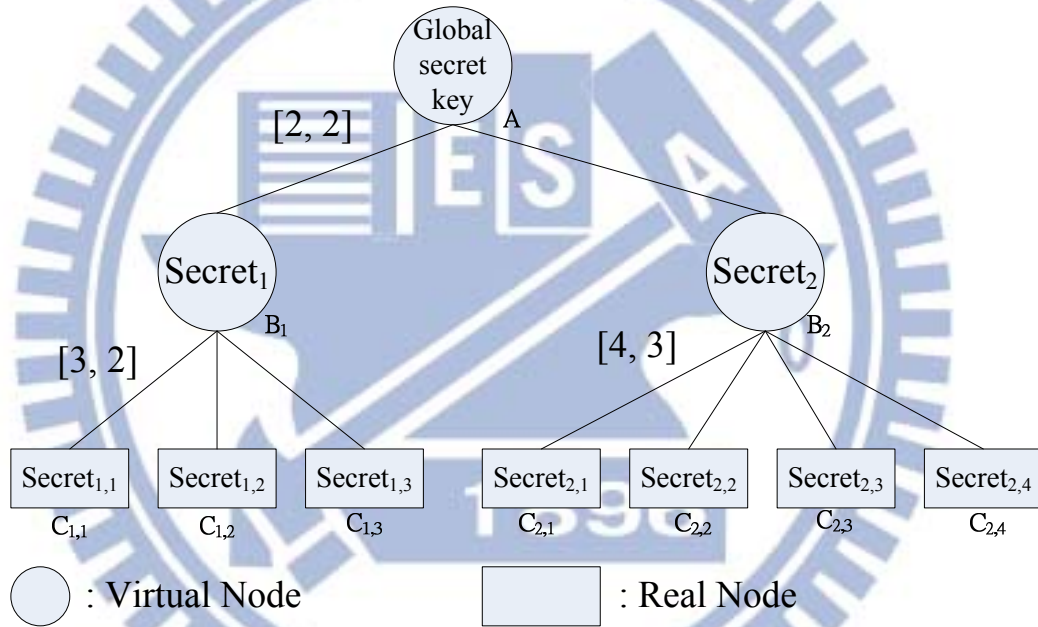


Figure 6 An example of AKM

4. Our Proposed Schemes

This dissertation proposed many schemes for content protection especially on symmetric encryptions algorithm, one-way hash functions, and secure protocols on MANET. The front two parts improve the core cryptography components used in the rear part, which protects content from adversaries' attacks on transmission.

4.1 A Transpositional Advanced Encryption Standard (AES) Resists 3-round Square Attack [67]

This work replaces some functions in the round transformation of AES and takes the bit as the operation unit to avoid 3-round square attacks. Applying linear cryptanalysis and differential cryptanalysis to our proposed block cipher, the results show our proposed cipher can resist these attacks in five and four rounds, respectively.

The rest of this section is organized as follows: Section 3.1.1 describes some mathematic preliminaries and the design of the AES. Section 3.1.2 specifies the design of the proposed cipher. Section 3.1.3 discusses cryptanalysis results. Finally, section 3.1.4 summarizes the paper. To make the article more easily readable, a terminology table is listed below for the reader to consult.

4.1.1 Cipher Structure

As shown in Figure 7, the proposed cipher *AES_Plus* is an iterated block cipher that consists of an initial round key addition modulo 2; $Nr-1$ rounds that have the same transformations; where Nr is the total number of rounds, and a final round.

There are $(Nr-1)$ rounds and one final round that are distinct transformations and

take the previous state and the round key as inputs. We denote the total round keys as an array round key with Nr elements whose size is equal to the size of the state. $RoundKey[0]$ is used by the initial round key addition which will be described in next section. $RoundKey[i]$ is used for the i^{th} round where $1 \leq i \leq Nr$. $RoundKey[Nr]$ is used for the final round. For encryption, each round of the proposed cipher consists of four procedures:

SubByte() — a non-linear substitution where each byte is replaced by another byte according to a lookup table.

TransByte() — takes half the state as an 8×8 square matrix where each component is one bit, then interchanges the row and the column with the same indices such that $b_{i,j}$ becomes $b_{j,i}$.

SubBlkXor() — sub-block exclusive-or (XOR) transformation of Feistel structure that is used to perform bitwise exclusive-or operation on some other sub-block.

AddRoundKey() — each byte of the state is combined with the round key; each round key is derived from the cipher key using a key schedule algorithm.

In Algorithm *AES_Plus*, *KeyExpansion()* expands the cipher key to the total number of round keys denoted as *ExpandedKey*. This procedure can be excluded from the algorithm. *AddRoundKey()* denotes the initial key addition. The most important parts of *AES_Plus* are *Round_Plus* and *FinalRound_Plus*, which are the parts mainly improved in this research. These parts are described in the next section.

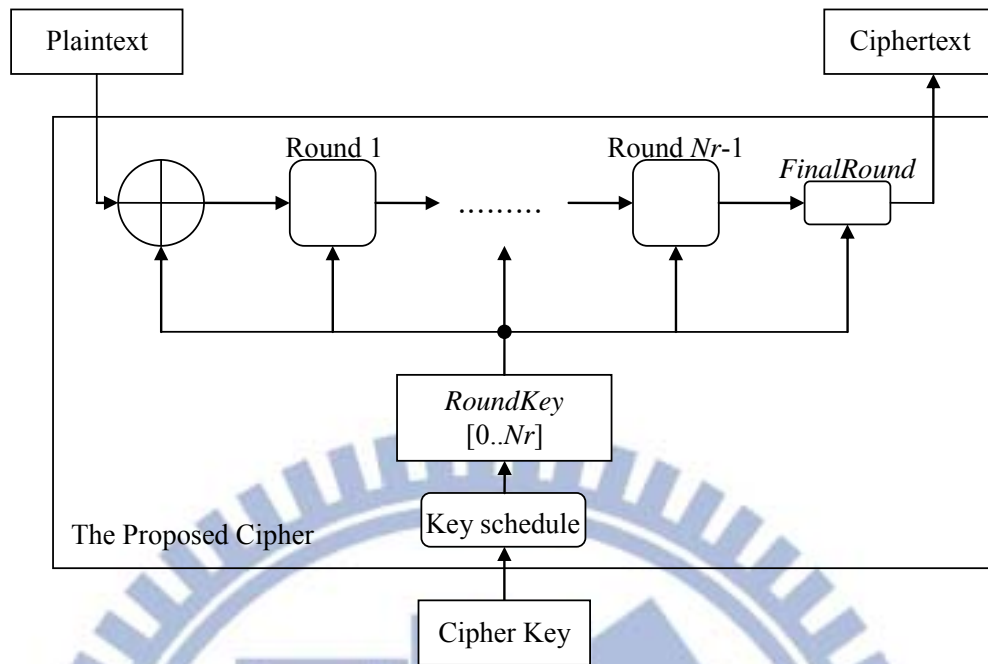


Figure 7 The structure overview of the proposed cipher where \oplus denotes the round key addition.

Algorithm 7 *AES_Plus* (State, Ciphertext)

// *AES_Plus* provides the encryption for the proposed //cipher.

- 1: *KeyExpansion*(CipherKey, ExpandedKey)
 - 2: *AddRoundKey*(State, RoundKey[0])
 - 3: **For**($i = 0; i < Nr; i++$)
 - 4: *Round_Plus*(State, RoundKey[i])
 - 5: *FinalRound_Plus*(State, RoundKey[Nr])
 - 6: **End For**
-

4.1.2 Our Proposed AES_Plus

The proposed algorithm *AES_Plus* improves AES in the area of round transformation. This section describes each of the *AES_Plus* procedures and other properties of *AES_Plus*.

4.1.2.1 The Round Transformation

There are four distinct procedures in the round transformation, *Round_Plus*: *ByteSub()*, *TransByte()*, *SubBlkXor()*, and *AddRoundKey()*. *Round_Plus* and *FinalRound_Plus* are illustrated in Figure 8 and Figure 9. The S-box is used to substitute the input byte. In the final phase, the i^{th} round key is added to the i^{th} state where $1 \leq i < Nr$ in *AddRoundKey()*. The final round, *FinalRound_Plus* has the same transformations, but *SubBlkXor()* is replaced by the Swap procedure. Algorithm 8 and Algorithm 9 shows that *Round_Plus* and *FinalRound_Plus* are composed of five procedures: *ByteSub()*, *TransByte()*, *SubBlkXor()*, *AddRoundKey()* and *Swap()*. These procedures will be described later.

Algorithm 8 *Round_Plus* (State, RoundKey[i])

- 1: *ByteSub*(State)
 - 2: *TransByte*(State)
 - 3: *SubBlkXor*(State)
 - 4: *AddRoundKey*(State, RoundKey[i])
-

Algorithm 9 *FinalRound_Plus* (State, ExpandedRoundKey[Nr])

- 1: *ByteSub*(State)
 - 2: *TransByte*(State)
 - 3: *Swap*(State)
 - 4: *AddRoundKey*(State, ExpandedRoundKey[Nr])
-

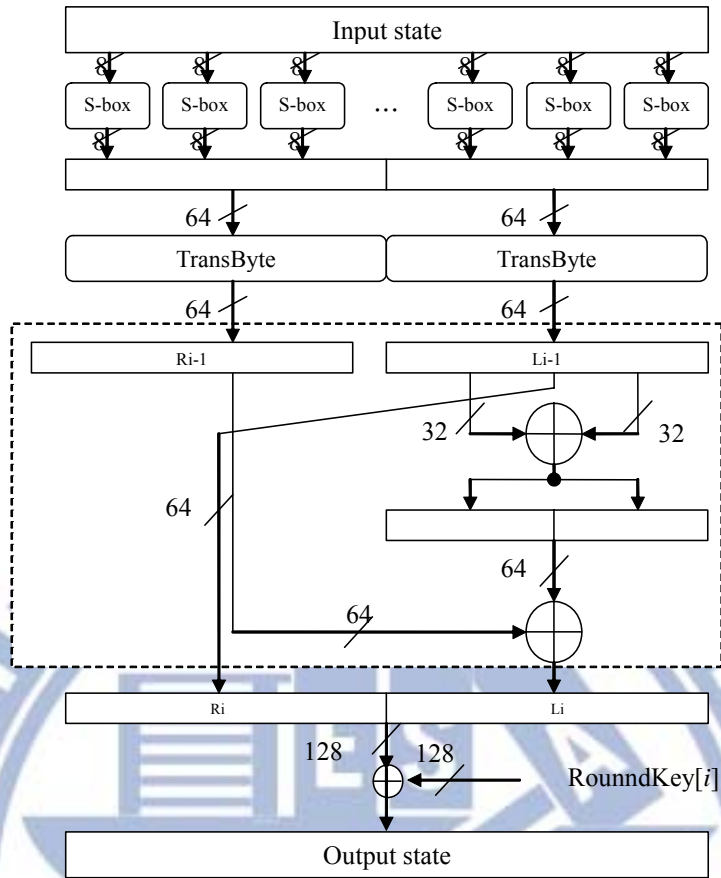


Figure 8 Round Plus of AES Plus

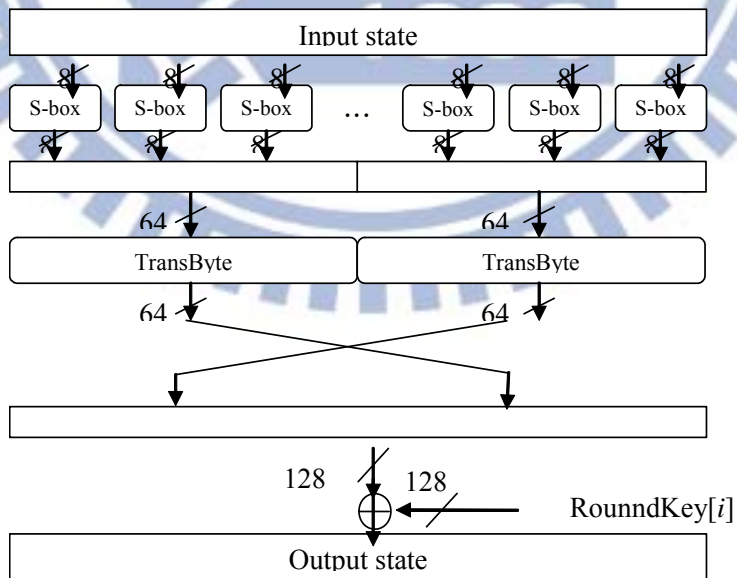


Figure 9 FinalRound Plus of AES Plus

4.1.2.1.1 ByteSub Procedure

This *ByteSub()* is the same as *SubBytes()* non-linear substitution procedure in AES [68] using the same S-box table. The procedure executes rapidly through table look-up implementation and provides strong enough secure complexity.

4.1.2.1.2 TransByte Procedure

The first eight bytes of the state can be taken as an 8×8 square matrix where each element is one byte. After a matrix transformation the new 8×8 matrix is now composed of 64 bytes. Figure 10 illustrates the matrix transposition, where $b_{i,j}$ is the i^{th} row and j^{th} column of the matrix. After *TransByte()*, $b_{i,j}$ interchanges its position with $b_{j,i}$. The first byte, $\{b_{0,0} b_{0,1} b_{0,2} b_{0,3} b_{0,4} b_{0,5} b_{0,6} b_{0,7}\}$, is replaced by $\{b_{0,0} b_{1,0} b_{2,0} b_{3,0} b_{4,0} b_{5,0} b_{6,0} b_{7,0}\}$. The inverse of the *TransByte()* operation is itself.

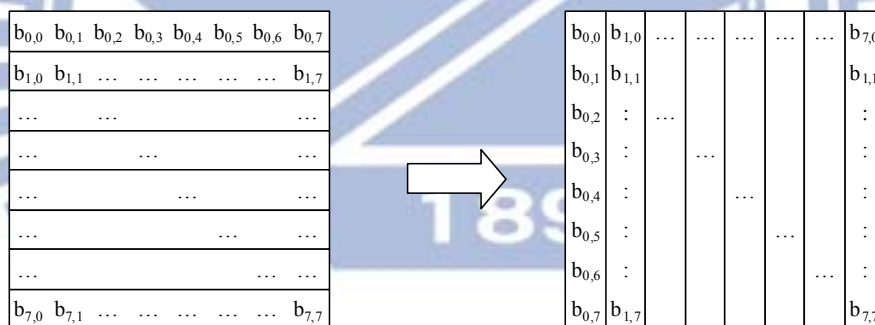


Figure 10 *TransByte*

4.1.2.1.3 Sub-Block XOR Procedure

The *SubBlkXor()* is a Feistel structure. It is fast for both encryption and decryption and is very easy to analyze. In this operation, it performs bitwise XOR in each round to distribute the effects on bits as much as possible. Figure 11 illustrates this procedure.

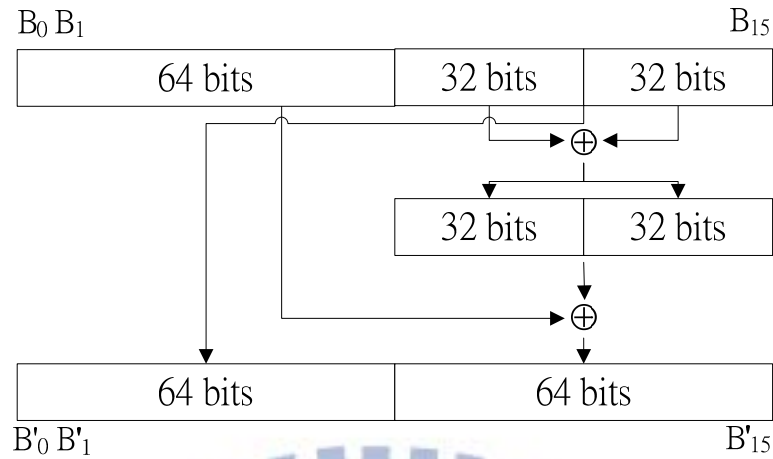


Figure 11 Sub-Block XOR

The first eight bytes (B'_0 to B'_7) are assigned from B_8 to B_{15} . For B'_8 to B'_{15} , B_8 to B_{11} are used to perform bitwise XOR operations with B_{12} to B_{15} . By repeating itself, the result is that 32 bits are expanded as two sets of 32 bits (64 bits). Finally, the 64 bits perform XOR again with B_0 to B_7 to be B'_8 to B'_{15} . The inverse of this operation is as follows (we denote B'_i as byte after *SubBlkXor()*):

$$B'_i = \begin{cases} B_{i+8}, & \text{if } 0 \leq i \leq 7 \\ B_{i+4} \oplus B_i \oplus B_{i \bmod 8}, & \text{if } 8 \leq i \leq 15 \end{cases} \quad \text{Eq 23}$$

4.1.2.1.4 Round Key Addition

AddRoundKey() performs the bitwise XOR operation with the round key the same as in AES. Each round key size is the same as the state size and is derived from the cipher key by the key scheduling algorithm. The inverse is the *AddRoundKey()* itself.

4.1.2.1.5 Swap

FinalRound_Plus is a special round in *AES_Plus*. It replaces *SubBlkXor()* with *Swap()*. It swaps the left and right 64-bit sub-blocks.

