

**STUDIES, DESIGN AND DEVELOPMENT OF
NETWORK SECURITY ENHANCEMENT SERVICES
USING NOVEL CRYPTOGRAPHIC ALGORITHMS**

**A thesis submitted
by**

SHEENA MATHEW

for the award of the degree of

DOCTOR OF PHILOSOPHY

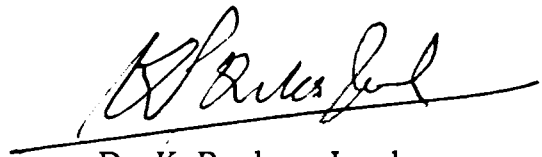
UNDER THE FACULTY OF TECHNOLOGY

**DEPARTMENT OF COMPUTER SCIENCE
COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY
KOCHI, KERALA – 682 022**

June 2008

CERTIFICATE

This is to certify that the thesis titled "STUDIES, DESIGN AND DEVELOPMENT OF NETWORK SECURITY ENHANCEMENT SERVICES USING NOVEL CRYPTOGRAPHIC ALGORITHMS" is a report of the original work carried out by Ms. Sheena Mathew, under my supervision and guidance in the Department of Computer Science, Cochin University of Science and Technology, Kochi. The work presented in this thesis has not been submitted for any other degree from any other university.



Dr. K. Poullose Jacob

(Supervising Guide)

Professor and Head

Department of Computer Science

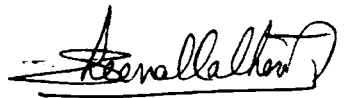
Cochin University of Science and Technology

Kochi - 682 022

16th June 2008

DECLARATION

I hereby declare that the work presented in this thesis is based on the original work carried out by me under the supervision of Dr. K. Poulouse Jacob, Professor and Head, Department of Computer Science, Cochin University of Science and Technology, Kochi. The work presented in this thesis has not been submitted for any other degree from any other University.

A handwritten signature in black ink, appearing to read 'Sheena Mathew', written over a horizontal line.

SHEENA MATHEW

Kochi - 682 022
16th June 2008

ACKNOWLEDGEMENTS

While submitting this thesis report with a profound sense of gratitude, I would avail this opportunity to thank those who have helped me generously in the completion of this work.

First and foremost, I am grateful to the God Almighty for having been my lodestar throughout my life.

I would express my deep and sincere gratitude to my supervisor and guide, Professor K. Poullose Jacob, Director, School of Computer Science Studies, Cochin University of Science and Technology for the guidance he has given me throughout the course of my research work. His knowledge, logical thinking, invaluable comments, caring and supportive attitude, patience, etc. were the main driving forces of my work.

I am also indebted to Prof. Veni Madhavan, Indian Institute of Science, Bangalore, whose fundamental approach to research in Cryptography was an inspiration to me. The initial idea for developing the algorithm for providing confidentiality has been the outcome of my dissertation for M.E. guided by him at the IISc.

I would also like to mention Dr. David Peter S. and all other faculty and staff members of the department and my friends Ancy, Damodaran, Latha, Pramod, Preetha, Ranjit, Rekha, Sara, Shahana, Sheena, Sudheep, Supriya, Surekha and Vinod, who have been my well wishers and sources of inspiration.

SHEENA MATHEW

ABSTRACT

Internet today has become a vital part of day to day life, owing to the revolutionary changes it has brought about in various fields. Dependence on the Internet as an information highway and knowledge bank is exponentially increasing so that a going back is beyond imagination. Transfer of critical information is also being carried out through the Internet. This widespread use of the Internet coupled with the tremendous growth in e-commerce and m-commerce has created a vital need for information security.

Internet has also become an active field of crackers and intruders. The whole development in this area can become null and void if fool-proof security of the data is not ensured without a chance of being adulterated. It is, hence a challenge before the professional community to develop systems to ensure security of the data sent through the Internet.

Stream ciphers, hash functions and message authentication codes play vital roles in providing security services like confidentiality, integrity and authentication of the data sent through the Internet. There are several such popular and dependable techniques, which have been in use widely, for quite a long time. This long term exposure makes them vulnerable to successful or near successful attempts for attacks. Hence it is the need of the hour to develop new algorithms with better security.

Hence studies were conducted on various types of algorithms being used in this area. Focus was given to identify the properties imparting security at this stage. By making use of a perception derived from these studies, new

algorithms were designed. Performances of these algorithms were then studied followed by necessary modifications to yield an improved system consisting of a new stream cipher algorithm MAJE4, a new hash code JERIM-320 and a new message authentication code MACJER-320. Detailed analysis and comparison with the existing popular schemes were also carried out to establish the security levels.

The Secure Socket Layer (SSL) / Transport Layer Security (TLS) protocol is one of the most widely used security protocols in Internet. The cryptographic algorithms RC4 and HMAC have been in use for achieving security services like confidentiality and authentication in the SSL / TLS. But recent attacks on RC4 and HMAC have raised questions about the reliability of these algorithms. Hence MAJE4 and MACJER-320 have been proposed as substitutes for them. Detailed studies on the performance of these new algorithms were carried out; it has been observed that they are dependable alternatives.

Contents	Page No.
List of Algorithms	vi
List of Figures	vii
List of Tables	viii
1. Introduction	1
1.1 Increasing Dependency of Modern World on Computers	1
1.2 Security Attacks	3
1.3 Threats on Internet	5
1.4 A Short Description on Different Aspects of Security Services	8
1.5 Secure Web Communication	10
1.6 Layout of the Thesis	11
2. Studies on PRNGs & Stream Ciphers and Design & Development of a Novel Stream Cipher & its Applications	13
2.1 Introduction to Pseudo Random Number Generators and Stream Ciphers	15
2.1.1 Design Criteria of Pseudo Random Number Generators	16
2.1.2 Studies on Popular PRNGs and Stream Ciphers	17
2.1.2.1 Shift Register Based Generators	18
2.1.2.1.1 Linear Shift Register	18
2.1.2.1.2 Nonlinear Shift Register	19
2.1.2.2 Arithmetic and Algebraic Operations Based Generators	20
2.1.2.2.1 Linear Congruential Generators (LCG)	20
2.1.2.2.2 $X^2 \text{ mod } N$	21
2.1.2.3 JEROBOAM	22
2.1.2.4 RC4	22
2.1.3 Empirical Randomness Tests	24
2.1.3.1 Frequency Test (mono bit test)	25
2.1.3.2 Serial Test (two bit test)	25
2.1.3.3 Poker Test	26
2.1.3.4 Runs Test	26

2.1.3.5 Autocorrelation Test	27
2.1.4 Implementation of PRNGs	28
2.1.4.1 Shift Register Based Generators	28
2.1.4.1.1 LFSR	28
2.1.4.1.2 Geffe Generator	28
2.1.4.2 Arithmetic and Algebraic Operations Based Generators	29
2.1.4.2.1 LCGs	29
2.1.4.2.2 $X^2 \bmod N$	29
2.1.4.3 JEROBOAM	29
2.1.4.4 RC4	30
2.1.5 Results	30
2.1.5.1 Shift Register Based Generators	30
2.1.5.1.1 LFSR	30
2.1.5.1.2 Geffe Generator	31
2.1.5.2 Arithmetic and Algebraic Operations Based Generators	31
2.1.5.2.1 LCGs	31
2.1.5.2.2 $X^2 \bmod N$	33
2.1.5.3 JEROBOAM	33
2.1.5.4 RC4	34
2.1.6 Performance Evaluation	34
2.2 Design and Development of Novel Stream Cipher: MAJE4	35
2.2.1 Motivation for Design of a New Stream Cipher	35
2.2.2 Design Considerations of the Stream Cipher, MAJE4	36
2.2.3 Description of MAJE4	37
2.2.4 Randomness Tests	40
2.2.5 Results	40
2.2.6 Performance Evaluation	42
2.2.6.1 Timing Analysis	42
2.2.6.2 Memory Requirements	43
2.3 Development of a Hybrid System MARS4 using MAJE4	44
2.3.1 Need for a Hybrid System	44
2.3.2 Objectives for MARS4	46
2.3.3 Description of MARS4	47
2.3.4 Results	50
2.3.5 Performance Evaluation	53

2.3.5.1	Timing Analysis	53
2.3.5.2	Memory Requirements	54
2.4	Message Integrity Enhancement of Nested Hash Functions Using MAJE4	55
2.4.1	Introduction	55
2.4.2	Nested Hash Function	58
2.4.3	Use of Hash Code and MAJE4	60
2.4.4	Results	61
2.5	Conclusions	64
3.	Studies on Hash Functions and Design & Development of a Novel Hash Function JERIM-320	67
3.1	Study of Hash Functions	68
3.2	Review of Popular Hash Functions	70
3.2.1	Similarities	71
3.2.2	SHA-1	72
3.2.3	SHA-256	74
3.2.4	RIPEMD-160	76
3.2.5	RIPEMD-320	78
3.2.6	FORK-256	78
3.2.7	Differences	80
3.3	Design of a novel hash function: JERIM-320	81
3.3.1	Motivation and Design Factors	81
3.3.2	Description of JERIM-320	84
3.3.2.1	Input Block Length and Padding	85
3.3.2.2	Structure of JERIM-320	85
3.3.2.3	Single Step Operations	86
3.3.2.4	Order of the Message Words	88
3.3.2.5	Shifts	89
3.3.2.6	Boolean Functions	90
3.3.2.7	Constants	91
3.4	A Bird's Eye View on Hash Functions	92

3.5 Detailed Comparison of JERIM-320 with FORK-256	92
3.6 Security Analysis	95
3.6.1 JERIM-320	95
3.6.2 Comparison with FORK-256	96
3.7 Performance Evaluation	98
3.7.1 Practical Implementations	98
3.7.1.1 Comparison with FORK-256	98
3.7.1.2 Comparison with RIPEMD-320	100
3.7.1.3 Comparison with SHA-1, SHA-256, RIPEMD-160	102
3.7.2 Single Step Computation	104
3.8 Statistical Analysis for the Dual Functioning of JERIM-320	105
3.8.1 Introduction	105
3.8.2 Randomness Tests	106
3.8.3 Results	106
3.8.4 Performance Evaluation	107
3.9 Conclusions	108
3.10 Test Vectors	109
3.10.1 JERIM-320 using One Message Block in Single Step Operation	109
3.10.2 JERIM-320 using Two Different Message Blocks in Single Step Operation	112
4. Design and Development of a New Message Authentication Code: MACJER-320	116
4.1 MACJER-320	117
4.1.1 Introduction	117
4.1.2 Motivation and Design Factors	119
4.1.3 Description of MACJER-320	120
4.1.4 Security Analysis	123
4.1.5 Properties of MACJER-320	123
4.1.6 Performance Evaluation	124
4.1.7 Test Vector	126