4.1.2.2 Number of Rounds

The number of rounds (Nr) depends on the result of cryptanalysis. We will show that six-round *AES_Plus* is strong enough to resist linear and differential attacks. Here, we choose $Nr = 10$ for the proposed cipher.

4.1.2.3 Deciphering

As shown in Figure 12, the decryption algorithm is the inverse of the encryption algorithm and uses the same key. Here, $ByteSub^{-1}()$ stands for the inverse of the $ByteSub()$ as in AES.

[Encryption]	[Decryption]
<i>AddRoundKey</i> (K_0)	<i>AddRoundKey</i> (K_{Nr})
<i>ByteSub</i>	<i>TransByte</i>
<i>TransByte</i>	<i>ByteSub</i> ⁻¹
<i>SubBlkXor</i>	<i>AddRoundKey</i> (K_{Nr-1})
<i>AddRoundKey</i> (K_1)	<i>SubBlkXor</i>
.....
<i>ByteSub</i>	<i>TransByte</i>
<i>TransByte</i>	<i>ByteSub</i> ⁻¹
<i>SubBlkXor</i>	<i>AddRoundKey</i> (K_1)
<i>AddRoundKey</i> (K_{Nr-1})	<i>SubBlkXor</i>
<i>ByteSub</i>	<i>TransByte</i>
<i>TransByte</i>	<i>ByteSub</i> ⁻¹
<i>Swap</i>	<i>Swap</i>
<i>AddRoundKey</i> (K_{Nr})	<i>AddRoundKey</i> (K_0)

Figure 12 Encryption and decryption of *AES_Plus*

4.2 Generalized Secure Hash Algorithm: SHA-X [69]

This section describes the processing of generalizing the Secure Hash Algorithm according to the SHA family algorithm. The process of generalization includes padding, parsing, setting the initial hash values, constants, Boolean expressions and functions, and message schedule; initializing the eight working variables and for-loop operation; and computing the i^{th} intermediate hash values. In the following section, we describe the processes of generalizing in detail.

4.2.1 Generalized Secure Hash Algorithm

4.2.1.1 The Length of One Word and the Number of Output Words

First, we define the length of one word as n such that $n = 32$ in SHA-224 and SHA-256, and $n = 64$ in SHA-384 and SHA-512.

Second, we should define the number of output words m . For example, the output length of SHA-256 is 256 bits, 8 words equally ($m = 8$, 256 bits = 8 word \times 32 bits/word). Similarly, $m = 6$ in SHA-384 (384 bits = 6 words \times 64 bits/word). On the basis of the SHA family, we define the value of m ($6 \leq m \leq 8$), and the length of one word/block n is multiple of 32. With the m and n , we can generalize the SHA family to SHA- mn .

In SHA- mn , where $m = \{6, 7, 8\}$, and $n = \{32, 64\}$, we find two additional formats, called SHA-192 ($m = 6$ and $n = 32$) and SHA-448 ($m = 7$ and $n = 64$). If SHA family includes SHA-192 and SHA-448, we call it *Complete SHA family*. The *Complete SHA family* is defined below.

Definition 2 *Complete SHA family* is defined:

$$\text{Complete-SHA} = \{\text{SHA-192}, \text{SHA-224}, \text{SHA-256}, \text{SHA-384}, \text{SHA-448}, \text{SHA-512}\}$$

Eq 24

Table 11 Values of m and n for SHA family

Property	SHA- nm					
	SHA-192	SHA-224	SHA-256	SHA-384	SHA-448	SHA-512
Word Size (n)	32			64		
# of Output Words (m)	6	7	8	6	7	8
Message Digest Size	192	224	256	384	448	512
Block Size	512			1024		
Security ¹	2^{96}	2^{112}	2^{128}	2^{192}	2^{224}	2^{256}

4.2.1.2 Padding the Message M

The section generalizes the padding step in SHA- mn . Assuming that M is l bits ($0 \leq l < 2^{2n}$), the padding process should satisfy the following two rules:

- If we have $l \leq 14n-1 \pmod{16n}$, we should pad “1||0*||(l)₂” up to the length of $\left\lceil \frac{l}{16n} \right\rceil \times 16n$. Notice that “1||0*” denotes that “1” is followed by zero “0” bit or more than one bits and the (l)₂ denotes the length of message in binary.
- If we have $l > 14n-1 \pmod{16n}$, we should pad “1||0*||(l)₂” up to the length of $\left(\left\lceil \frac{l}{16n} \right\rceil + 1 \right) \times 16n$. Notice that “1||0*” denotes that “1” is followed by zero “0” bit or more than one bits and the (l)₂ denotes the length of message in binary.

¹ The security complexity is under birthday attack.

Algorithm 10 Padding

1: **If** $l \leq 14n - 1 \pmod{16n}$

2: **Then** $M' = M \parallel 1 \parallel 0^* \parallel (I)_2$ such that $|M'| = \left\lceil \frac{1}{16n} \right\rceil \times 16n$

3: **Else** $M' = M \parallel 1 \parallel 0^* \parallel (I)_2$ such that $|M'| = \left(\left\lceil \frac{1}{16n} \right\rceil + 1 \right) \times 16n$

4: **End IF**

4.2.1.3 Parsing the Padded Message into Message Blocks

Based on the properties of SHA family, SHA- mn parses the padded message into N $16 \times n$ bits blocks denoted by $M^{(1)} \dots M^{(N)}$. For each $16 \times n$ -bit $M^{(i)}$, the M will be divided into sixteen n -bit sub-blocks denoted by $M_0^{(i)} \dots M_{15}^{(i)}$.

Algorithm 11 Parsing

1: parsing M' into $M^{(1)} \dots M^{(N)}$

2: **For** $i \leftarrow 1$ to N **Do**

3: $M^{(i)} = M_0^{(i)} \parallel M_1^{(i)} \parallel \dots \parallel M_{15}^{(i)}$, $|M^{(i)}| = 16n$

4: **End For**

4.2.1.4 Setting the Initial Hash Values

The initial hash values consist of eight n -bit words denoted by $H_0^{(0)} \dots H_7^{(0)}$. The following are the rules of setting initial hash value in each SHA family members.

- In SHA-256(or in SHA-512), each initial hash value is 32(or 64) bits which are the first 32(or 64) bits of the fractional parts of the square roots of the 1st eight prime numbers. The first eight prime numbers are 2, 3, 5, 7, 11, 13, 17 and 19.
- In SHA-224, each initial hash value is 32 bits which are the 33th ~ 64th bits of the fractional parts of the square roots of the 9th through 16th prime

numbers. The 9th through 16th prime numbers are 23, 29, 31, 37, 41, 43, 47 and 53.

- In SHA-384, each initial hash value is 64 bits which are the first 64 bits of the fractional parts of the square roots of the 9th through 16th prime numbers. The 9th through 16th prime numbers are 23, 29, 31, 37, 41, 43, 47 and 53.

Based on SHA family, the paper defines initial hash value for the additional SHA-192 and SHA-448.

- In SHA-192(or in SHA-448), each initial hash value is 32(or 64) bits, which are the first 32(or 64) bits of the fractional parts of the square roots of the 17th through 24th prime numbers. The 17th through 24th prime numbers are 59, 61, 67, 71, 73, 79, 83 and 89.

H(1) = 6 a 0 9 e 6 6 7 f 3 b c c 9 0 8	H(09) = c b b b 9 d 5 d c 1 0 5 9 e d 8
H(2) = b b 6 7 a e 8 5 8 4 c a a 7 3 b	H(10) = 6 2 9 a 2 9 2 a 3 6 7 c d 5 0 7
H(3) = 3 c 6 e f 3 7 2 f e 9 4 f 8 2 b	H(11) = 9 1 5 9 0 1 5 a 3 0 7 0 d d 1 7
H(4) = a 5 4 f f 5 3 a 5 f 1 d 3 6 f 1	H(12) = 1 5 2 f e c d 8 f 7 0 e 5 9 3 9
H(5) = 5 1 0 e 5 2 7 f a d e 6 8 2 d 1	H(13) = 6 7 3 3 2 6 6 7 f f c 0 0 b 3 1
H(6) = 9 b 0 5 6 8 8 c 2 b 3 e 6 c 1 f	H(14) = 8 e b 4 4 a 8 7 6 8 5 8 1 5 1 1
H(7) = 1 f 8 3 d 9 a b f b 4 1 b d 6 b	H(15) = d b 0 c 2 e 0 d 6 4 f 9 8 f a 7
H(8) = 5 b e 0 c d 1 9 1 3 7 e 2 1 7 9	H(16) = 4 7 b 5 4 8 1 d b e f a 4 f a 4
<div style="display: flex; justify-content: space-between; width: 100%;"> SHA-256 ; m=8, n=32 SHA-224; m=7, n=32 </div> <div style="display: flex; justify-content: space-between; width: 100%; margin-top: 5px;"> SHA-512; m=8, n=64 SHA-384; m=6, n=64 </div>	

Figure 13 Initial values of standard SHA family

$$\left\{ \begin{array}{ll}
 H(17) = \text{ae5f9156} & \text{e7b6d99b} \\
 H(18) = \text{cf6c85d3} & \text{9d1a1e15} \\
 H(19) = \text{2f73477d} & \text{6a4563ca} \\
 H(20) = \text{6d1826ca} & \text{fd82e1ed} \\
 H(21) = \text{8b43d457} & \text{0a51b936} \\
 H(22) = \text{e360b596} & \text{dc380c3f} \\
 H(23) = \text{1c456002} & \text{ce13e9f8} \\
 H(24) = \text{6f196331} & \text{43a0af0e}
 \end{array} \right.$$

$\underbrace{\hspace{10em}}_{\text{SHA-192; } m=6, n=32}$
 $\underbrace{\hspace{10em}}_{\text{SHA-448; } m=7, n=64}$

Figure 14 Initial values of SHA-192 and SHA-448

We generalize the properties of setting initial hash value for SHA- mn :

- For some x , if $m = 8$ and $n = 64x - 32$ or $64x$, we map to 1st to 8th prime numbers. And the $64x - 32$ bits are obtained by truncating the last 32 bits of the $64x$ bits.
- For some x , if $m = 7$ and $n = 64x - 32$ or $m = 6$ and $n = 64x$, we map to 9th to 16th prime numbers. The $64x - 32$ bits are obtained by truncating the first 32 bits of the $64x$ bits.
- For some x , if $m = 6$ and $n = 64x - 32$ or $m = 7$ and $n = 64x$, we map to 17th ~ 24th prime numbers. The $64x - 32$ bits are obtained by truncating the last 32 bits of the $64x$ bits.

$$\left. \begin{array}{l}
 H(1) = \dots \dots \quad \dots \dots \\
 H(2) = \dots \dots \quad \dots \dots \\
 H(3) = \dots \dots \quad \dots \dots \\
 H(4) = \dots \dots \quad \dots \dots \\
 H(5) = \dots \dots \quad \dots \dots \\
 H(6) = \dots \dots \quad \dots \dots \\
 H(7) = \dots \dots \quad \dots \dots \\
 H(8) = \dots \dots \quad \dots \dots
 \end{array} \right\} \underbrace{\hspace{10em}}_{\substack{m=8, n=64x-32 \\ m=8, n=64x}}$$

$$\left. \begin{array}{l}
 H(09) = \dots \dots \quad \dots \dots \\
 H(10) = \dots \dots \quad \dots \dots \\
 H(11) = \dots \dots \quad \dots \dots \\
 H(12) = \dots \dots \quad \dots \dots \\
 H(13) = \dots \dots \quad \dots \dots \\
 H(14) = \dots \dots \quad \dots \dots \\
 H(15) = \dots \dots \quad \dots \dots \\
 H(16) = \dots \dots \quad \dots \dots
 \end{array} \right\} \underbrace{\hspace{10em}}_{\substack{m=7, n=64x-32 \\ m=6, n=64x}}$$

$$\left. \begin{array}{l}
 H(17) = \dots \dots \quad \dots \dots \\
 H(18) = \dots \dots \quad \dots \dots \\
 H(19) = \dots \dots \quad \dots \dots \\
 H(20) = \dots \dots \quad \dots \dots \\
 H(21) = \dots \dots \quad \dots \dots \\
 H(22) = \dots \dots \quad \dots \dots \\
 H(23) = \dots \dots \quad \dots \dots \\
 H(24) = \dots \dots \quad \dots \dots
 \end{array} \right\} \underbrace{\hspace{10em}}_{\substack{m=6, n=64x-32 \\ m=7, n=64x}}$$

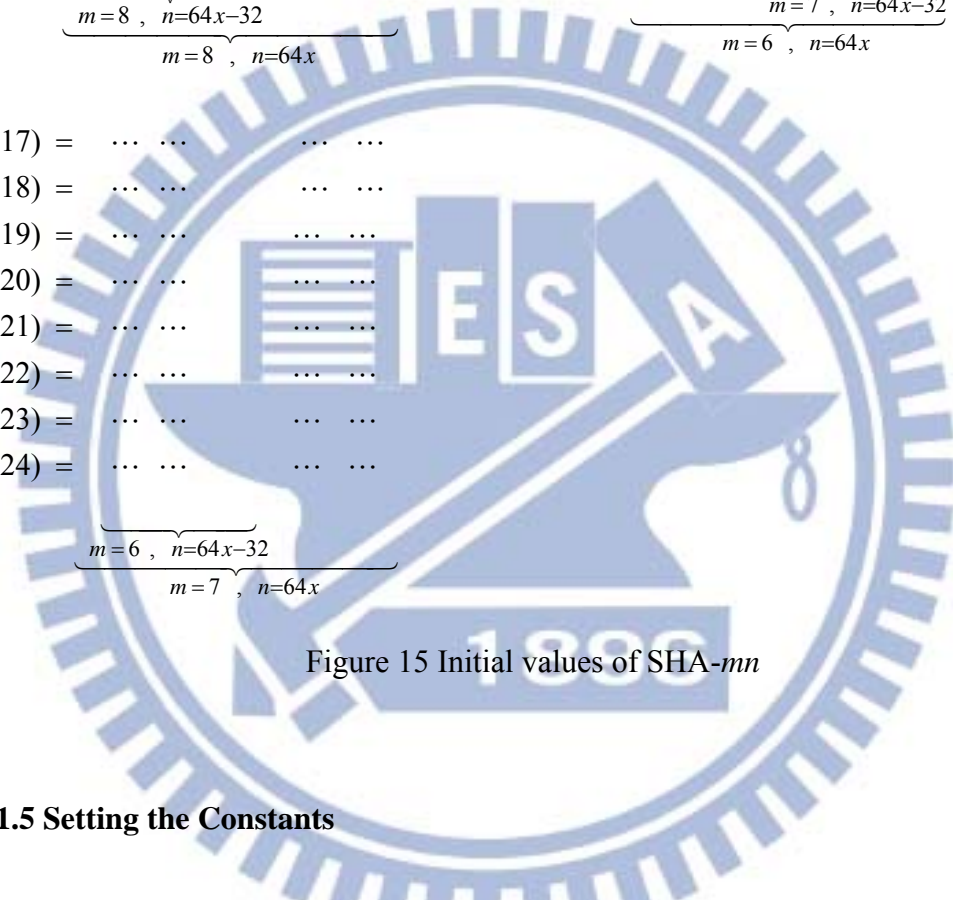


Figure 15 Initial values of SHA-*mn*

4.2.1.5 Setting the Constants

In SHA family, SHA-224 and SHA-256 obtain 64 constants by computing the first 32 bits of the fractional parts of the cube roots of the first 64 prime numbers denoted by $K_0^{\{256\}} \dots K_{63}^{\{256\}}$. Similarly, SHA-384 and SHA-512 obtain 80 constants by computing the first 64 bits of the fractional parts of the cube roots of the first 80 prime numbers denoted by $K_0^{\{512\}} \dots K_{79}^{\{512\}}$.

We can compute the constants by computing the first *n* bits of the fractional parts of the cube roots of the first $f_{13}(n)$ prime numbers.

$$f_{13}(n) = \frac{1}{2}n + 48$$

Eq 25

4.2.1.6 Boolean Expressions and Functions

In SHA- mn , the paper renames $Ch()$ and $Maj()$ functions to g_1 and g_2 and merges some Σ and σ functions described in SHA family. Note that $ROTR^k(x)$ means to rotate right k bits, and $SHR^k(x)$ means to rotate right k bits.

- $\sum_0^{\{224\}}(x) = \sum_0^{\{256\}}(x) = ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x)$ and $\sum_0^{\{384\}}(x) = \sum_0^{\{512\}}(x) = ROTR^{28}(x) \oplus ROTR^{34}(x) \oplus ROTR^{39}(x)$ are merged to $g_3 = ROTR^{f_1(n)}(x) \oplus ROTR^{f_2(n)}(x) \oplus ROTR^{f_3(n)}(x)$, where $f_1(n) = \frac{13}{16}n - 24$, $f_2(n) = \frac{21}{32}n - 8$, and $f_3(n) = \frac{17}{32}n + 5$.
- $\sum_1^{\{224\}}(x) = \sum_1^{\{256\}}(x) = ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x)$ and $\sum_1^{\{384\}}(x) = \sum_1^{\{512\}}(x) = ROTR^{14}(x) \oplus ROTR^{18}(x) \oplus ROTR^{41}(x)$ are merged to $g_4 = ROTR^{f_4(n)}(x) \oplus ROTR^{f_5(n)}(x) \oplus ROTR^{f_6(n)}(x)$, where $f_4(n) = \frac{1}{4}n - 2$, $f_5(n) = \frac{7}{32}n + 4$, and $f_6(n) = \frac{1}{2}n + 9$.
- $\sigma_0^{\{224\}}(x) = \sigma_0^{\{256\}}(x) = ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x)$ and $\sigma_0^{\{384\}}(x) = \sigma_0^{\{512\}}(x) = ROTR^1(x) \oplus ROTR^8(x) \oplus SHR^7(x)$ are merged to $g_5 = ROTR^{f_7(n)}(x) \oplus ROTR^{f_8(n)}(x) \oplus SHR^{f_9(n)}(x)$, where $f_7(n) = -\frac{3}{16}n + 13(\text{mod } n)$, $f_8(n) = -\frac{5}{16}n + 28(\text{mod } n)$, and $f_9(n) = \frac{1}{8}n - 1$.
- $\sigma_1^{\{224\}}(x) = \sigma_1^{\{256\}}(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x)$ and $\sigma_1^{\{384\}}(x) = \sigma_1^{\{512\}}(x) = ROTR^{19}(x) \oplus ROTR^{61}(x) \oplus SHR^6(x)$ are merged to $g_6 = ROTR^{f_{10}(n)}(x) \oplus ROTR^{f_{11}(n)}(x) \oplus SHR^{f_{12}(n)}(x)$, where $f_{10}(n) = \frac{1}{16}n + 15(\text{mod } n)$, $f_{11}(n) = \frac{21}{16}n - 23(\text{mod } n)$, and $f_{12}(n) = -\frac{1}{8}n + 14(\text{mod } n)$.

4.2.1.7 Message Schedule

In SHA-224 and SHA-256, the padded message is parsed into N 512-bit blocks, $M^{(1)} \dots M^{(N)}$, for each 512-bit block, $M^{(i)}$, which is divided into 16 32-bit blocks, $M_0^{(i)} \dots M_{15}^{(i)}$. In SHA-384 and SHA-512, for each 1024-bit block, $M^{(i)}$, which is divided into 16 64-bit blocks, $M_0^{(i)} \dots M_{15}^{(i)}$. The message schedule $\{W_t\}$ is implemented as following.

- $$W_t = \begin{cases} M_t^{(i)} & 0 \leq t \leq 15 \\ \sigma_0^{\{256\}}(W_{t-15}) + \sigma_1^{\{256\}}(W_{t-2}) + W_{t-16} + W_{t-7} & 0 \leq t \leq 63 \end{cases} \text{ and}$$

$$W_t = \begin{cases} M_t^{(i)} & 0 \leq t \leq 15 \\ \sigma_0^{\{512\}}(W_{t-15}) + \sigma_1^{\{512\}}(W_{t-2}) + W_{t-16} + W_{t-7} & 0 \leq t \leq 79 \end{cases} \text{ are merged to}$$

$$W_t = \begin{cases} M_t^{(i)} & 0 \leq t \leq 15 \\ g_5(W_{t-15}) + g_6(W_{t-2}) + W_{t-16} + W_{t-7} & 0 \leq t \leq f_{13}(n) - 1 \end{cases}$$

, where $f_{13}(n) = \frac{1}{2}n + 48$ and the addition(+) is performed modulo 2^n .

4.2.1.8 Initialize the Eight Working Variables

The step initials the eight working variables (a ~ h), with the $(i-1)^{\text{th}}$ hash value. For each message block, $M^{(i)}$, $i = 1, 2, 3 \dots N$, is processed in order, the eight working variables a ~ h are given as

- $$a = H_0^{(i-1)}, b = H_1^{(i-1)}, c = H_2^{(i-1)}, d = H_3^{(i-1)}, e = H_4^{(i-1)}, f = H_5^{(i-1)}, g = H_6^{(i-1)}, h = H_7^{(i-1)},$$

and are generalized as $a_j = H_j(i-1)$ ($0 \leq j \leq 7$).

4.2.1.9 For-Loop Operation

The paper generalizes the for-loop operation of SHA- mn , which is the core part of SHA family algorithms. For each message block $M^{(i)}$, $i = 1, 2, \dots, N$, should be executed $f_{13}(n)$ rounds. Notice that addition (+) is performed modulo 2^n .

Algorithm 12 For-loop Operations

- 1: **For** $t = 0$ to $f_{13}(n)-1$
 - 2: $T_1 = a_7 + g_4(a_4) + g_1(a_4, a_5, a_6) + K_t^{(mn)} + W_t$;
 - 3: $T_2 = g_3(a_0) + g_2(a_0, a_1, a_2)$;
 - 4: $a_7 = a_6$; $a_6 = a_5$; $a_5 = a_4$; $a_4 = a_3 + T_1$;
 - 5: $a_3 = a_2$; $a_2 = a_1$; $a_1 = a_0$; $a_0 = T_1 + T_2$;
 - 6: **End For**
-

4.2.1.10 Compute the i^{th} Intermediate Hash Value $H^{(i)}$

For each $16 \times n$ -bit block $M^{(i)}$, $i = 1, 2, \dots, N$, the intermediate message digests in the SHA family standard execute the following operations:

- $H_0^{(i)} = a + H_0^{(i-1)}$; $H_1^{(i)} = b + H_1^{(i-1)}$; $H_2^{(i)} = c + H_2^{(i-1)}$; $H_3^{(i)} = d + H_3^{(i-1)}$;
 $H_4^{(i)} = e + H_4^{(i-1)}$; $H_5^{(i)} = f + H_5^{(i-1)}$; $H_6^{(i)} = g + H_6^{(i-1)}$; $H_7^{(i)} = h + H_7^{(i-1)}$;

The paper generalizes the equations as follows:

- $H_j^{(i)} = a_j + H_j^{(i-1)}$, $0 \leq j \leq 7$.

4.2.1.11 The Message Digest

After repeating steps N times (i.e., After processing $M(N)$), the $m \times n$ bits message digest of the message is:

- $H_0^{(N)} \parallel H_1^{(N)} \parallel \dots \parallel H_{m-1}^{(N)}$.

4.2.2 SHA(x) Family

The paper reduces the parameter $n = \{32, 64\}$ as $n = 32 \times (2 - i)$, $i \in \{0, 1\}$ and replaces the generalized SHA as SHA(x) defined in Definition 3, which is the family of SHAs of x . The section discusses SHA(1) and SHA(2) first and the *LHV* (*Length-of-the-Hash-Value*) problem of SHA(x).

Definition 3 SHA(x)

$$SHA(x) = \{SHA-m[32 \times (2x-i)] \mid i \in \{0, 1\} \text{ and } m \in \{6, 7, 8\}\} \quad \text{Eq 26}$$

4.2.2.1 SHA(1) and SHA(2)

According to Definition 3, $SHA(1) = \{SHA-192, SHA-224, SHA-256, SHA-384, SHA-448, SHA-512\}$. That is, “Complete SHA family”, which is discussed in Section 5. And $SHA(2) = \{SHA-576, SHA-672, SHA-768, SHA-896, SHA-1024\}$ for $x = 2$. The number of $SHA(2)$ elements is 5, because $m \times n = 768$ when $n = 96, m = 8$ and $n = 128, m = 6$. Therefore, we only use SHA-768 to denote the two cases. The elements of $SHA(2)$ are listed in Table 12.

Table 12 SHA(2)

SHA(2)	SHA-576	SHA-672	SHA-768	SHA-768	SHA-896	SHA-1024
m	6	7	8	6	7	8
n (bits)	96	96	96	128	128	128
$m \times n$ (bits)	576	672	768	768	896	1024

4.2.2.2 Length-of-the-Hash-Value Problem

From the cases of SHA-768 and SHA-1536, which is $m \times n = 1536$ when $n = 256, m = 6$ and $n = 192, m = 8$, it exists LHV problem that some $SHA-r$ cannot be expressed as $r = mn$ uniquely. The LHV problems are classified into 6-7-8-LHV problem, 6-7-LHV problem, 6-8-LHV problem and 7-8-LHV problem. The section defines the LHV problem in Definition 4. Theorem 1 shows that if r is in LHV-set, $SHA-r$ has LHV problem. Otherwise, $SHA-r$ has no LHV problem.

Definition 4 LHV (*Length-of-the-Hash-Value*) problem

- (1) Let $SHA-r$ have a LHV problem if r satisfies $\{r = m \times n = m' \times n' \mid \exists \text{ distinct } n, n' \in \{32(2x-i) \mid x \in N, i = \{0, 1\}\}, \forall m, m' \in \{6, 7, 8\}\}$.

- (2) Let SHA- r have 6-7-8-LHV problem if r satisfies $\{r = 6 \times n = 7 \times n' = 8 \times n'' \mid \exists \text{ distinct } n, n', n'' \in \{32(2x - i) \mid x \in N, i = \{0, 1\}\}\}$.
- (3) Let SHA- r have 6-7-LHV problem if r satisfies $\{r = 6 \times n = 7 \times n' \neq 8 \times n'' \mid \exists \text{ distinct } n, n', n'' \in \{32(2x - i) \mid x \in N, i = \{0, 1\}\}\}$.
- (4) Let SHA- r have 6-8-LHV problem if r satisfies $\{r = 6 \times n = 8 \times n' \neq 7 \times n'' \mid \exists \text{ distinct } n, n', n'' \in \{32(2x - i) \mid x \in N, i = \{0, 1\}\}\}$.
- (5) Let SHA- r have 7-8-LHV problem if r satisfies $\{r = 7 \times n = 8 \times n' \neq 6 \times n'' \mid \exists \text{ distinct } n, n', n'' \in \{32(2x - i) \mid x \in N, i = \{0, 1\}\}\}$.

Theorem 1 LHV Sets

- (1) Let 6-7-8-LHV-set = $\{5376k \mid k \in N\}$. If r is in 6-7-8-LHV-set, SHA- r has a 6-7-8-LHV problem. Otherwise, SHA- r has no 6-7-8-LHV problem.
- (2) Let 6-7-LHV-set = $\{1344k \mid k \in N\} - \{5376k \mid k \in N\}$. If r is in 6-7-LHV-set, SHA- r has a 6-7-LHV problem. Otherwise, SHA- r has no 6-7-LHV problem.
- (3) Let 6-8-LHV-set = $\{768k \mid k \in N\} - \{5376k \mid k \in N\}$. If r is in 6-8-LHV-set, SHA- r has a 6-8-LHV problem. Otherwise, SHA- r has no 6-8-LHV problem.
- (4) Let 7-8-LHV-set = $\{5376k \mid k \in N\} - \{5376k \mid k \in N\}$. If r is in 7-8-LHV-set, SHA- r has a 7-8-LHV problem. Otherwise, SHA- r has no 7-8-LHV problem.

Proof:

Proof for (1) 6-7-8-LHV-set:

We show that if r is in 6-7-8-LHV-set, SHA- r has a 6-7-8-LHV problem by induction subject to k .

- (i) For $k = 1$, $5376k = 5376$, $5376 = 6 \times 896 = 7 \times 768 = 8 \times 672$, SHA-5376 has a 6-7-8-LHV problem.
- (ii) Assume $k = x$, $5376x = 6 \times 896x = 7 \times 768x = 8 \times 672x$, SHA-5376 x has a 6-7-8-LHV problem.

(iii) Then, when $k = x + 1$, $5376(x+1) = 6 \times [896(x+1)] = 7 \times [768(x+1)] = 8 [672(x+1)]$, SHA-1792($x+1$) has a 7-8-LHV problem.

The proofs for 6-7-LHV-set, 6-8-LHV-set, and 7-8-LHV-set are similar to the proof for 6-7-8-LHV-set.

Q.E.D.

As defined, $SHA(x) = \{SHA-6 \times (64x-32), SHA-6 \times 64x, SHA-7 \times (64x-32), SHA-7 \times 64x, SHA-8 \times (64x-32), SHA-8 \times 64x\}$, where $x \in N$ and 6-7-8-LHV-set = $\{5376k | k \in N\}$, 6-7-8-LHV problem exists between $SHA(x)$ and $SHA(x')$ if $x, x' \in N$ such that $SHA-r \in SHA(x) \cap SHA(x')$. For example, if we take $(x, x') = (12, 14)$, $SHA(12) = \{SHA-4416, SHA-4608, SHA-5152, SHA-5376, SHA-5888, SHA-6144\}$ and $SHA(14) = \{SHA-5184, SHA-5376, SHA-6048, SHA-6272, SHA-6912, SHA-7168\}$, it is found that $SHA-5376 \in SHA(12) \cap SHA(14)$, thus SHA-5376 has 6-7-8-LHV problem. Similarly, $SHA-1344 \in SHA(3) \cap SHA(4)$ and SHA-1344 has 6-7-LHV problem. All the situations of the LHV problem within $SHA(x)$ are categorized in **Lemma 1**.

Lemma 1 *LHV Sets in $SHA(x)$*

- (1) *If $(x, x') \in \{(12i, 14i), (24i-12, 21i-10), (24i, 21i), (28i-14, 21i-10), (28i, 21i) | i \in N\}$, there is 6-7-8-LHV problem between $SHA(x)$ and $SHA(x')$.*
- (2) *If $(x, x') \in \{(6i-3, 7i-3) | i \in N\} \cup \{(12i-6, 14i-7) | i \in N\}$, there is a 6-7-LHV problem between $SHA(x)$ and $SHA(x')$.*
- (3) *If $(x, x') \in \{(3i+2, 4i+2) | i \in N - \{7k-3 | k \in N\}\} \cup \{(3i, 4i) | i \in N - \{7k | k \in N\}\}$, there is a 6-8-LHV problem between $SHA(x)$ and $SHA(x')$.*
- (4) *If $(x, x') \in \{(7i+4, 8i+4) | i \in N - \{3k-2 | k \in N\}\} \cup \{(7i, 8i) | i \in N - \{3k | k \in N\}\}$, there is a 7-8-LHV problem between $SHA(x)$ and $SHA(x')$.*

Proof:

The proof for (1):

According Definition 3, $SHA(x) = \{SHA-(384x - 192), SHA-384x, SHA-(448x - 224), SHA-(512x - 256), SHA-512x\}$, if there exists 6-7-8-LHV problem within $SHA(x)$, at least two elements of $SHA(x)$ are multiple of 5376. We discuss the following six cases:

- When $(384x-192)$ is multiple of 5376, $384x-192 = 5376k \Rightarrow x = (28k+1)/2$. x and k must be integral, but $\forall k \in N$, x is not integral. Thus, we ignore this case.
- When $(384x)$ is multiple of 5376, $384x = 5376k \Rightarrow x \in \{14i \mid i \in N\}$.
- When $(448x-224)$ is multiple of 5376, $448x - 224 = 5376k \Rightarrow x = (24k+1)$. x and k must be integral, $\forall k \in N$, x is not integral. Thus, we ignore this case.
- When $(448x)$ is multiple of 5376, $448x = 5376k \Rightarrow x \in \{12i \mid i \in N\}$.
- When $(512x-256)$ is multiple of 5376, $512x - 256 = 5376k$, $x \in \{21i-10 \mid i \in N\}$.
- When $(512x)$ is multiple of 5376, $512x = 5376k \Rightarrow x \in \{21i \mid i \in N\}$.

So, there exists 6-7-8-LHV problem between $SHA(x)$ and $SHA(x')$ if $(x, x') \in \{(12i, 14i), (24i-12, 21i-10), (24i, 21i), (28i-14, 21i-10), (28i, 21i) \mid i \in N\}$.

The proofs for (2) to (4) are similar to (1).

Q.E.D.

4.2.2.3 $SHA'(x)$ without LHV problem

The previous section defines LHV problem and proves all the situations of the LHV problem within $SHA(x)$. Consider the length of one word in the form of $32 \times 2^{k-1}$ for $k \in N$, if we take $m = 6$, $n = 32 \times 2^3 = 256$, $r = m \times n = 1536$. However, $n' = 1536/8 = 192 \notin \{32 \times 2^{k-1} \mid k \in N\}$. That is, $\nexists n \in \{32 \times 2^{k-1} \mid k \in N\}$ such that $1536 = 6 \times 256 = 8 \times n$. We solve 6-8-LHV problem. Lemma 2 will prove that $SHA-m \times (32$

$\times 2^{k-1}$) has no LHV problem and redefine $SHA'(x)$ in Definition 3. Therefore, we have $SHA'(x) = \{SHA-192 \times 2^{x-1}, SHA-224 \times 2^{x-1}, SHA-256 \times 2^{x-1} \mid x \in N\}$.

Lemma 2 Let $X = \{6, 7, 8\}$, $Y = \{32 \times 2^{k-1} \mid k \in N\}$, $SHA-m \times n$ has no LHV problem for all distinct $m, m' \in X$ and all distinct $n, n' \in Y$.

Proof:

Without loss of generality, we let $n = 32 \times 2^{a-1} = 2^{a+4}$ and $n' = 32 \times 2^{b-1} = 2^{b+4}$, for all $a, b \in N, a > b, a \neq b$.

Suppose $mn = m'n'$ for distinct $m, m' \in X$, we have $m \times 2^{a+4} = m' \times 2^{b+4} \Rightarrow m'/m = 2^{a-b} \geq 2$. It exists contradiction because a and b do not exist. Therefore, $\exists x, x' \in \{(192 \times 2^{k-1}), (224 \times 2^{k-1}), (256 \times 2^{k-1}) \mid k \in N\}$, there is no LHV problem between $SHA(x)$ and $SHA(x')$.