4.2 Performance Evaluation between MACJER-320 and HMAC-SHA-1	126
4.2.1 HMAC	127
4.2.2 Security Analysis of MACs and Hash Functions	128
4.2.2.1 HMAC-SHA-1	128
4.2.3 Performance Evaluation	129
4.3 Conclusions	131
5. Use of MAJE4 and MACJER-320 in Secure Socket Layer / Transport Layer Security Protocol	133
5.1 Introduction	134
5.2 Motivation	135
5.3 Security Analysis of Algorithms	136
5.3.1 RC4	136
5.4 Alternate usage of MAJE4 and MACJER-320 in SSL / TLS protocol	139
5.4.1 MAJE4 & RC4	139
5.4.1.1 Timing Analysis & Memory Requirements	139
5.5 Conclusions	140
6. Summary of Results, Conclusions and Future Work	141
6.1 MAJE4	142
6.2 MARS4	142
6.3 Nested Hash Function	143
6.4 JERIM-320	144
6.5 MACJER-320	144
6.6 Use in SSL / TLS	145
6.7 Research Conclusions	146
6.8 Future Work	146
Published Work of the Author	149
Bibliography	152

List of Algorithms

vi

Algorithm 2.1: MAJE4	38
Algorithm 2.2: RSA	48
Algorithm 2.3: MARS4	49
Algorithm 2.4: Nested Hash Function	58
Algorithm 4.1: MACJER-320	121
Algorithm 4.2: HMAC	128

List of Figures

vii

Fig. 2.1: Working of the Stream Cipher: MAJE4	36
Fig. 2.2: Comparison of Number of Random Bits Produced per Second	43
Fig. 2.3: A Novel Hybrid Cryptographic System: MARS4	46
Fig. 2.4: A Comparison of Memory Sizes for MAJE4, RSA and MARS4	55
Fig. 2.5: Use of Combined Hash Code and Encryption	57
Fig. 2.6: Model of a Nested Hash Function	60
Fig. 2.7: Total Time Taken for Hash Code Generation & Verification and Encryption / Decryption	64
Fig. 3.1: Hash Code Generation and Verification	69
Fig. 3.2: A Single Step Operation of SHA-1	74
Fig. 3.3: A Single Step Operation of SHA-256	75
Fig. 3.4: A Single Step Operation of RIPEMD-160	77
Fig. 3.5: A Single Step Operation of FORK-256	80
Fig. 3.6: Outline of the Compression Functions of JERIM-320	86
Fig. 3.7: A Single Step Operation of JERIM-320	87
Fig. 4.1: Message Authentication Code (MAC)	118
Fig. 4.2: MACJER-320 Structure	122

List of Tables

viii

Table 2.1: Timing Analysis of JEROBOAM and RC4	35
Table 2.2: Statistical Analysis using Autocorrelation Test	40
Table 2.3: Statistical Analysis using Frequency, Serial, Poker and Runs Tests with 128-bit Key	41
Table 2.4: Statistical Analysis using Frequency, Serial, Poker and Runs Tests with 256-bit Key	41
Table 2.5: Timing Analysis	42
Table 2.6: Time Taken for Encryption or Decryption of Files of Various Sizes using MAJE4	50
Table 2.7: Time Taken for Encryption or Decryption of Files of Various Sizes using RSA (N=187)	51
Table 2.8: Time Taken for Encryption or Decryption of Files of Various Sizes using RSA (N=3431)	51
Table 2.9: Time Taken for Encryption or Decryption of Files of Various Sizes using RSA (N=44377)	52
Table 2.10: Time Taken for Encryption or Decryption of Files of Various Sizes using MARS4	52
Table 2.11: Time Required for Encryption and Decryption using MAJE4, RSA and MARS4	53
Table 2.12: Memory Requirements for MAJE4, RSA and MARS4	54
Table 2.13: Time Taken for Encryption or Decryption of Files of Various Sizes using MAJE4	62

viii

Table 2.14: Time Taken for Producing the Hash Code of Files of Various Sizes using Nested Hash Function	62
Table 2.15: Total Time Taken for Producing the Hash Code and Encryption / Decryption of Files of Various Sizes using Nested Hash Function and MAJE4	63
Table 3.1: Basic Notations in JERIM-320	84
Table 3.2: Order Rule of Message Words in Different Branches	88
Table 3.3: Message Order in Different Rounds	89
Table 3.4: Amount of Shifts in each Round for Different Message Blocks	90
Table 3.5: Boolean Functions	90
Table 3.6: Boolean Functions used in each Round	91
Table 3.7: Constants used in each Round	91
Table 3.8: Hash Functions at a Glance	92
Table 3.9: Comparison of JERIM-320 with FORK-256	93
Table 3.10: Comparison between the Number of Operations of JERIM-320 and FORK-256	99
Table 3.11: Performance Comparison between JERIM-320 and FORK-256	100
Table 3.12: Comparison between the Number of Operations of JERIM-320 and RIPEMD-320	101
Table 3.13: Performance Comparison between JERIM-320 and RIPEMD-320	102
Table 3.14: Comparison between the Number of Operations of SHA-1, SHA-256, RIPEMD-160 and JERIM-320	103
Table 3.15: Performance Comparison between SHA-1, SHA-256, RIPEMD-160 and JERIM-320.	103

Table 3.16: Statistical Analysis using Frequency and Serial Tests	106
Table 3.17: Statistical Analysis using Poker and Runs Tests	107
Table 3.18: Statistical Analysis using Autocorrelation Test	107
Table: 3.19: Performance Evaluation of JERIM-320 as Hash Function and as PRNG	108
Table 4.1: Variables used in MACJER-320	120
Table 4.2: Comparison between the Number of Operations of JERIM-320 and MACJER-320	125
Table 4.3: Performance Comparisons between JERIM-320 and MACJER-320	125
Table 4.4: Basic Notations in HMAC	127
Table 4.5: Comparison between the Number of Operations of MACJER-320 and HMAC	130
Table 4.6: Performance Comparison between MACJER-320 and HMAC	131
Table 5.1: Timing Analysis & Memory Requirements	140