Q.E.D.

Definition 5 $SHA'(x)$ without LHV problem:

$$SHA'(x) = \{SHA-m[32 \times (2^{x-1})] \mid x \in N \text{ and } m \in \{6, 7, 8\}\} \quad \text{Eq 27}$$

4.3 Finding Near-Optimum Message Scheduling Settings for SHA-256 Variants Using Genetic Algorithms [70]

4.3.1 SHA Message Scheduling Evaluation Criterion

This section proposes an evaluation criterion of SHA message scheduling. The number of terms involved in the message schedule is treated as an evaluation criterion of SHA message scheduling. This study uses SHA-0 and SHA-1 as examples to show that SHA-1 is more secure than SHA-0 by comparing their message scheduling equations.

4.3.1.1 Local Collision

A local collision appearing on all the SHA families is a collision within intermediate steps of the hash function [29]. The starting point for hash function collision attacks is a local collision. Local collisions are found using linear approximations of Boolean functions that are used in various rounds in message scheduling (and other conditions as defined in [29]). The first observation is that SHA-0 has a 6-step local collision that can start at any step i . The differential path is a sequence of grouped local collisions with possible overlaps [71]. Wang [29] tried to find a set of starting steps for each local collision to construct such a path. The disturbance vector is applied to satisfy the recursion defined by the message expansion. Once a local collision is found, an attempt is made to consider the message expansion and other non-linear designs to find a collision for the full hash function. For SHA-0, 3 vectors are found successfully for three conditions in [29]. However, it is more complicated to find a good disturbance vector due to the large search space on SHA-1, and the probability of n interleaved local collision complexities increases

exponentially with n for SHA-256 [32].

Mendel provides an approach for collision searches as follows [32]:

- (1) Identify local collisions in each round of transformation.
- (2) Search for disturbance vectors that need to satisfy some additional properties.
- (3) Build the difference vector by interleaving the local collisions.
- (4) The complexity of the collision search is related to the characteristic within these interleaved local collisions.
- (5) Adjusting message bits for the chosen characteristic reduces the computational cost for the collision search.

The issue that arises is how to reduce the number of local collisions in an expansion process. Our study applies a genetic approach to find the optimal parameter set of the SHA family message expansion function based on the evaluation criterion with the lowest number of local collisions.

4.3.1.2 Local Collision in SHA-0 and SHA-1

In [27], it is pointed out that SHA-1 is safer than SHA-0 because of a single bit-wise rotation in SHA-1 that affects the local collisions existing in SHA-0. Table 13 shows the SHA-0 and SHA-1 equations.

Table 13 SHA-0, SHA-1, and SHA-256-XOR equations

Algorithm	Equation
SHA-0	$W_t = \begin{cases} M_t^{(i)} & ,0 \leq t \leq 15 \\ W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16} & ,16 \leq t \leq 79 \end{cases}$
SHA-1	$W_t = \begin{cases} M_t^{(i)} & ,0 \leq t \leq 15 \\ ROTL(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) & ,16 \leq t \leq 79 \end{cases}$
SHA-256-XOR	$W_t = \begin{cases} M_t^{(i)} & ,0 \leq t \leq 15 \\ \sigma_0^{(256)}(W_{t-2}) \oplus W_{t-7} \oplus \sigma_1^{(256)}(W_{t-1}) \oplus W_{t-16} & ,16 \leq t \leq 63 \end{cases}$

The following are examples that compare the terms involved in W_{27} in both SHA-0 and SHA-1, and that in W_{20} in SHA-256-XOR where M_j^n (or W_j^n) indicates that the message block M_j (or intermediate message word W_j) undergoes an n-bitwise left rotation. Each message word W_t is obtained by recursively computing other words with lower indices and being replaced by message blocks until $t \leq 15$.

Figure 2 represents the number of terms involved in full SHA-0, SHA-1, and SHA-256-XOR.

[SHA-0]

$$\begin{aligned}
 W_{27} &= W_{24} \oplus W_{19} \oplus W_{13} \oplus W_{11} \\
 &= (W_{21} \oplus W_{16} \oplus W_{10} \oplus W_8) \oplus W_{19} \oplus W_{13} \oplus W_{11} \\
 &= \dots \\
 &= M_{15} \oplus M_4 \oplus M_2 \oplus M_7 \oplus M_8 \oplus M_3 \\
 &\Rightarrow 6 \text{ terms are involved.}
 \end{aligned}$$

[SHA-1]

$$\begin{aligned}
 W_{27} &= W_{24}^1 \oplus W_{19}^1 \oplus W_{13}^1 \oplus W_{11}^1 \\
 &= (W_{21}^2 \oplus W_{16}^2 \oplus W_{10}^2 \oplus W_8^2) \oplus W_{19}^1 \oplus W_{13}^1 \oplus W_{11}^1 \\
 &= \dots \\
 &= M_{15}^4 \oplus M_{10}^4 \oplus M_4^4 \oplus M_2^4 \oplus M_{13}^3 \oplus M_7^3 \oplus M_5^3 \oplus M_{10}^2 \oplus \\
 &\quad M_8^2 \oplus M_{11}^2 \oplus M_5^2 \oplus M_3^2 \oplus M_{13}^1 \oplus M_{11}^1 \\
 &\Rightarrow 14 \text{ terms are involved.}
 \end{aligned}$$

[SHA-256-XOR]

$$\begin{aligned}
 W_{20} &= \sigma_0(W_{18}) \oplus W_{13} \oplus \sigma_1(W_{15}) \oplus W_4 \\
 &= W_{18}^7 \oplus W_{17}^{18} \oplus W_{13} \oplus W_5^{17} \oplus W_5^{19} \oplus W_4 \\
 &= (W_{14}^{14} \oplus W_{14}^{25} \oplus W_9^7 \oplus W_1^{24} \oplus W_1^{26} \oplus W_6^7 \oplus W_{14}^4 \oplus W_9^{18} \\
 &\quad \oplus W_1^3 \oplus W_5^5 \oplus W_0^{18} \oplus W_{11} \oplus W_3^{17} \oplus W_3^{19} \oplus W_2)^7 \oplus \\
 &\quad (W_{14}^{14} \oplus W_{14}^{25} \oplus W_9^7 \oplus W_1^{24} \oplus W_1^{26} \oplus W_6^7 \oplus W_{14}^4 \oplus W_9^{18} \\
 &\quad \oplus W_1^3 \oplus W_5^5 \oplus W_0^{18} \oplus W_{11} \oplus W_3^{17} \oplus W_3^{19} \oplus W_2)^{18} \oplus W_5^{17} \\
 &\quad \oplus W_3^{19} \oplus W_4 \\
 &= M_{14}^{21} \oplus M_{14}^{32} \oplus M_9^{14} \oplus M_1^{31} \oplus M_1^1 \oplus M_0^{14} \oplus M_{11}^7 \oplus M_3^{24} \\
 &\quad \oplus M_3^{26} \oplus M_2^7 \oplus M_{14}^{11} \oplus M_{14}^{22} \oplus M_9^4 \oplus M_1^{21} \oplus M_0^0 \oplus \\
 &\quad M_{11}^{18} \oplus M_3^3 \oplus M_3^5 \oplus M_2^{18} \oplus M_5^{17} \oplus M_5^{19} \oplus M_4 \\
 &\Rightarrow 22 \text{ terms are involved.}
 \end{aligned}$$

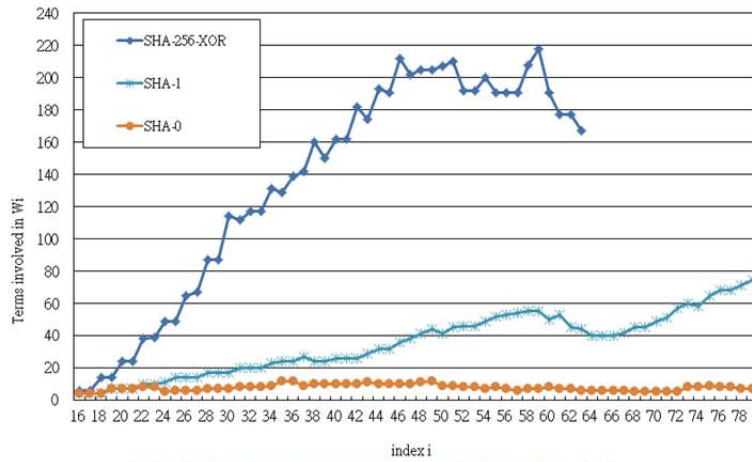


Figure 16 Comparison of the number of terms involved in each W_i in message scheduling for SHA-0, SHA-1 and SHA-256

4.3.1.3 Our Conjecture

Matusiewicz et al. proved that the functions (Σ and σ) or the message expansion are essential for the security of SHA-256 by showing that it is possible to find collisions with a complexity of 264 hash operations for a variant without them [72].

We proposed that message scheduling of the SHA algorithm has higher security complexity (or fitness) if each message word (W_i) involves more message blocks (M_i) in each round.

Chabaud and Joux showed that SHA-1 is more secure than SHA-0 [27]. Furthermore, Wang found collisions in full SHA-0 and SHA-1 hash operations with complexities less than 2^{39} [71] and 2^{69} [29], respectively.

Consider the analyses of the terms involved in each message block. Figure 16 clearly shows that the number of terms involved in SHA-1 is more than that in SHA-0, taking W_{27} as an example ($14 > 6$). Therefore, SHA-1 has a higher security complexity (hence security fitness) than SHA-0. In this paper, we use the term “security fitness” to evaluate the security of each possible W_i in message scheduling.

4.3.1.4 The Best Setting of Message Scheduling Equation in SHA-1

The message scheduling equation in SHA-1 can be generalized as

$$W_t = \begin{cases} M_t^{(i)} & , 0 \leq t \leq 15 \\ ROTL^1(W_{t-A} \oplus W_{t-B} \oplus W_{t-C} \oplus W_{t-D}) & , 16 \leq t \leq 79 \end{cases} \quad \text{Eq 28}$$

The best four variables are produced by the brute force (or exhaustive) approach, and the values found are $\{A, B, C, D\} = \{1, 2, 11, 16\}$. The best complexity occurs in round 60 when 212 terms are involved. The modified equation is

$$W_t = \begin{cases} M_t^{(i)} & , 0 \leq t \leq 15 \\ ROTL^1(W_{t-1} \oplus W_{t-2} \oplus W_{t-11} \oplus W_{t-16}) & , 16 \leq t \leq 79 \end{cases} \quad \text{Eq 29}$$

and called optSHA-1 [73]. Figure 17 compares the number of terms involved in SHA-1 and optSHA-1.

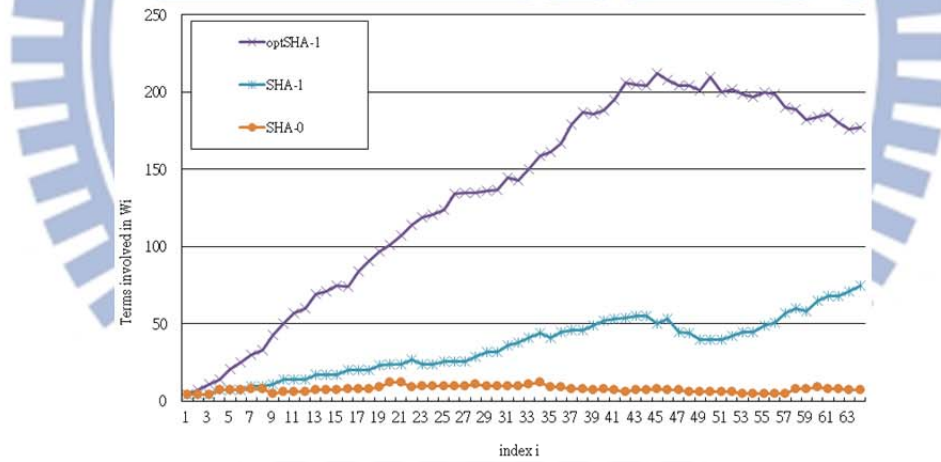


Figure 17 Comparison of the number of terms involved in each W_t in message scheduling for SHA-1 and SHA-1-OPT

4.3.2 Improving SHA-256-XOR Via Genetic Algorithms

4.3.2.1 Specialized GA for SHA-256-XOR

To find optimum parameters, the message scheduling equation in SHA-256-XOR

can be generalized as

$$W_t = \begin{cases} M_t^{(i)} & , 0 \leq t \leq 15 \\ \sigma_0^{\{256\}}(W_{t-A}) \oplus W_{t-B} \oplus \sigma_1^{\{256\}}(W_{t-C}) \oplus W_{t-D} & , 16 \leq t \leq 63 \end{cases} \quad \text{Eq 30}$$

and

$$\begin{aligned} \sigma_0^{\{256\}}(x) &= ROTL^7(x) \oplus ROTL^{18}(x) \oplus SHR^3(x) \\ \sigma_1^{\{256\}}(x) &= ROTL^{17}(x) \oplus ROTL^{19}(x) \oplus SHR^{10}(x) \end{aligned} \quad \text{Eq. 13}$$

Consider two operations, *ROTL* and *SHR*. A bitwise rotation operation, *ROTL*, is a circular shift operation that is a permutation of the entries in a tuple where the last element becomes the first element and all of the other elements are shifted. The shift operation, $SHR^n(x)$, which sets 0 as the first element, does not influence the experimental results because $SHR^n(x)$ and $ROTR^n(x)$ produce different results. Based on this assumption, the generalized form is modified to

$$\begin{aligned} \sigma_0^{\{256\}}(x) &= ROTL^7(x) \oplus ROTL^{18}(x) \\ \sigma_1^{\{256\}}(x) &= ROTL^{17}(x) \oplus ROTL^{19}(x) \end{aligned} \quad \text{Eq 31}$$

In the previous section, the optimal values are calculated using the brute force approach in otpSHA-1. To find the optimum parameters using the brute force approach for SHA-256-XOR, we would need to test 2^{64} possible combinations of $\{A, B, C, D\}$ for each round t ($16 \leq t \leq 63$), and to perform up to 48×2^{94} operations in the whole experiment. We applied genetic algorithm operators of recombination and perturbation to reduce the number of infeasible solutions needed to find the near optimal variable set $\{A, B, C, D\}$.

The design of the GA involves some main components: genetic representation, population initialization, fitness function, selection scheme, crossover, and mutation. Each component is described as follows, and the parameters used with GAUL are listed in Table 14:

- Genetic representation: The genes represent the input variables, A, B, C, D, t , of the generalized SHA-256-XOR, and each chromosome represents a possible solution. In the simulation, the length of each chromosome is 5.
- Population initialization: Each chromosome presents a potential solution for the problem in genetic algorithms. The initial population is randomly generated and the size is set to 500.
- Fitness Function: The fitness function counts the number of terms in the equation for W_i . After the process of selection, crossover, and mutation, the optimal chromosome indicates the maximum number of terms involved in the equations.

$$F(t) = \# \text{ of different terms involved in } W_i \text{ equation} \quad \text{Eq 32}$$

- Selection Scheme: Selection is a genetic operator that chooses a chromosome from the current generation's population for inclusion in the next generation's population. We adopt the binary tournament selection based on the fitness value in the simulation.
- Crossover and Mutation: Crossover enables genetic algorithms to extract the best genes from different individuals, and to produce potentially superior children. The mutation operation randomly modifies the gene to prevent the falling of all solutions into a local optimum, and extends the search space. In the simulation, we adopt the one-point crossover with a ratio 0.9, and a single-point mutation with a ratio 0.1.

Table 14 Genetic algorithm parameters

Parameter	Value
Library	GAUL
Population size	500
Number of chromosomes	1
Length of each chromosome	5
Evolutionary mode	GA_SCHEME_DARWIN
Elitism mode	GA_ELITISM_PARENTS_SURVIVE
Crossover ratio	0.9
Mutation ratio	0.1
Fitness function	# of terms involved in W_i equation



4.4 Modified Autonomous Key Management [74]

This section modifies the secret sharing of Autonomous Key Management (AKM). AKM runs dynamically in seven node-based/region-based operations. The seven operations are update, join, leave, merge, partition, expansion, and contraction.

These operations are designed based on the following rules:

- (1) All leaves in the hierarchy of AKM are real nodes. Each real node i has its own secret key SK_i , and $PK_i = g^{SK_i} \bmod p$, where g is a random generator.
- (2) The non-leaf nodes are virtual nodes, and their secret keys are generated directly/indirectly from real nodes through some region-based operations.
- (3) A tree with node A as root is called $Region_A$. For example, region A has virtual nodes B_1, B_2 , and real nodes $C_{1,1}, C_{1,2}, C_{1,3}, C_{2,1}, C_{2,2}, C_{2,3}$, and $C_{2,4}$. The number of the nodes that know the secret of region is *Overall Region Size (ORS)*.
- (4) The *Regional Trust Coefficient (RTC)* is the ratio of the threshold to *ORS*, and indicates how secure the region is. The AKM sets a *Global Trust Coefficient (GTC)* as a lower bound of all the *RTC*. Figure 6 shows an example, in which the *ORS* is 4 and *RTC* is 0.75 of the region B_2 . The *GTC* of region A would be 0.2.