Chapter 1

Introduction

1.1 Increasing Dependency of Modern World on Computers

Earlier computerized systems were used for the purpose of doing complex calculations and for huge data storage by scientists and engineers alone. Now the computers have become an inevitable component of modern human life. For example, in home for playing games and word processing, in office for spread sheet and data base management, in banks and other financial institutions for electronic banking, in airlines for air traffic control systems as well as reservations, in universities and other scientific institutions for the analysis of scientific and other experimental data like that for weather forecasting and for modeling & simulation, in process industry for the control of chemical and other plants, in engineering and electronic industries for the control of machine tools and robots, and so on. In short, computers have become inevitable and form a tool for controlling the economy as a whole.

The birth of Internet has opened up the gigantic world of information brining it under the finger-tips of even school children. For doing business, it offers a powerful and ubiquitous medium of commerce and enables greater connectivity of disparate groups throughout the world. Simple and cheap ways of data transfer like e-mails and video conferences have made drastic changes in day to day life, which could not have been dreamed of a few decades back. Most of these tremendous opportunities of computer based systems have resulted in huge savings in time and money and increased

comfort level of human living, but these would not have been possible without networking the computers. The influence of networking is steadily growing and the number of devices, which connect to the network, increases day by day.

As the usage of network spreads to more and more areas, it even involves transfer of critical information including those with serious financial implications like usage of ATMs for cash withdrawals as well as debit and credit cards for purchasing goods. Nowadays, every user sends various types of data and he would therefore like them to be protected while in transit over a public network. On one hand network provides a quick, easy and cost effective medium, while on the other hand the risk involved in secure data transfer is increasing heavily. Recurring events such as attacks of virus and worms and the success of criminal attackers illustrate the weakness in the current network. Some of the major threats in the Internet are loss of privacy, loss of data integrity and denial of service. Providing security of data in transit over the Internet has become a difficult and important task because of the steadily growing data volume and importance.

Organizations spending for protection of critical information assets continue to increase. In an attempt to secure current systems and networks, the organizations are resorting to a pool of information security systems. However, these systems have their inherent risks. Secured Socket Layer (SSL) [Frier A. et al., 1996] addresses some of these issues by providing security services such as confidentiality, data integrity and authentication.

Development of wireless communication networks led to the birth of a new era based on low-power and resource constrained systems like embedded systems, Personal Digital Assistants (PDAs), cellular telephones, smart cards, etc. A number of enabling technologies are being used in the delivery of mobile service applications. They include Interactive Voice Response (IVR), Short Messaging Service (SMS), Wireless Access Protocol (WAP) [WAP Forum, 1998], etc. The advent of light, low-power handheld computer devices such as PalmTM and Handspring, are changing the way in which mobile users interact with their home office. But the sensitive data transferred between these devices are subject to the risk of interception by a third party. Today, software and systems are created with disclaimers telling the public to use these technologies at their own risk. This is not a desirable state of affair. But, real-time encryption / decryption of data in handheld computers are complicated by the limited storage space available in these devices. Hence new techniques, which consume less memory, are to evolve for rescue of the system as a whole.

1.2 Security Attacks

The main goal of providing security is to restrict access to information and resources to just those principals that are authorized to have the access for that information. Attacks on the security of a computer or network are generally characterized as interruption, interception, modification and fabrication [William Stallings, 2001]. Interruption means an asset of the system is destroyed or becomes unavailable or unusable. This is an attack on availability. Examples include destruction of a piece of hardware, such as hard disk, the cutting of a communication line or the disabling of the file

management system. Interception means an unauthorized party gains access to an asset. This is an attack on confidentiality. The unauthorized party could be a person, a program or a computer. Examples include wiretapping to capture data in a network and the unauthorized copying of files or programs. Modification means an unauthorized party not only gains access to but tampers with, an asset. This is an attack on integrity. Examples include changing values in a data file, altering a program so that it performs differently and modifying the content of messages being transmitted in a network. Fabrication inserts counterfeit objects in to the system by an unauthorized party. This is an attack on authenticity; examples include the insertion of spurious messages in a network or the addition of records to a file.

A useful classification of these attacks is described below in terms of active and passive attacks [William Stallings, 2001]:

Active attacks: These attacks involve some modification of the data stream or the creation of a false stream and can be subdivided into four categories: masquerade, replay, modification of messages, and denial of service. Masquerading - sending or receiving messages using the identity of another principal without their authority. Replaying - storing intercepted messages and sending them at a later date. Modification of messages - intercepting messages and altering their contents before passing them on to the intended recipient. Denial of service - flooding a channel or other resource with messages in order to deny access by others.

Passive attacks: These attacks are in the nature of eavesdropping or monitoring of transmissions. The goal of the opponent is to obtain information that is

being transmitted. Two types of passive attacks are 1. release of message contents and 2. traffic analysis. The release of message content can be easily understood. A telephone conversation, an electronic mail message, and a transferred file may contain sensitive or confidential information, the release of which comes in the first category. Traffic analysis is the extraction of data during its transfer. It is more difficult since the information is masked. Even if the information is captured, the opponents cannot easily extract the information from the message. The common technique for masking contents is encryption. Passive attacks are very difficult to detect because they do not involve any alteration of the data. However, it is feasible to prevent the success of these attacks. The emphasis in dealing with passive attacks is on prevention rather than detection.

1.3 Threats on Internet

Today, the global threat landscape is arguably more dynamic than ever. Identity theft is an increasingly prevalent threat and is a major security issue, particularly for organizations that store and manage information regarding identity of people. Compromises that result in the loss of personal data could be quite costly, not only to the people whose identity may be at risk and their respective financial institutions, but also to the organization responsible for collecting the data. Data breaches that lead to identity theft could damage an organization's reputation, and undermine customer and institutional confidence in the organization. This data can include government-issued identification numbers, credit cards, bank cards, personal identification numbers (PINs), user accounts and email address lists.

The vulnerabilities of current IT systems have become regular fare in the press. Hardly a day passes without several news items or articles about some security bug or exploit. Hacker break-ins and compromises of personal computers running various versions of Microsoft Windows are of no news value. For example, 7000 PCs in the Stanford University campus [Stanford University, 2006] were compromised in August, 2003 and again hundreds more in the subsequent years. Industrial espionage is no new economic threat [Nowell Security Enforcement, 2003]. According to the Federal Bureau of Investigation, industrial espionage costs U.S. companies anywhere from \$24 billion to \$100 billion annually. In another survey by the American Society for Industrial security, the potential monetary losses as a result of security lapses cost American industries as much as \$63 billion [Ben N. Venzke, 2002]. Corporate Espionage is a reality in this age of the Internet and the global economy. In an anonymous survey by the Computer Security Institute (CSI) and the FBI in the US, over 50 percent of information security professionals cited corporate competitors as likely sources of cyber attack [Richard Power, 2000]. Economic and industrial espionage occurs around the world and U.S. companies are prime targets. Enterprise leaders must continue to decisively address the threats posed by corporate espionage and other methods of information security breaches.

Yet we continue to transmit purchase orders and other private messages over unsecured telephone lines via e-mail in ASCII text, which is the least common denominator for electronic text. We rely on passwords, cards, personal identification numbers, and keys to access restricted information or confidential files. But these forms of identification can be stolen, forged, lost,

or given away. Moreover, these forms serve primarily to identify the person. They cannot verify or authenticate that the person really is who he or she claims to be. Many systems rely on IP address verification that limits access to users with a specific domain name or Internet address. Basically, this procedure identifies an individual by the machine he or she uses. Anybody using a particular computer can impersonate the rightful owner.

To meet due diligence requirements, corporate information officers in all sectors must take measures to protect their networks and to create better security architectures. With no assurance regarding the security qualities or even the origin of software and systems, system owners have few components from which to construct sound security architectures. Consequently, we have entered a period of cyber security uncertainty. It is essential to protect the communication channels and the interfaces of any system that handles information that could be the subject of attacks. Secure Socket Layer (SSL) [Alan O. Freier et al., 1996] and Wireless Transport Layer Security (WTLS) [WAP Forum, 2000] are examples amongst various security protocol tools that have been proposed to address this issue. Security protocols are carefully designed to guard against loopholes. To this end, a practical SSL protocol has been adopted for protection of data in transit that encompasses all network services that use TCP/IP [Braden R., 1989] to support typical application tasks of communication between servers and clients.

The global e-Security market is estimated to be about \$27.7 billion in 2005 and is expected to rise at an average annual growth rate (AAGR) of 16.0%, reaching \$58 billion by 2010 [Smart Cards Expo, 2007]. This high growth rate is attributed to a higher demand for strong security solutions in

markets. Firewall and content management currently account for a majority share of the market. Unified threat management solutions are expected to become dominant. The Indian security market grew from Rs. 11 billion in 2002 to about Rs. 50 billion by 2006 according to IDC India [Smart Cards Expo, 2007].

1.4 A Short Description on Different Aspects of Security Services

In this section we discuss the four main issues that must be addressed while designing security systems. The four main security issues are confidentiality, authentication, integrity and non-repudiation. Confidentiality means that only the sender and the intended recipient should be able to access the contents of the information. Authentication means that the user accessing the information ensures that the message has come from the intended person and not from an imposter. That is, the receiver of the data should be able to determine its true origin. Integrity has to ensure that the received information is identical to the transmitted information without being modified by others during transmission. To ensure data integrity, the system must be able to detect data insertion, deletion and modification. Non-repudiation ensures that senders and receivers have undeniably transmitted or received information, respectively. Non-repudiation services prevent an individual from denying that previous actions had been performed. The goal is to ensure that the recipient of the data is assured of the sender's identity.

Cryptographic algorithms are utilized to encrypt an original plaintext message into a cipher text at the sender side and to decrypt the cipher text

back to the original message at the receiver side. The encryption and decryption processes generally depend on a secret key being shared between the sender and the receiver.

There are three types of cryptographic algorithms: symmetric-key algorithms, asymmetric-key algorithms, and hashing functions, which are explained as follows:

Symmetric key algorithms:- In Symmetric key algorithms or private key algorithms, both the sender and the receiver utilize the same key for both encryption and decryption. In a two-party communication, both parties must know the same key before transmission and measures must be taken to keep the key a secret. The key distribution becomes increasingly more difficult when the network grows since each pair of users must exchange keys. The total number of key exchanges required in an n-person network is $n(n-1)/2$. Though, symmetric key algorithms provide strong security, they suffer from this key distribution problem. The widely adopted symmetric key algorithms by the industry include Data Encryption Standard (DES) [NIST FIPSPUB 46-3, 1999], Triple DES (3DES) [NIST FIPSPUB 46-3, 1999], RC4 [Kaukonen K. and Thayer R., 1999] and Advanced Encryption Standard (AES) [NIST FIPSPUB 197, 2001].