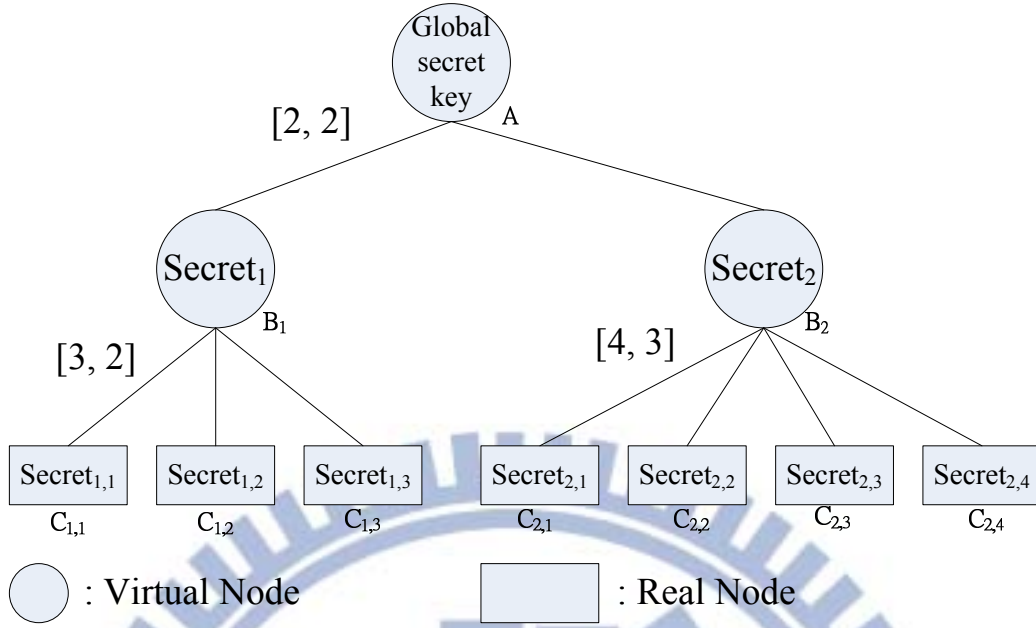


Figure 6 An example of AKM

4.4.1 Function Update

Function update prevents any intruders from compromising the secret, and the AKM updates keys periodically. First, the region with (n, t) -threshold must select t nodes and each node is indicated as node $i \in 1, \dots, t$.

Each node i generates update share $S_{i,j} (1 \leq j \leq n)$ of key 0. The node i selects random numbers $x_j (1 \leq j \leq n)$ and $r_d (0 \leq d \leq i - 1)$ to compute coefficients $a_d = (r_d | 0) (0 \leq d \leq t - 1)$. $S_{i,j} = a_d(x_j) = \sum_{r=0}^{t-1} a_r(x_j)^r \pmod{p}$, for $1 \leq j \leq n$. Node i then distributes $S_{i,j}$ to node $j \in 1, \dots, n$. When node j receives the update shares distributed from other t nodes in the region, it computes a new share

$$S'_j = S_j + \sum_{i=1}^t S_{i,j} \pmod{p} \quad \text{Eq 33}$$

The previous section describes how AKM can manage its secret sharing hierarchical structure using seven region-based functions. These operations cover all possible region changes from node joining to leaving. The key update frequency in

MANET is adjustable depending on the application environment. If the frequency is high, the MANET would be secure enough against adversaries, but would result in lower performance and heavy power consumption. On the contrary, if the frequency is low, the communication between nodes in MANET suffers from key inconsistency after many nodes join and leave continuously.

4.4.2 Function Join

Function Join is used when a node i wants to join a (t, n) -threshold region. The node sends a request to node $j \in 1, \dots, t$ in the region. Upon receiving the request, node j checks its *certificate revoking list (CRL)* first. If node j accepts the request, it computes a partial share S'_j of node i :

$$S'_j = S_j l_j(i) + \Delta_j \pmod{q} \quad \text{Eq 34}$$

, where

$$l_j(i) = \prod_{r=1, r \neq j}^t \frac{ID_i - ID_r}{ID_j - ID_r} \pmod{q} \quad \text{Eq 35}$$

$$\Delta_j = \sum_{r=1, r \neq j}^t \sigma(j-r) \cdot S_{j,r} \quad \text{Eq 36}$$

that $S_{j,r}$ is a number which pairs of nodes $(j, r) \in 1 \leq j \leq t, 1 \leq r \leq t$, and

$$\sigma(x) = \begin{cases} 1 & , x > 0 \\ -1 & , x < 0 \\ 0 & , \text{otherwise} \end{cases} \quad \text{Eq 37}$$

After receiving all partial shares, node i generates its secret share S_i :

$$S_i = \sum_{j=1}^t S'_j = \sum_{j=1}^t S_j l_j(ID_i) + \sum_{j=1}^t \Delta_j \pmod{q} \quad \text{Eq 38}$$

4.4.3 Function Leave

Function Leave is used when a node leaves a region. Any node j removes the certificate of node i from its key management records when receiving Leave request from node i or detecting the node leaves. The share key of node j does not change until the AKM updates key periodically.

4.4.4 Function Merge

Function Merge is used when the number of nodes in a region is below the threshold. The region is simply divided into many parts and they join to the other sibling regions respectively. As in Algorithm 13, AKM performs Function Merge on region S_i and merges its nodes $S_{i,1}$ to $S_{i,r}$ into regions S_j and S_k as $S_{j,(n+1)}, \dots, S_{j,(n+p)}$ and $S_{k,(n+1)}, \dots, S_{k,(n+q)}$.

Algorithm 13 Merge

// **Require:** The merged region S_i which contains nodes $S_{i,1}, \dots, S_{i,r}$, and the destination t regions $S_{D_0}, S_{D_1}, \dots, S_{D_{t-1}}$.
//**Ensure:** Region $S_{D_0}, S_{D_1}, \dots, S_{D_{t-1}}$.
1: Separate S_i into t parts: $[S_{i,1}, \dots, S_{i, \lceil r/t \rceil}]$, $[S_{i, \lceil r/t \rceil + 1}, \dots, S_{i, 2 \lceil r/t \rceil}]$, \dots , $[S_{i, (t-2) \lceil r/t \rceil + 1}, \dots, S_{i, (t-1) \lceil r/t \rceil}]$,
 $[S_{i, (t-1) \lceil r/t \rceil + 1}, \dots, S_{i,r}]$
2: **For** $u = 0$ to $t - 2$ **Do**
3: **For** $v = 1$ to $\lceil \frac{r}{t} \rceil$ **Do**
4: Join $S_{i, u \lceil r/t \rceil + v}$ into S_{D_u}
5: **End For**
6: **End For**
7: **For** $v = 1$ to $r - t \lceil \frac{r}{t} \rceil$ **Do**
8: Join $S_{i, (t-1) \lceil r/t \rceil + v}$ into $S_{D_{t-1}}$
9: **End For**

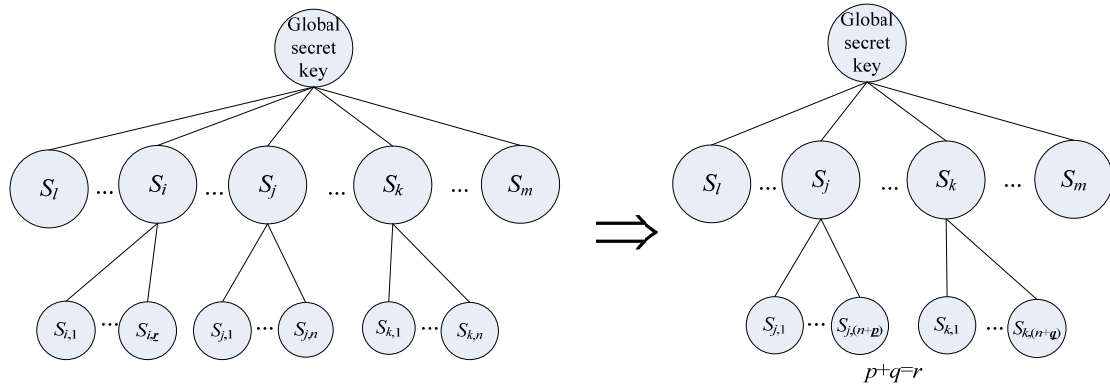


Figure 18 Function Merge – merges S_i into S_j and S_k

4.4.5 Function Partition

Function Partition is used when the *RTC* of a region is under the *GTC*. For example, Figure 19 shows that AKM partitions region S_i with $2n$ nodes into S_i and $S_{(m+1)}$ with the same size n under threshold k . To assign the secret share to the nodes in $S_{(m+1)}$, it first randomly selects t regions from S_1 to S_m and randomly chooses t nodes $\{S_{j,1}, \dots, S_{j,t}\}$ from each S_j region. Second, it creates a new node $S_{(m+1)}$, and joins into AKM.

Note that

$$S_i = \sum_{j=1}^t S_j l_j(ID_{S_i}) \pmod{q} \quad \text{Eq 39}$$

, where

$$l_j(ID_{S_i}) = \prod_{r=1, r \neq j}^t \frac{ID_{S_i} - ID_{S_r}}{ID_{S_j} - ID_{S_r}} \pmod{q} \quad \text{Eq 40}$$

by Lagrange interpolation. Note that

$$S_j = \sum_{v=1}^t S_{j,v} l_{j,v}(0) \pmod{q} \quad \text{Eq 41}$$

, where

$$l_{jv}(0) = \prod_{r=1, r \neq j}^t \frac{ID_{S_{j,r}}}{ID_{S_{j,r}} - ID_{S_{j,v}}} \pmod{q} \quad \text{Eq 42}$$

Thus

$$S_i = \sum_{j=1}^t \sum_{v=1}^t S_{j,v} l_{j,v}(0) l_j(ID_{S_i}) \pmod{q} \quad \text{Eq 43}$$

We also can get

$$S_{m+1} = \sum_{j=1}^t \sum_{v=1}^t S_{j,v} l_{j,v}(0) l_j(ID_{S_{m+1}}) \pmod{q} \quad \text{Eq 44}$$

, where

$$l_j(ID_{S_{m+1}}) = \prod_{r=1, r \neq j}^t \frac{ID_{S_{m+1}} - ID_{S_r}}{ID_{S_j} - ID_{S_r}} \pmod{q} \quad \text{Eq 45}$$

To generate each share $S_{(m+1),j}$ ($1 \leq j \leq n$) of region $S_{(m+1)}$, $S_{(m+1),v}'$, where

$$S_{(m+1),v}' = S_{(m+1),v} l_{(m+1),v}(0) R_{(m+1)} \pmod{q} \quad \text{Eq 46}$$

$$R_{(m+1)} = l_{(m+1)}(ID_{S_{m+1}}) - l_j(ID_{S_i}) \quad \text{Eq 47}$$

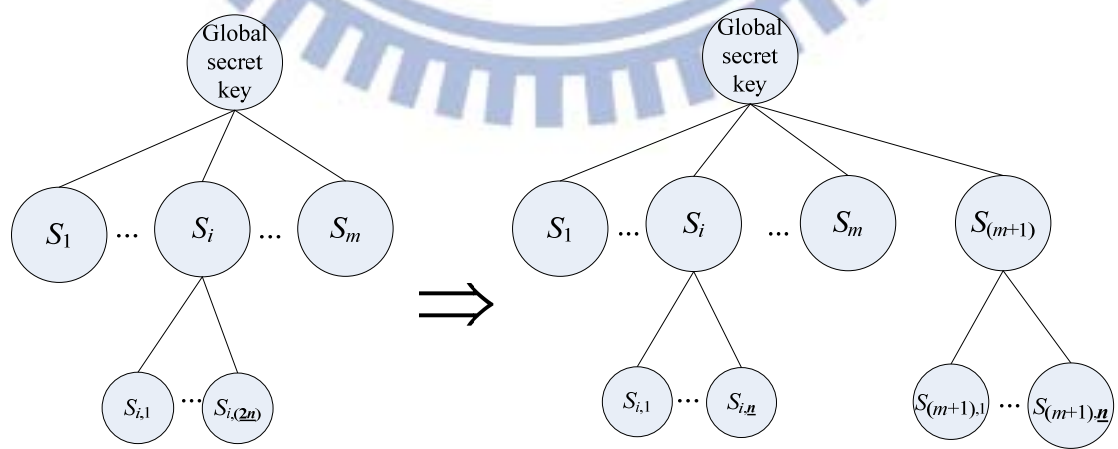


Figure 19 Function Partition – partition of S_i into S_j and S_{m+1}

4.4.6 Function Expansion

Function Expansion is used when the *RTC* of a region is under the *GTC*. AKM must perform expansion operation to extend the hierarchy when the *RTCs* are under or equal to *GTC* in all the AKM regions. The function ensures that all the *RTCs* of regions are not lower than *GTC* when nodes increase continuously. Figure 20 shows that AKM extends region S_i from one level to two levels with the same threshold. It selects t nodes in region S_i , and executes function join to create a new node $S_{i,(n+1)}$. It then moves $S_{i,1}, \dots, S_{i,m}$ to be $S_{i,(n+1)}$'s children, $S_{i,(n+1),1}, \dots, S_{i,(n+1),m}$ with shares $S_{i,(n+1),j}, 1 \leq j \leq m$, that

$$S_{i,(n+1),j} = a(ID_{i,(n+1),j}) = \sum_{r=1}^t a_r x^r \pmod{q} \quad \text{Eq 48}$$

, where $a_r = r_r | s_r (1 \leq r \leq t)$, $S_{i,(n+1)} = s_t s_{t-1} \dots s_1$, and all r, s are the same used in region S_i . Region $S_{i,(n+1)}$ continues (n, t) -threshold as in region S_i .

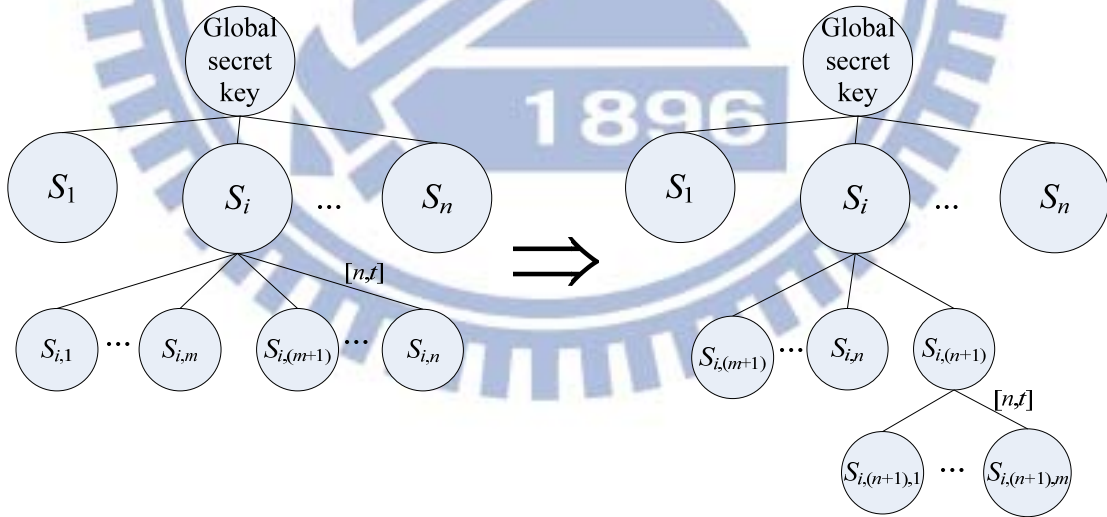


Figure 20 Function Expansion

4.4.7 Function Contraction

Function Contraction is the opposite of function Expansion. This function is used

when the number of nodes is less than the threshold in the region. The function merges the nodes contained in the contracted region into the other regions whose *RTCs* are less than *GTC* and decreases the level of AKM. As in Algorithm 14, AKM performs Function Contraction on region $S_{i,(m+1)}$ and merges its nodes $S_{i,(m+1),1}$ to $S_{i,(m+1),r}$ into regions S_i and S_j as $S_{i,(m+1)}, \dots, S_{i,(m+p)}$ and $S_{j,(n+1)}, \dots, S_{j,(n+q)}$.

Algorithm 14 Contraction

// **Require:** Region S_i which contains nodes $S_{i,1}, \dots, S_{i,r}$.
// **Ensure:** Region $S_{D_0}, S_{D_1}, \dots, S_{D_{t-1}}$.
1: Merge S_i into $\{S_{D_0}, S_{D_1}, \dots, S_{D_{t-1}}\}$
2: **IF** $S_i \notin \{S_{D_0}, S_{D_1}, \dots, S_{D_{t-1}}\}$
3: Delete S_i
4: **End IF**

The seven-region-based operations on MANET of modified AKM handle key management. The scheme needs a *trusted authority (TA)* to start up, neither any central authorities to compute and distribute shares.

5. Discussion and Analyses

5.1 Cryptanalysis of Transpositional AES

In this section, we give the experiment results of linear cryptanalysis, differential cryptanalysis, and square attack analysis, comparing the proposed cipher to AES. The differences between the proposed cipher and the original AES are summarized in Table 15.