Asymmetric key algorithms:- These are based on each party having their own private key, which is shared with no-one, and a public key that is known to all other communicating parties. This is also called public key algorithms. When sending a message to a particular receiver, the receiver's public key is used to encrypt the message. After receiving the message, it is

decrypted using the receivers own private key. Compared to symmetric key algorithms, asymmetric key algorithms eliminate the need to secretly distribute a key, and therefore solve the key distribution problem. Examples of asymmetric key algorithms are RSA [RSA Laboratories, 2002], DSA (Digital signature algorithms) [NIST FIPSPUB 186-3, 2007] and elliptic curve cryptography [SECG, 2000].

Hashing functions:- Unlike the other two types of cryptographic algorithms mentioned above, hashing functions do not involve the use of keys. They take a variable length string as input and convert it to a fixed length output. Well-known hash functions are MD5 [Rivest R.L., 1992] and SHA-1 [NIST FIPS-180-2, 2002].

1.5 Secure Web Communications

Internet communications that are based on the Transfer Control Protocol / Internet Protocol (TCP/IP), such as the Hypertext Transfer Protocol (HTTP), Telnet and File Transfer Protocol (FTP), are not secure because all communication occurs in plaintext. Confidential or sensitive information that is transmitted with these protocols can easily be intercepted and read unless the information is protected by encryption technology.

In addition, because any web client can send HTTP requests to a web server and exploit weaknesses in the HTTP protocol or its implementation, web servers that use only standard HTTP to communicate with web clients are easy targets for denial-of-service attacks and other types of attacks.

Many applications need to securely transmit data to remote applications and computers. Secure Socket Layer (SSL) is an Internet security protocol for point-to-point connections. Clients and servers are able to authenticate each other and to establish a secure link, or “pipe” across the Internet or Intranets to protect the Information transmitted. SSL was designed to solve this problem in an open standard. In SSL, a connection is made, parties authenticated, and data securely exchanged. The latest enhancement of SSL is called Transport Layer Security (TLS) [Michael Chernick C. et al., 2005]. In applications using SSL, the confidentiality of information is ensured using strong encryption technologies. SSL provides the transparent authentication of servers and clients. It uses the RSA algorithm to enable security using digital signatures [NIST FIPSPUB186-3, 2007] and digital enveloping. For very fast encryption and decryption of data for transmission after an SSL connection has been established, the RC4 algorithm has been the preferred algorithm. Other algorithms are available in the SSL specification as well. Based on the strong cryptography in SSL, users have confidence that their information is confidential, authentic and original during transfer over a network connection.

Few other network security mechanisms are firewalls, biometrics, antivirus software, steganography, passwords, network intrusion-detection systems, VPN systems, etc.

1.6 Layout of the Thesis

Chapter 1 points out the increasing dependency on computer systems and networks and consequently the growing need for network security. The

threats faced by the internet and the security services required to avoid or reduce the threats are also discussed.

Chapter 2 takes up the study on various pseudo random number generators / stream ciphers. The design of a new stream cipher MAJE4 and development of two new applications are also included.

Chapter 3 describes the study of five popular hash functions. A new hash function JERIM-320 is introduced for providing data integrity. It is suggested as an alternative for the present day hash functions. The new hash function's performance evaluation has been done.

Chapter 4 illustrates the development of a new message authentication code MACJER-320 and compares its performance with the current candidate.

Chapter 5 explores the use of newly developed algorithms MAJE4 and MACJER-320 in Secure Socket Layer / Transport Layer Security Protocol.

Chapter 6 concludes by summarizing the results in the work and the possible developments in future.

The performance evaluation of different algorithms detailed in this work has been done using Pentium IV Processor, Linux Operating System and C compiler.

Chapter 2

Studies on PRNGs & Stream Ciphers and Design & Development of a Novel Stream Cipher & its Applications

Abstract:

Section 2.1 of this chapter introduces Pseudo Random Number Generators (PRNGs) and stream ciphers, bringing out their growing importance in various applications. The study, implementation, statistical analysis and performance evaluation of various PRNGs and stream ciphers have been carried out. Extensive software implementation as well as statistical experimentation was conducted and a stream cipher JEROBOAM was identified upon which further studies could be conducted for bringing improvements.

The aim of Section 2.2 is to design a stream cipher which generates a long unpredictable key stream with better performance and which can be used for cryptographic applications. Upon this view, a new fast stream cipher MAJE4 was designed with a variable key size of 128-bit or 256-bit. The randomness property of the stream cipher was analysed by using the empirical tests. The performance evaluation of the MAJE4 was done in comparison with JEROBOAM.

Section 2.3 focuses on developing an enhanced hybrid system by combining the two cryptographic methods with a view to getting the

advantages of both. A novel and fast hybrid technique MARS4 was developed using MAJE4 and the popular asymmetric key algorithm RSA. The performance evaluation of MARS4 was done in comparison with MAJE4 and RSA.

Further work aims at providing integrity and confidentiality of messages in a swift and cost effective manner and is described in Section 2.4. A nested hash function with lower computational and storage demands was developed with a view to providing integrity in addition to the confidentiality available with MAJE4.

2.1 Introduction to Pseudo Random Number Generators and Stream Ciphers

Random numbers have been in use traditionally for games, computer simulations, test generation for the performance evaluation of computer algorithms, Monte Carlo techniques [Halton J.H., 1970] in numerical analysis, statistical sampling, stochastic optimization methods, etc. Today, security issues are coming to the forefront because of the increasingly demanding security requirements in many new applications on the internet such as e-mail, e-commerce, e-governance, etc. Hence PRNGs are used by and large in the development of privacy software for generating public / private key pairs, creating digital signatures [NIST FIPSPUB186-2, 2000] and message authentication codes, developing stream ciphers and in many other uses of encryption for various network security applications.