Table 15 Differences between AES_Plus and AES

	AES_Plus	AES
Structure Type	Feistel structure	Square structure
Plaintext/Ciphertext Length	128 bits	128
Cipher Key Length	128 ~ 256 bits (multiple of bytes)	128 ~ 256 bits (multiple of words)
Number of Rounds	10	10 ~ 14 (depends on block length and key length)
Round Transformations (Operation Unit)	TransByte (64-bit), SubBlkXor (64-bit)	ShiftRow (byte), MixColumn (word)

5.1.1 Linear Cryptanalysis

Linear cryptanalysis was proposed by Matsui [68] for Data Encryption Standard (DES)-like ciphers, but it is also effective for most iterated ciphers. This attack finds an equation that consists of XOR operations with plaintext and cipher text bits. If this equation exists and approximates to zero or one with a higher probability, then the attack can be successful.

A *linear trail* is a simulated trail that is a concatenation of some linear and non-linear components (e.g. S-box) through the cipher involved in the equation. The approximation of the equation is derived from the approximations of components in the linear trail. The bias of the equation is computed by the biases of the components

in the linear trail. We take each bit in the linear trail as a random variable, and the variable also has its own bias. If the bias of the variable is not equal to zero, then there is a linear trail reaching the bit with a probability of the bias plus one-half. When a linear trail extends by piling-up lemma, the bias decreases continually round by round; more linear trails are combined, or approximations of components are joined.

At the end of the extension we obtain the biases of the ciphertext bits and the equation can be derived by tracing the linear trail back to the plaintext bits. The largest bias of the equation means that fewer pairs are needed to mount the attack. We describe how the bias varies under each procedure and how to compute the largest bias as follows:

- Byte Substitution Procedure

In *ByteSub()*, the S-box combines input linear trails and those extended according to linear equations from its inputs. For each output bit, there are 256 equations according to the linear combination of eight input bits and the output bit. For example, $E = x_0 \oplus x_1 \oplus x_7 \oplus y_0$ where x_0 and x_7 denote eight input bits and y_0 denotes the highest output bit of the S-box. Because $y_0 = E \oplus x_0 \oplus x_1 \oplus x_7$, the bias of y_0 can be computed by the biases of E , x_0 , x_1 , and x_7 by piling-up lemma: $B_{y_0} = 2^3 \times B_E \times B_{x_0} \times B_{x_1} \times B_{x_7}$, where B_{y_0} , B_E , B_{x_0} , B_{x_1} , and B_{x_7} , denote the biases of variables y_0 , E , x_0 , x_1 , and x_7 .

- Transformation Byte Procedure

Because the *TransByte()* procedure performs only bit transposition, the biases of state bits do not change, but the proceeding direction of linear trails changes.

- Sub-Block XOR Procedure

Because the new left-half 64-bit data of the state are assigned from the previous right 64 bits in the *SubBlkXor()* procedure ($B'_i = B_{i+8}$, $0 \leq i \leq 7$), the biases of these bits are the same as those of previous corresponding last right 64 bits. However, for

the new right 64 bits of the state, each bit results from the XOR operation of three bits of three words. For example, the new bit $b'_{8,1}$ is computed by previous bits $b_{8,1}$, $b_{12,1}$ and $b_{0,1}$, where b_{ij} denotes the j^{th} bit of byte b_i , $i = 0 \cdots 15$ and $j = 0 \cdots 7$. For example, $b'_{8,1} = b_{8,1} \oplus b_{12,1} \oplus b_{0,1}$.

The bias of $b'_{8,1}$ is computed from the biases of $b_{8,1}$, $b_{12,1}$, and $b_{0,1}$ by the piling-up lemma: $Bias_{b'_{8,1}} = 2^2 \times Bias_{b_{8,1}} \times Bias_{b_{12,1}} \times Bias_{b_{0,1}}$. Different linear trails will be combined into one trail with a smaller bias by the XOR operation.

- Round Key Addition

The quantity of bias does not change after *AddRoundKey()*. Because the key bit is fixed at zero or one, the bias of the key bit is $\pm 1/2$. Therefore, only sign of the bias may change depending on the key value: $Bias_{out} = 2 \times Bias_{in} \times \left(\pm \frac{1}{2}\right) = \pm Bias_{in}$.

The largest bias of state bits at the end of n^{th} round can be computed with upper-bound biases at the end of the preceding two rounds. Differences between evaluation and real values are caused by the approximations of the S-boxes. The evaluation uses only the upper-bound bias of the linear equation of S-box 2^{-4} , but in the real case the computed bias is not necessarily the largest.

Our experiment computed the largest bias of all state bits at the end of each round as shown in Table 16. Since the proposed cipher is in Feistel structure, the bias increasing rate does not grow as steep as AES in square structure. There are no 4-round biases with a correlation above 2^{-208} . It is impossible to collect these needed pairs. Therefore, the proposed cipher is secure enough against linear attacks with more than five rounds.

Table 16 Largest bias of state bits at the end of each round (up to the tenth round)

Round #	AES_Plus	AES
1	2^{-4}	2^{-13}
2	2^{-13}	2^{-61}
3	2^{-34}	$\sim 2^{-253}$
4	2^{-85}	$\sim 2^{-1023}$
5	$\sim 2^{-208}$	$\sim 2^{-4100}$
6	$\sim 2^{-505}$	$\sim 2^{-16408}$
7	$\sim 2^{-1222}$	$\sim 2^{-65638}$
8	$\sim 2^{-2953}$	$\sim 2^{-262561}$
9	$\sim 2^{-7132}$	$\sim 2^{-1050253}$
10	$\sim 2^{-17221}$	$\sim 2^{-4201023}$

5.1.2 Differential Cryptanalysis

The differential cryptanalysis was first proposed by Biham and Shamir [75] for Data Encryption Standard (DES). It is also applicable for other iterated ciphers [6]. Differential cryptanalysis is used to derive a *differential trail* with high enough probability. A differential trail is derived from input differential bits to the cipher and the differential bits propagate through the cipher round by round. The probability of a differential trail is computed by multiplying *propagation ratios of differentials* for active S-boxes involved in the trail.

A differential trail is composed of difference patterns: $(x_0^*, x_1^*, \dots, x_r^*)$. The probability of this trail is the probability that an initial difference pattern x_0^* propagates to difference patterns $x_0^*, x_1^*, \dots, x_r^*$ after 1, 2, ..., r rounds, respectively. We describe how to compute the highest probabilities of differential trails for each procedure.

- Byte Substitution Procedure

The differential of *ByteSub()* depends on the propagation ratio of the input and output pair (called the input and output differential for the S-box). For a given input

differential x' , there are 256 tuples (x, y, x^*, y^*) such that $x' = x \oplus x^*$, $y = \text{ByteSub}(x)$, and $y^* = \text{ByteSub}(x^*)$. Therefore, the differential y' ($y' = y \oplus y^*$) distributes to 256 possible values and there is a probability for a differential x' , y' . We select the differential with the highest propagation ratio for a given input differential. We discuss the probability of differential trails in $\text{ByteSub}()$ in two parts:

- (1) Trails into the S-box: The S-box combines different trails into one (Figure 21). If T1, T2, and T3 denote three differential trails into the S-box, then the probabilities of the trails are P1, P2, and P3, respectively.

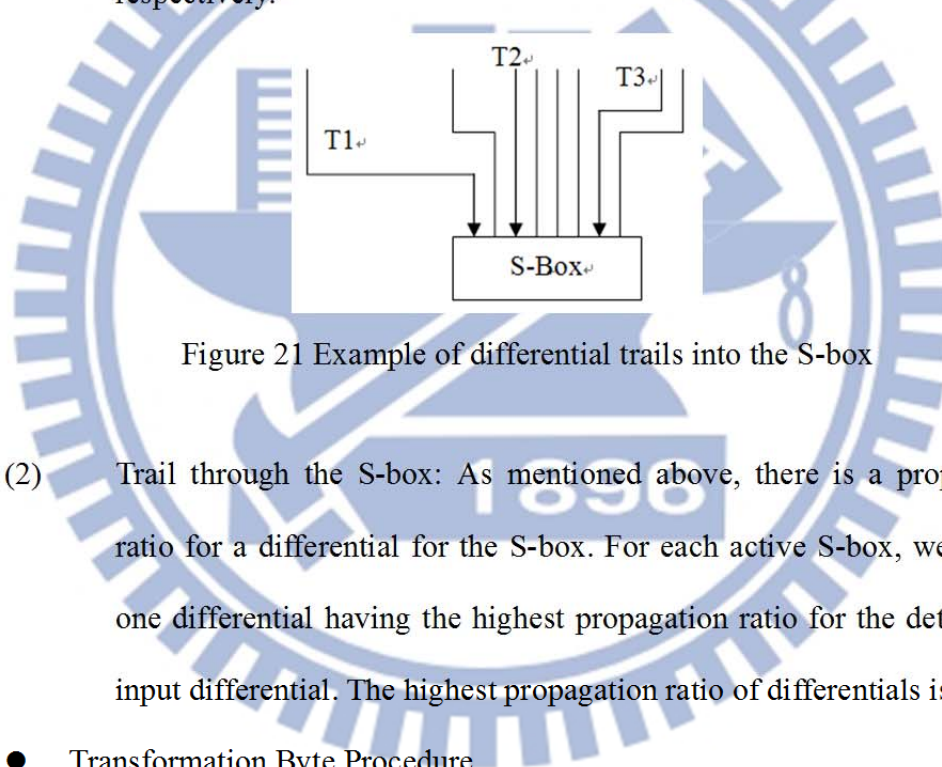


Figure 21 Example of differential trails into the S-box

- (2) Trail through the S-box: As mentioned above, there is a propagation ratio for a differential for the S-box. For each active S-box, we choose one differential having the highest propagation ratio for the determined input differential. The highest propagation ratio of differentials is 2^{-7} .

- Transformation Byte Procedure

In the $\text{TransByte}()$ procedure, the probabilities of the differential trails do not change, because there are no additional active S-boxes involved. This procedure just changes the extension direction of the differential trails.

- Sub-Block XOR Procedure

In the $\text{SubBlkXor}()$ operation, the left 64 bits of the state are delivered from the right 64 bits of the previous state, so the differential trails do not change. However, for

the new right 64 bits, each bit is XORed from three different bits of three words. If some input bit carries a differential trail, then the output bit will carry a differential trail joining all input differential trails (Figure 22). In the example, T1, T2, and T3 are three differential trails. The differential T5 is joined by T1, T2, and T3.

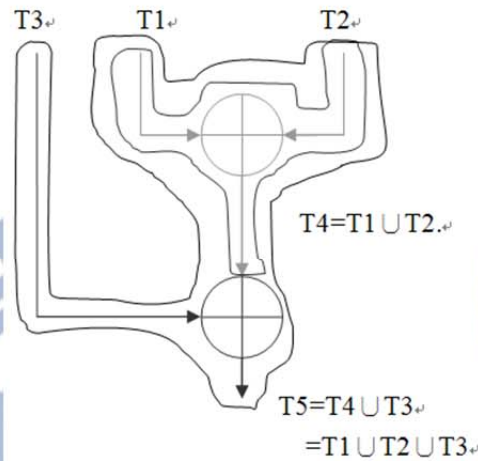


Figure 22 Example of joining of differential trails in XOR Operation

- Round Key Addition

AddRoundKey() does not change the input differential bits or the probability of the differential trails. If two bits a and a^* make the input differential bit $a' = a \oplus a^*$, then b and b^* are two outputs of *AddRoundKey()* (i.e. $b = a \oplus k$, $b^* = a^* \oplus k$). $b' = b \oplus b^* = (a \oplus k) \oplus (a^* \oplus k) = (a \oplus a^*) \oplus (k \oplus k) = a'$. This shows that *AddRoundKey()* does not change the input differential bits or the differential trails. Therefore, it does not change the probabilities of input differential trails.

The experiment results are given in Table 3, which shows the highest probability of 4080 differential trails reaching some round for the proposed cipher and AES. In Table 17, all values in gray are below 2^{-128} after the fourth round. This means that more than 2^{128} plaintext-ciphertext pairs are needed to perform the differential attack. In fact, it is impossible to collect so many pairs; therefore, our cipher can resist differential attacks for the given number of rounds.

Table 17 Highest probabilities of differential trails for two ciphers up to some rounds

Round #	The Proposed Cipher	AES
1	2^{-7}	2^{-7}
2	2^{-7}	2^{-14}
3	2^{-54}	2^{-72}
4	$\sim 2^{-894}$	$\sim 2^{-300}$
5	$\sim 2^{-10746}$	$\sim 2^{-1248}$
6	$\sim 2^{-143418}$	$\sim 2^{-5010}$
7	$\sim 2^{-1835142}$	$\sim 2^{-20064}$
8	$\sim 2^{-23859942}$	$\sim 2^{-80280}$
9	$\sim 2^{-308328672}$	$\sim 2^{-321144}$
10	$\sim 2^{-3993665760}$	$\sim 2^{-1284600}$

5.1.3 Square Attack

The square attack is a dedicated attack for the cipher square [7] that exploits the byte-oriented structure of the cipher. AES inherits the same property from the square cipher; thus a four-round AES is threatened by the square attack [8]. Because our proposed cipher uses the *TransByte()* operation, which breaks the property exploited in a square attack, the attack cannot be successfully carried out. We define a Λ -set as a set of 256 states that differ in active bytes and are equal in passive bytes:

$$\forall x, y \in \Lambda : \begin{cases} x_{i,j} \neq y_{i,j}, & \text{if } (i,j) \text{ is active} \\ x_{i,j} = y_{i,j}, & \text{others} \end{cases} \quad \text{Eq 49}$$

The Λ -set property causes all bytes to be balanced:

$$\bigoplus_{a \in \Lambda} a_{i,j} = 0. \quad \text{Eq 50}$$

In our proposed cipher, the *AddRoundKey()* and *ByteSub()* procedures do not influence the Λ -set with the position of active bytes indicated as K_j where $K \in A, \dots, D$ and $j = \{l, r\}$ means the Λ -set at a byte of left 64 bits (Λ_l) or right 64 bits (Λ_r) and its substitution through the S-box in each round (ex: $A \rightarrow B \rightarrow C \rightarrow D$). As shown in

Figure 23, it separates the active byte from one block into eight blocks in the first round. The *TransByte()* procedure breaks the Λ -set property but remains balanced in the first round. We discuss the security after the *SubBlkXor()* procedure. M_j^i where $j = \{l, r\}$ indicates the active bytes mixed i times, resulting in the state being unbalanced. Regarding the influence of Λ_l , *SubBlkXor()* moves the influence of Λ_l to the two right columns in the first round and makes them unbalanced in the second round as illustrated in Figure 24. After the third round, none of the states are balanced. Thus, it is difficult to recover any input using a square attack after the third round as shown in Figure 25.

Similar to Λ_l , the right 64 bits (Λ_r) are not balanced after *SubBlkXor()* in the first and second rounds. Thus, it resists the square attack after the second round as shown in Figure 26, Figure 27 and Figure 28. Our cipher improves on the AES, which requires four rounds.

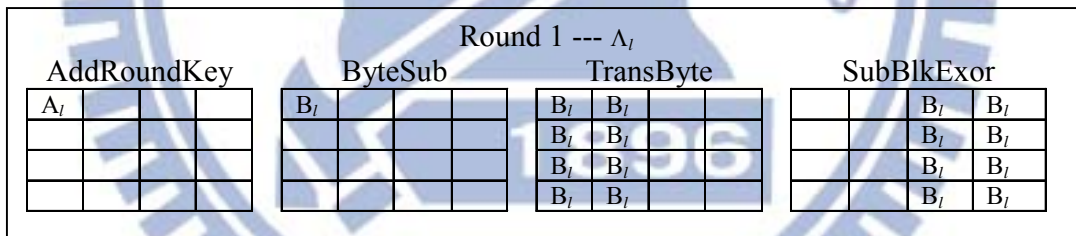


Figure 23 The influence of active byte Λ_l in 1st round

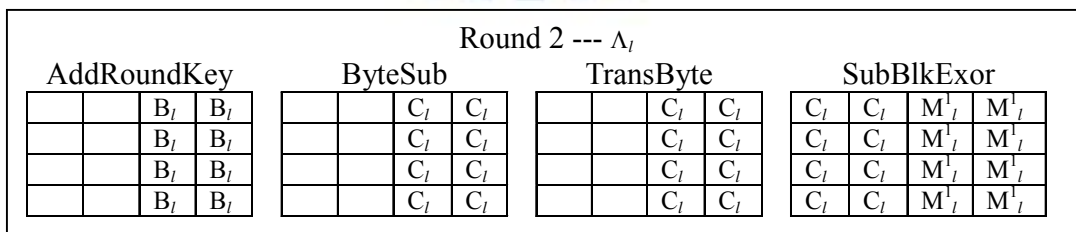


Figure 24 The influence of active byte Λ_l in 2nd round

Round 3 --- Λ_l															
AddRoundKey				ByteSub				TransByte				SubBlkExor			
C_l	C_l	M_l^1	M_l^1	D_l	D_l	M_l^1	M_l^1	D_l	D_l	M_l^1	M_l^1	M_l^1	M_l^1	M_l^2	M_l^2
C_l	C_l	M_l^1	M_l^1	D_l	D_l	M_l^1	M_l^1	D_l	D_l	M_l^1	M_l^1	M_l^1	M_l^1	M_l^2	M_l^2
C_l	C_l	M_l^1	M_l^1	D_l	D_l	M_l^1	M_l^1	D_l	D_l	M_l^1	M_l^1	M_l^1	M_l^1	M_l^2	M_l^2
C_l	C_l	M_l^1	M_l^1	D_l	D_l	M_l^1	M_l^1	D_l	D_l	M_l^1	M_l^1	M_l^1	M_l^1	M_l^2	M_l^2

Figure 25 The influence of active byte Λ_l in 3rd round

Round 1 --- Λ_r															
AddRoundKey				ByteSub				TransByte				SubBlkExor			
		A_r				B_r				B_r	B_r	B_r	B_r	M_r^1	M_r^1
										B_r	B_r	B_r	B_r	M_r^1	M_r^1
										B_r	B_r	B_r	B_r	M_r^1	M_r^1
										B_r	B_r	B_r	B_r	M_r^1	M_r^1

Figure 26 The influence of active byte Λ_r in 1st round

Round 2 --- Λ_r															
AddRoundKey				ByteSub				TransByte				SubBlkExor			
B_r	B_r	M_r^1	M_r^1	C_r	C_r	M_r^1	M_r^1	C_r	C_r	M_r^1	M_r^1	M_r^1	M_r^1	M_r^2	M_r^2
B_r	B_r	M_r^1	M_r^1	C_r	C_r	M_r^1	M_r^1	C_r	C_r	M_r^1	M_r^1	M_r^1	M_r^1	M_r^2	M_r^2
B_r	B_r	M_r^1	M_r^1	C_r	C_r	M_r^1	M_r^1	C_r	C_r	M_r^1	M_r^1	M_r^1	M_r^1	M_r^2	M_r^2
B_r	B_r	M_r^1	M_r^1	C_r	C_r	M_r^1	M_r^1	C_r	C_r	M_r^1	M_r^1	M_r^1	M_r^1	M_r^2	M_r^2

Figure 27 The influence of active byte Λ_r in 2nd round

Round 3 --- Λ_r															
AddRoundKey				ByteSub				TransByte				SubBlkExor			
M_r^1	M_r^1	M_r^2	M_r^2	M_r^1	M_r^1	M_r^2	M_r^2	M_r^1	M_r^1	M_r^2	M_r^2	M_r^2	M_r^2	M_r^3	M_r^3
M_r^1	M_r^1	M_r^2	M_r^2	M_r^1	M_r^1	M_r^2	M_r^2	M_r^1	M_r^1	M_r^2	M_r^2	M_r^2	M_r^2	M_r^3	M_r^3
M_r^1	M_r^1	M_r^2	M_r^2	M_r^1	M_r^1	M_r^2	M_r^2	M_r^1	M_r^1	M_r^2	M_r^2	M_r^2	M_r^2	M_r^3	M_r^3
M_r^1	M_r^1	M_r^2	M_r^2	M_r^1	M_r^1	M_r^2	M_r^2	M_r^1	M_r^1	M_r^2	M_r^2	M_r^2	M_r^2	M_r^3	M_r^3

Figure 28 The influence of active byte Λ_r in 3rd round

5.2 Experiment results of SHA-256-XOR

Table 18 lists 10 generations of the simulation results for $\{A, B, C, D\}$. The simulation requires heavy computational times for each t . We have not generated optimum parameters for additional rounds because of the computational requirements. However, we believe that we have demonstrated the basis of our contribution, which is a possible approach for the selection of optimal message scheduling parameters and the analysis of the security fitness.

The values for the 5 variables converge after 42 generations. It appears that the approximate optimal values are $\{A, B, C, D\} = \{4, 1, 1, 16\}$. Thus, the best equation for W_t of SHA-256-XOR, named optSHA-256-XOR, should be

$$W_t = \begin{cases} M_t^{(i)}, & 0 \leq t \leq 15 \\ \sigma_0^{\{256\}}(W_{t-4}) \oplus W_{t-1} \oplus \sigma_1^{\{256\}}(W_{t-1}) \oplus W_{t-16}, & 16 \leq t \leq 63 \end{cases} \quad \text{Eq 51}$$

Figure 29 compares SHA-256-XOR with optSHA-256-XOR by showing clearly that optSHA-256-XOR is indeed more secure than SHA-256.

Table 18 The last 10 generations of the simulation

Generation	A	B	C	D	Fitness
41	8	1	1	16	238
42	4	1	1	16	259
43	4	1	1	16	265
44	4	1	1	16	265
45	4	1	1	16	265
46	4	1	1	16	270
47	4	1	1	16	270
48	4	1	1	16	270
49	4	1	1	16	270
50	4	1	1	16	270

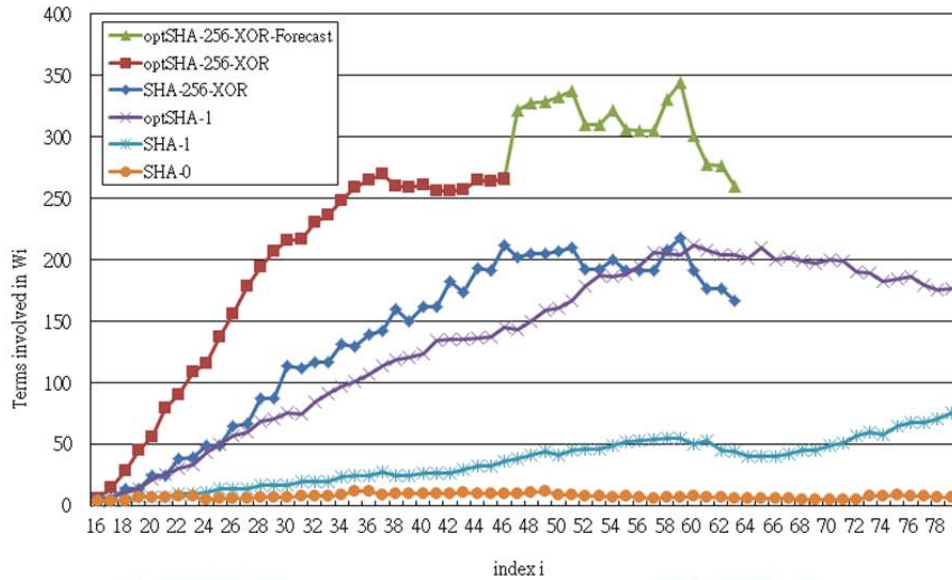


Figure 29 Comparison of the number of terms involved in each W_t in message scheduling for SHA-256-XOR and optSHA-256-XOR

With regards to the performance, Figure 30 compares the running time for each W_t ($16 \leq t \leq 30$) in brute force and genetic algorithms. We have not yet been able to complete the simulation for every W_t . The later items in the experiment will consume additional time because the equation is a recursive function.

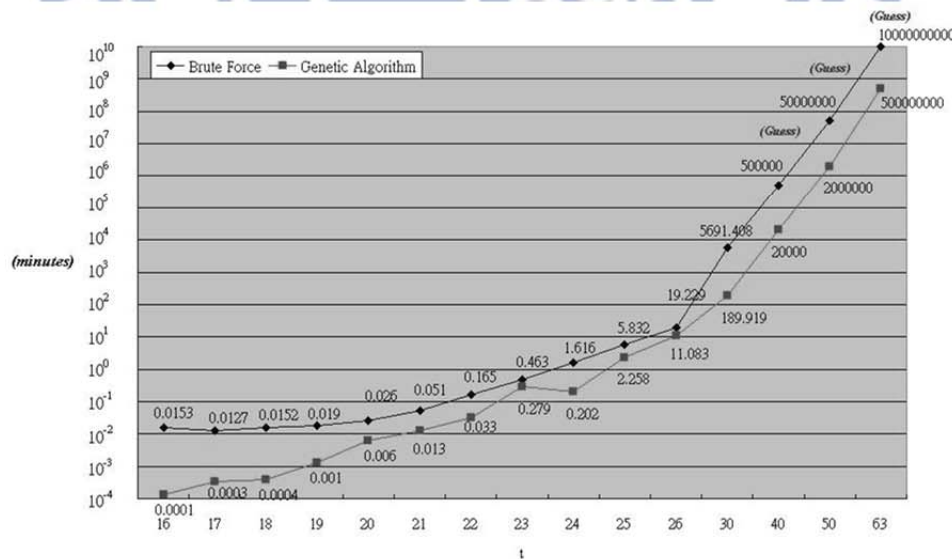


Figure 30 Comparison of the running time in W_t between genetic algorithm and brute force

5.3 Performance analysis of Modified Autonomous Key Management

This section discusses the performance improvement of the proposed method in terms of communication cost and computation cost. The modified AKM inherits the AKM structure, and transmissions between each node are (update) shares. Thus, the single message discussion must be transmitted with significant information.

The length of secret key k , protected by the secret sharing scheme, must be long enough for some security issues (i.e., 2 048 bits or more). In Shamir's secret sharing scheme, k is constant in the $a(x)$ equation. The length of all the shares $a(x_i) = \sum_{j=1}^{t-1} a_j x^j + k$, $1 \leq i \leq n$, is bounded by $|k|$. For example, if $|k| = 2048$ bits long, the length of each share is at least 2048 bits. However, the modified secret sharing scheme reduces share length to $\frac{1}{t}$ without security loss. The secret key is divided in each coefficient $a_j = r_k | k_j$, and $k = k_1 k_2 \dots k_t$ with the length $|a(x_i)|$ as $\frac{1}{t}$ of $|k|$ on appropriate prime number p . Therefore, the modified MANET communication cost can be reduced to $\frac{1}{t}$.

Table 19 Message length comparison

	Message (share) length size
AKM	$ y_i = k \leq p $
Modified AKM	$ y_i = \frac{ k }{t} \leq k \leq p $

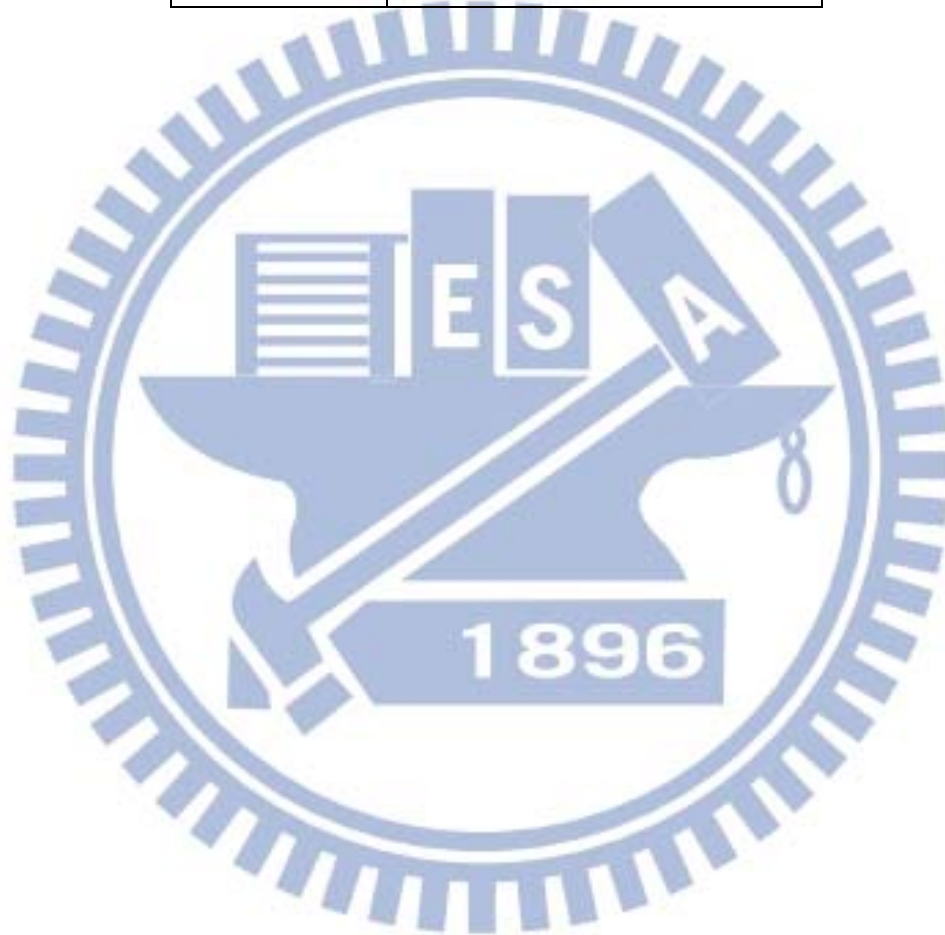
Computation cost on the MANET environment is a very important issue. Certain mobile ad-hoc devices have restricted power, and cannot support jobs requiring heavy computation cost. The proposed improvement also influences computation cost. Finding that the critical mathematical operation is module multiplication (/division) in all operations is easy, depending on operand length. Almost all operands in modified

AKM reduce, resulting from each modified AKM share as $\frac{1}{t}$ faster than AKM.

Furthermore, the computation cost of all operations can be reduced to $\frac{1}{t}$.

Table 20 Operand length comparison

	operand length size
AKM	$ y_i = k \leq p $
Modified AKM	$ y_i = \frac{ k }{t} \leq k \leq p $



6. Conclusions

While IT has evolved from people-to-machine (Web 1.0) through people-to-people (Web 2.0) to machine-to-machine (Web 3.0), working styles have gradually changed from writing on paper to Cloud storage. Digital content development is critical to these changes. However, information security and privacy issues should also be addressed. Therefore, we proposed improvements to symmetric ciphers, one-way hash functions, and secure protocols.

The advanced encryption standard (AES) is applied as an encryption standard to replace data encryption standard (DES) and triple-DES in fields including e-commerce, embedded systems, and ubiquitous computing. Originally, AES performed matrix operations using the *MixColumns()* procedure, resulting in more complicated computations and increasingly complex software and hardware designs. The proposed AES variant replaces the matrix with an XOR operation providing stronger security.

The proposed cipher's advantages are

- The security of round transformation in the proposed cipher is made stronger than AES by strengthening the resistance of the square attack from 4 to 3 rounds.
- Most operations in the proposed cipher, including the *TransByte()* procedure, can be used for both encryption and decryption.

In future, we will focus on speeding up the cipher, especially on *TransByte()* and *SubBlkXor()*.

Since 1993, the secure hash function family is an important standard in cryptography. We propose a novel view of complexity (and hence security fitness) by counting the number of terms involved in each equation, instead of analyzing the

probability of finding collisions within hash functions. We identified the near optimal versions, optSHA-1 and optSHA-256-XOR, using brute force and genetic approaches of SHA-1 and SHA-256-XOR, respectively; the latter had more computational efficiency. This analysis is useful for designers interested in the security of modular-addition-free hash functions suitable for hardware implementation with lower gate counts. The obtained message schedule parameter sets will be a good reference for further improvements of secure hash algorithm (SHA) functions.

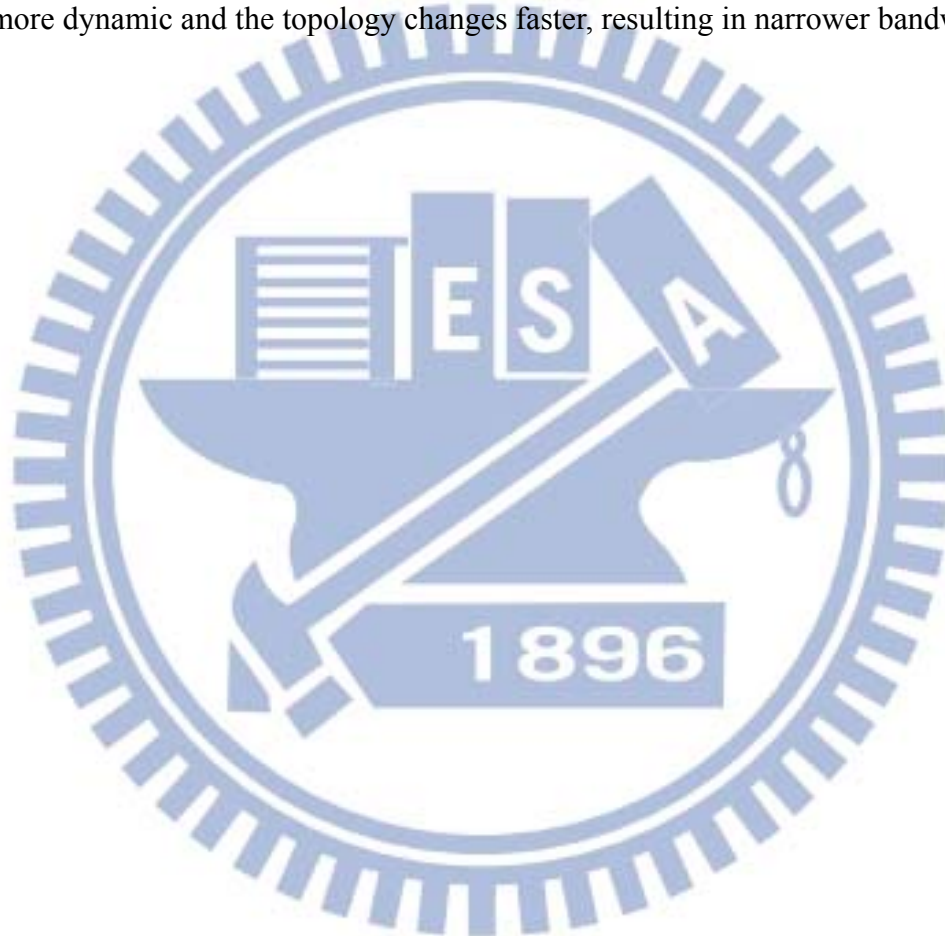
The proposed generalized SHA (SHA- mn) uses arbitrary length messages as inputs for generating message digests with required lengths. We modified each SHA- mn step as a generalized version containing padding and parsing; setting the initial hash values, constants, Boolean expressions, functions, and message schedule; initializing the eight working variables and for-loop operation; and computing the i^{th} intermediate hash values. Furthermore, we solved the LHV problem, which does not exist in the original SHA family standard.