The computers used today are completely deterministic in operation, and therefore lack convenient sources of randomness. As a result, developers of security software rely on software-based PRNGs. Now it is hard to imagine a well-designed cryptographic application that does not use PRNGs; they have gained an obligatory role, which relies on randomness to generate keys, creating padding bytes, and deriving other security-critical parameters like passwords.

Stream ciphers are an important class of symmetric encryption algorithms. Their basic design feature is the same as that for a One-Time-Pad cipher [Frank Rubin, 1997], which encrypts the plain text by XORing with a

random key produced by PRNGs. The stream ciphers require only a short random key, which is expanded into a pseudo-random key stream, that is then XORed with the plain text to generate the cipher text. Again the same key stream is used to decrypt by XORing with the cipher text to form the plain text. Thus the stream ciphers used in cryptographic algorithms rely on these PRNGs for producing cipher texts.

Stream ciphers are usually used in applications where large amounts of data are employed, or extremely high throughput is needed, or low complexity hardware is a requirement. Most cutting-edge applications with these requirements are in multimedia applications, for example music and video and mobile phones.



With almost all security protocols relying on sources of randomness, possible flaws in random number generator have become a common security problem. But creating good random numbers is a hard problem, so hard that there isn't a library we can just use. Hence in-depth exploration of this area by the research community and enhancement of capabilities of PRNGs is a need of the hour.

2.1.1 Design Criteria of Pseudo Random Number Generators

Random number generators have a central place in cryptographic designs owing to their property of picking numbers unpredictably and using these numbers to choose cryptographic keys [Seigenthaler T., 1985], [William Aiello et al., 1995], [Boyar J., 1989]. In order to understand the strength of a cryptographic algorithm, which is the ability to resist attacks [Bruce Schneier, 1996] the matter of predictability is extremely important.

PRNGs used for cryptographic purposes are required to have:

1. maximum period to accommodate the long length of the transmitted message.
2. capability to speed up the process.
3. complexity for analysis, since analysis could penetrate the cryptographic system.
4. competence to produce a good distribution of values.

The aim is to produce a highly random sequence so that the cryptanalytic attacks are not feasible.

Here some of the popular PRNGs and stream ciphers are considered, with a view to analyzing them, evaluating their performance and to select an appropriate one for further development.

2.1.2 Studies on Popular PRNGS and Stream Ciphers

The PRNGs and stream ciphers considered for study, implementation and statistical analysis in this work are:

1. Shift Register Based Generators
 - 1.1 Linear Shift Register [Bruce Schneier, 1996]
Linear Feedback Shift Register (LFSR)
 - 1.2 Nonlinear Shift Register [Wei Zeng D. et al., 1991]
Geffe Generator
2. Arithmetic and Algebraic Operations Based Generators
 - 2.1 Linear Congruential Generators (LCGs) [Knuth D.E., 1997]
 - 2.2 $X^2 \bmod N$ [Blum M. et al., 1986]

3. A Fast Stream Cipher 'JEROBOAM' [Herve Chabanne and Emmanuel Michon, 1998]

4. RC4 Stream Cipher

2.1.2.1 Shift Register Based Generators

2.1.2.1.1 Linear Shift Register

The simplest kind of feedback shift register is the Linear Feedback Shift Register (LFSR). It is made up of two parts: a shift register and a feedback function. The shift register is a sequence of bits. The length of a shift register is figured in bits, if it is n bits long, it is called an n -bit shift register. Each time a bit is needed all of the bits in the shift register are shifted one bit to the right. The new left-most bit is computed as a function of the other bits in the register. That is when we simply XOR certain bits in the register, the list of these bits is called a tap sequence. The period of a shift register is the length of the output sequence before it starts repeating [Bruce Schneier, 1996]. LFSRs are easily implemented in digital hardware.

An n -bit LFSR can be in one of the 2^n-1 internal states. This means that it can, in theory, generate 2^n-1 bits long pseudo random sequence before repeating. Here 2^n-1 bits are generated since a shift register filled with zeros can cause the LFSR to output a never-ending stream of zeros. For a particular LFSR to be a maximal period LFSR, the polynomial formed from a tap sequence must be a primitive polynomial. A polynomial over a unique factorization domain is called primitive if its coefficients are relatively prime. Any field is a unique factorization domain, in which each nonzero element is a unit and there are no primes. The integers form a unique factorization domain

in which the units are +1 and -1, and the primes are $\pm 2, \pm 3, \pm 5, \pm 7, \pm 11$, etc. In general there is no easy way to generate primitive polynomials for a given degree. Easiest way is to choose a random polynomial and test whether it is primitive. For example consider the polynomial $x^{13} + x^5 + x^3 + x + 1$. The first number says 13 is the length of the LFSR. Take the bits at positions 13, 5, 3 and 1, and do XOR operation in these bits to produce a resultant bit and store that bit in the Most Significant Bit (MSB) of LFSR after shifting all the bits in the shift register once to the right.

2.1.2.1.2 Nonlinear Shift Register

Since LFSR sequences can be predicted from a small subset of their subsequences, it has been proposed to use a non-linear feedback mechanism to produce a pseudo-random sequence [Bruce Schneier,1996]. The resulting sequence will be more difficult to analyze.