For security purposes, SHA- mn was generalized based on SHA family design rules. While the design was improved, there is disagreement regarding the method used to calculate the complexity according to the birthday paradox, as the collision of full SHA-1 was reported in 2005. Many studies focus on developing efficient ways for finding SHA-256/512 collisions. We therefore believe that the approximate complexity of SHA- mn under the birthday attack is $2^{mn/2}$.

The security of mobile ad hoc networks influences their applications. To achieve adequate security, autonomous key management (AKM) for numerous nodes is important. We propose modified AKM to reduce communications and computation costs to $\frac{1}{t}$ of the original values without compromising security. Results show that modified AKM is more practical because it can handle large numbers of dynamic

nodes in a MANET, while maintaining adequate security requirements. The proposed methodology is applicable to all schemes based on cryptographic threshold schemes for truncating message size without endangering security.

Further research will attempt to simplify the computation complexity of AKM operations for the workability of ad hoc devices. Furthermore, we will apply the proposed concept to vehicular ad hoc networks (VANETs) because their environments are more dynamic and the topology changes faster, resulting in narrower bandwidth.



References

- [1] H. Sundmaeker, P. Guillemin, P. Friess and S. Woelfflé, "Vision and Challenges for Realising the Internet of Things," The Cluster of European Research projects on the Internet of Things (CERP-IoT), 2010.
- [2] M. Luo, "Research on the Knowledge Management Based on GIS," in *National Conference on Information Technology and Computer Science (CITCS)*, 2012.
- [3] B. Daskala, "Flying 2.0 Enabling automated air travel by identifying and addressing the challenges of IoT & RFID technology," European Network and Information Security Agency (ENISA), 2010.
- [4] J. Daemen and V. Rijmen, Advanced Encryption Standard (AES), Federal Information Processing Standards Publications (FIPS PUBS) 197, NIST, 2001.
- [5] E. Biham¹, A. Biryukov and A. Shamir, "Cryptanalysis of Skipjack Reduced to 31 Rounds Using," in *Proceedings of Eurocrypt, LNCS 1592*, 1999.
- [6] E. Biham and N. Keller, "Cryptanalysis of reduced variants of Rijndael," in *Proceedings of 3rd AES Conference*, 2000.
- [7] J. Daemen, L. Knudsen and V. Rijmen, "The block cipher Square," in *Proceedings of the 4th International Workshop on Fast Software Encryption (FSE), LNCS 1267*, 1997.
- [8] J. Daemen and V. Rijmen, The Design of Rijndael, AES - The Advanced Encryption Standard, Springer, 2002.
- [9] Eli Biham, Orr Dunkelman and Nathan Keller, "Related-Key Boomerang and Rectangle Attacks," in *Proceedings of Eurocrypt, LNCS 3557*, 2005.
- [10] A. Biryukov, "The Boomerang Attack on 5 and 6-round AES," in *Proceedings of Advanced Encryption Standard 4, LNCS 3373*, 2005.
- [11] D. R. Stinson, Cryptography Theory and Practice, 3 ed., CRC Press, 2005.
- [12] J. H. Cheon, M. Kim, K. Kim, J. Y. Lee and S. Kang, "Improved Impossible Differential Cryptanalysis of Rijndael and Crypton," in *Proceedings of The 4th International Conference on Information Security and Cryptology (ICISC), LNCS 2288*, 2002.
- [13] R. C. Phan, "Impossible Differential Cryptanalysis of 7-round Advanced Encryption Standard (AES)," *Information Processing Letters*, vol. 91, no. 1, pp. 33-38, 2004.

- [14] C. Jie, W. Yongzhuang and H. Yupu, "A New Method for Impossible Differential Cryptanalysis of 7-round Advanced Encryption Standard," in *Proceedings of Communications, Circuits and Systems (ICCCAS)*, 2006.
- [15] W. Zhang, W. Wu and D. Feng, "New Results on Impossible Differential Cryptanalysis of Reduced AES," in *Proceedings of 10th International Conference on Information Security and Cryptology (ICISC)*, LNCS 4817, 2007.
- [16] E. Biham, O. Dunkelman and N. Keller, "Related-Key Impossible Differential Attacks on 8-Round AES-192," in *Proceedings of RSA Conference*, LNCS 3860, 2006.
- [17] J. Liu, B. Wei, X. Cheng and X. Wang, "An AES S-box to Increase Complexity and Cryptographic Analysis," in *Proceedings of the 19th International Conference on Advanced Information Networking and Applications (AINA)*, 2005.
- [18] A. Grochowska-Czurylo and J. Stoklosa, "Random Generation of S-Boxes for Block Ciphers," in *Biometrics, Computer Security Systems and Artificial Intelligence Applications*, Springer, 2006, pp. 121-135.
- [19] D. Bhattacharya, N. Bansal and A. Banerjee, "A Near Optimal S-Box Design," in *Proceedings of the 3rd International Conference on Information Systems Security (ICISS)*, LNCS 4812, 2007.
- [20] L. Cui and Y. Cao, "A new s-box structure named Affine-power-affine," *International Journal of Innovative Computing, Information and Control (IJICIC)*, vol. 3, no. 3, pp. 751-759, 2007.
- [21] National Institute of Standards and Technology, Secure hash standard, Federal Information Processing Standards Publications (FIPS PUBS) 180-4, 2012.
- [22] P. Pal and P. Sarkar, "PARSHA-256 – A New Parallelizable Hash Function and a Multithreaded Implementation," in *Proceedings of 10th International Workshop on Fast Software Encryption 2003 (FSE)*, LNCS 2887, 2003.
- [23] H. Handschuh and D. Naccache, "SHACAL," in *Proceedings of 1st Open NESSIE Workshop*, 2001.
- [24] J. Lu, J. Kim, N. Keller and O. Dunkelman, "Related-Key Rectangle Attack on 42-Round SHACAL-2," in *Proceedings of 9th Information Security Conference (ISC)*, LNCS 4176, 2006.
- [25] H. Yoshida and A. Biryukov, "Analysis of a SHA-256 Variant," in *Proceedings of 12th Annual Workshop on Selected Areas in Cryptography (SAC)*, LNCS 3897, 2005.
- [26] B. Schneier, *Applied Cryptography*, 4 ed., John Wiley & Sons, 1996.

- [27] F. Chabaud and A. Joux, "Differential Collisions in SHA-0," in *Proceedings of Crypto*, 1998.
- [28] V. Rijmen and E. Oswald, "Update on SHA-1," in *Proceedings of CT-RSA, LNCS 3376*, 2005.
- [29] X. Wang, H. Yu and Y. L. Yin, "Finding Collision in the Full SHA-1," in *Proceedings of Crypto, LNCS 3621*, 2005.
- [30] E. A. Grechnikov, "Collisions for 72-step and 73-step SHA-1: Improvements in the Method of Characteristics," Cryptology ePrint Archive, Report 2010/413, 2010.
- [31] H. Gilbert and H. Handschuh, "Security Analysis of SHA-256 and Sisters," in *Proceedings of 6th Information Security Conference (SAC), LNCS 3006*, 2003.
- [32] F. Mendel, N. Pramstaller, C. Rechberger and V. Rijmen, "Analysis of Step-Reduced SHA-256," in *Proceedings of 13th annual Fast Software Encryption workshop (FSE), LNCS 4047*, 2006.
- [33] I. Nikolić and A. Biryukov, "Collisions for Step-Reduced SHA-256," in *Proceedings of 15th International Workshop on Fast Software Encryption (FSE), LNCS 5086*, 2008.
- [34] S. K. Sanadhya and P. Sarkar, "New Collision Attacks against Up to 24-Step SHA-2," in *Proceedings of Indocrypt, LNCS 5365*, 2008.
- [35] S. Indesteege, F. Mendel, B. Preneel and C. Rechberger, "Collisions and Other Non-random Properties for Step-Reduced SHA-256," in *Proceedings of Selected Areas in Cryptography (SAC), LNCS 5381*, 2009.
- [36] K. Aoki, J. Guo, K. Matusiewicz, Y. Sasaki and L. Wang, "Preimages for Step-Reduced SHA-2," in *Proceedings of Asiacrypt*, 2009.
- [37] F. Mendel, T. Nad and M. Schl affer, "Finding SHA-2 Characteristics: Searching through a Minefield of Contradictions," in *Proceedings of Asiacrypt, LNCS 7073*, 2011.
- [38] A. Biryukov, M. Lamberger, F. Mendel and I. Nikolić, "Second-Order Differential Collisions for Reduced SHA-256," in *Proceedings of ASIACRYPT, LNCS 7073*, 2011.
- [39] A. Khalili, J. Katz and W. Arbaugh, "Toward Secure Key Distribution in Truly Ad Hoc Networks," in *Proceedings of the International Symposium on Applications and the Internet*, 2003.
- [40] B. Lehane, L. Doyle and D. O'Mahony, "Shared RSA key generation in a mobile ad hoc network," in *Proceedings of the IEEE Military Communications Conference (MILCOM)*, 2003.

- [41] H. Luo, J. Kong, P. Zerfos, S. Lu and L. Zhang, "Self-Securing Ad Hoc Wireless Networks," in *Proceedings of the 7th IEEE Symposium on Computers and Communications (ISCC)*, 2002.
- [42] H. Luo, J. Kong, P. Zerfos, S. Lu and L. Zhang, "URSA: Ubiquitous and Robust Access Control for Mobile Ad Hoc Networks," *IEEE/ACM Transactions on Networking*, vol. 12, no. 6, pp. 1049-1063, 2004.
- [43] J. Kong, P. Zerfos, H. Luo, S. Lu and L. Zhang, "Providing Robust and Ubiquitous Security Support for Mobile Ad Hoc Networks," in *Proceedings of the IEEE 9th International Conference on Network Protocols (ICNP)*, 2001.
- [44] L. Zhou and Z. J. Haas, "Securing Ad Hoc Networks," *IEEE Network on Network Security*, vol. 13, no. 6, pp. 24-30, 1999.
- [45] S. Capkun, L. Butty'an and J. P. Hubaux, "Self-Organized Public-Key Management for Mobile Ad Hoc Networks," Technical Report 2002/34, EPFL/IC, 2002.
- [46] M. Omar, Y. Challal and A. Bouabdallah, "Reliable and Fully Distributed Trust Model for Mobile Ad Hoc Networks," *Computers & Security*, vol. 28, pp. 199-214, 2009.
- [47] Y. Park, Y. Park and S. Moon, "ID-based Private Key Update Protocol with Anonymity for Mobile Ad-Hoc Networks," in *Proceedings of 2010 International Conference of Computational Science and its Applications*, 2010.
- [48] K. Hamouid and K. Adi, "Secure and Robust Threshold Key Management (SRKM) Scheme for Ad Hoc Networks," *Security and communication networks*, vol. 3, pp. 517-534, 2010.
- [49] L. Li and R. S. Liu, "Securing Cluster-Based Ad Hoc Networks with Distributed Authorities," *IEEE Transactions on Wireless Communications*, vol. 9, no. 10, pp. 3072-3081, 2010.
- [50] D. Saravanan, D. Rajalakshmi and D. Maheswari, "DYCRASEN: A Dynamic Cryptographic Asymmetric Key Management for Sensor Network using Hash Function," *International Journal of Computer Applications*, vol. 18, no. 8, pp. 1-3, 2011.
- [51] H. Yang, H. Luo, F. Ye, S. Lu and L. Zhang, "Security in Mobile Ad Hoc Networks Challenges and Solutions," *IEEE Wireless Communications*, vol. 11, no. 1, pp. 38-47, 2004.
- [52] B. Zhu, F. Bao, R. H. Deng, M. S. Kankanhalli and G. Wang, "Efficient and Robust Key Management for Large Mobile Ad Hoc Networks," *Computer networks*, vol. 48, pp. 657-682, 2005.

- [53] A. Shamir, "How to Share a Secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612-613, 1979.
- [54] Y. Desmedt, "Threshold Cryptography," *European Transactions on Telecommunications*, vol. 5, no. 4, pp. 449-457, 1944.
- [55] Y. Desmedt and Y. Frankel, "Threshold Cryptosystems," in *Proceedings of Crypto, LNCS 0435*, 1990.
- [56] R. Gennaro, S. Jarecki, H. Krawczyk and T. Rabin, "Secure Distributed Key Generation for Discrete-Log Based Cryptosystem," in *Proceedings of Eurocrypt, LNCS 1592*, 1999.
- [57] W. J. Tsaur and H. T. Pai, "Dynamic Key Management Schemes for Secure Group Communication Based on Hierarchical Clustering in Mobile Ad Hoc Networks," in *Proceedings of International Workshops on Frontiers of High Performance Computing and Networking (ISPA), LNCS 4743*, 2007.
- [58] K. Lauter, "The Advantages of Elliptic Curve Cryptography for Wireless Security," *IEEE Wireless Communications*, vol. 11, no. 1, pp. 62-67, 2004.
- [59] M. Girault, "Self-Certified Public Keys," in *Proceedings of Eurocrypt, LNCS 547*, 1991.
- [60] S. Bakhtiari, R. Safavi-Naini and J. Pieprzyk, "Cryptographic Hash Functions: A Survey," Technical Report 95-09, University of Wollongong, 1995.
- [61] National Institute of Standards and Technology, Secure hash standard, Federal Information Processing Standards Publications (FIPS PUBS) 180, 1993.
- [62] National Institute of Standards and Technology, Secure hash standard, Federal Information Processing Standards Publications (FIPS PUBS) 180-1, 1995.
- [63] National Institute of Standards and Technology, Secure hash standard, Federal Information Processing Standards Publications (FIPS PUBS) 180-2, NIST, 2002.
- [64] National Institute of Standards and Technology, Secure hash standard, Federal Information Processing Standards Publications (FIPS PUBS) 180-3, 2008.
- [65] J. H. Holland, *Adaptation in Natural and Artificial System*, The University of Michigan Press, 1975.
- [66] S. Adcock, "Genetic Algorithm Utility Library," [Online]. Available: <http://gaul.sourceforge.net>.
- [67] Y. S. Yeh, C. Y. Lee, T. Y. Huang and C. H. Lin, "A Transpositional Advanced Encryption Standard (AES) Resists 3-Round Square Attack," *International Journal of Innovative Computing, Information and Control*, vol. 5, no. 5, pp. 1349-4198, 2009.
- [68] M. Matsui, "Linear cryptanalysis method for DES cipher," in *Proceedings of*

Eurocrypt, LNCS 765, 1994.

- [69] C. Y. Lee, C. H. Lin, D. J. Chen and Y. S. Yeh, "Generalized Secure Hash Algorithm: SHA-X," *International Journal of Advancements in Computing Technology(IJACT)*, vol. 4, no. 7, pp. 41-52, 2012.
- [70] C. H. Lin, C. Y. Lee, K. M. Kavi, D. J. Chen and Y. S. Yeh, "An evaluation criterion and an approach to improve the security fitness of SHA-256 via genetic algorithm," *Journal of Information Science and Engineering*, (to appear).
- [71] X. Wang, H. Yu and Y. L. Yin, "Efficient Collision Search Attacks on SHA-0," in *Proceedings of Crypto, LNCS 3621, 2005.*
- [72] K. Matusiewicz, J. Pieprzyk, N. Pramstaller, C. Rechberger and V. Rijmen, "Analysis of Simplified Variants of SHA-256," in *Proceedings of Western Europe - an Workshop on Research in Cryptology (WEWoRC), 2005.*
- [73] Y. S. Yeh, T. Y. Huang, I T. Chen and S. C. Chou, "Analyze SHA-1 in Message Schedule," *Journal of Discrete Mathematical Sciences & Cryptography*, vol. 10, no. 1, pp. 1-7, 2007.
- [74] C. H. Lin, C. Y. Lee and D. J. Chen, "Modified automous key management scheme with reduced communication/computation costs in MANET," *Computing and Informatics*, vol. 30, pp. 1167-1180, 2011.
- [75] E. Biham and A. Shamir, *Differential Cryptanalysis of the Data Encryption Standard*, Springer, 1993.
- [76] C. H. Lin, C. Y. Lee, S. Y. Lu and S. P. Chien, "Unseen Visible Watermarking for Gray Level Images Based on Gamma Correction," in *Proceedings of International Conference on Future-Generation Communication and Networking (FGCN), CCIS 265, 2011.*
- [77] C. H. Lin, C. Y. Lee, T. C. Yang and S. P. Lai, "Visible Watermarking Based on Multi-parameters Adjustable Gamma Correction," in *Proceedings of International Conference on Future-Generation Communication and Networking (FGCN), CCIS 265, 2011.*