In non linear shift register, LFSRs of different lengths and different feedback polynomials are considered. If the lengths are all relatively prime and the feedback polynomials are all primitive, the whole generator is of maximal length. Key for each LFSR is given as its initial state. Every time a bit is needed, the LFSRs are shifted once to the right. The output bit is a nonlinear function of different bits of LFSRs. This function is called a combining function. The combining function used in the Geffe generator is given below as an example.

Geffe generator uses three LFSRs combined in a nonlinear manner [Wei Zeng D. et al., 1991]. Two of the LFSRs are inputs into a multiplexer, and the third LFSR controls the output of the multiplexer. If $LFSR_1, LFSR_2$

and $LFSR_3$ are the outputs of the three LFSRs, then the output of the Geffe Generator (result) is found using the nonlinear function given by the equation

$$\text{result} = (LFSR_1 \wedge LFSR_2) \oplus ((\neg LFSR_1) \wedge LFSR_3)$$

From the 'result' of this equation any number of bits can be taken to form the random sequence. The procedure can be repeated to produce more random numbers so that the length of the random sequence produced can be increased as desired.

The period of the generator is the least common multiple of the periods of the three generators. Assuming the degrees of the three primitive feedback polynomials are relatively prime, the period of this generator is the product of the periods of the three LFSRs.

2.1.2.2 Arithmetic and Algebraic Operations Based Generators

Two types of these generators have been considered, they are Linear Congruential Generator and $X^2 \bmod N$.

2.1.2.2.1 Linear Congruential Generators (LCG)

LCG is one of the oldest type of random number generators [Knuth D.E, 1969]. This is still the most common type because of its simple iterative formula $X_n = aX_{n-1} + b \bmod m$, which is relatively fast and easy to compute. The values X_0 (Seed or key) and m (modulus) are fixed by the designer. Here 'a' and 'b' are constants: 'a' is the multiplier and 'b' is the increment.

The simple formula means that the LCG is relatively easy to program, but selecting appropriate parameter values for X_0 and m are not easy. The current level of analysis seems insufficient to predict the parameters for best randomness and hence the design of a statically acceptable LCG involves much trial and error and expensive randomness testing. This generator has a period no longer than m . If X_0 and m are properly chosen, then the generator will be a maximal period generator. Here m is a prime number.

Advantages of LCGs are that they are fast and requiring very few operations per bit. But unfortunately LCG cannot be used in cryptography since they are predictable. They remain useful for non-cryptographic applications like simulations. They are generally efficient and show good statistical behavior with respect to most of the reasonable empirical tests.

2.1.2.2.2 $X^2 \bmod N$

The next generator based on arithmetic and algebraic operations is ' $X^2 \bmod N$ ' developed by Blum, Blum and Shub [Blum M. et al., 1986]. This PRNG seems unique in that it is claimed to be 'polynomial time unpredictable' and 'cryptographically secure'. PRNG consists of the iterative equation $X_{[i+1]} = X_{[i]}^2 \bmod N$ where N is the product of two large distinct primes [Ritter T., 1991].

Vazirani and Vazirani shows that $\log_2(N)$ lsb's of $X_{[i+1]}$ can be safely used [Vazirani U and Vazirani V, 1985]. Select N as the product of two large distinct primes P and Q . Prime P is special if $P = 2P_1 + 1$ and $P_1 = 2P_2 + 1$, where P_1 and P_2 are odd primes [Ritter T., 1991].

Because the $X^2 \bmod N$ generator generally defines multiple cycles with various numbers of states, the initial value $X_{[0]}$ must be specifically selected to make sure that it is not on a short cycle.

For cryptographic work, both X and N will be very large quantities, hence the multiplication and division required for each PRNG step would be slow. To form N , some sort of probabilistic primality test [The prime pages, 2006] on very large random numbers is applied. The $X^2 \bmod N$ PRNG is claimed to be ‘unpredictable’ (when properly designed), but even then there is no guarantee of secrecy because it could not resist all the attacks.

2.1.2.3 JEROBOAM

The new fast stream cipher JEROBOAM was proposed by Chabanne and Michon [Herve Chabanne and Emmanuel Michon, 1998]. It works with a key of 128 or 248 bits. JEROBOAM produces a pseudo random stream which can be used as a symmetric cipher to XOR a clear text of any length. The heart of JEROBOAM consists of eight 32-bit mwc (multiply with carry) registers, a FIFO queue of two 16-bit data and a particular 16-bit datum. One can choose between a 248-bit key and a 128-bit key. In 248-bit key, the key is given by eight 32-bit words. The 32nd bit in each word must be zero and none of these words can be zero. In 128-bit key, key is given by eight 16-bit words. The algorithm remains the same for both the 248-bit key and 128-bit key.

2.1.2.4 RC4

RC4 is a variable key size stream cipher developed in 1987 by Ron Rivest for RSA Data Security, Inc. The RC4 stream cipher has two phases,

the key set-up and the key stream generation. Both phases must be performed for every new key. The algorithm is based on the use of a random permutation. A variable length key K , of size 1 to 256 bytes is used to initialize a 256-byte state vector S , with elements S_0, S_1, \dots, S_{255} .

Initially the entries of S are set with the values 0 to 255 in ascending order. A temporary vector T , is also created. For a key of length keylen bytes, the first keylen elements of T , are copied from K , and then K is repeated as many times as necessary to fill out T . Then, T is used to produce the initial permutation of S . The pseudo-code for the key set-up is as:

```
for i = 0 to 255
  Si = i
  Ti = K[i mod keylen]
end for
k = 0
for i = 0 to 255
  k = (k + Si + Ti) mod 256
  Swap(Si, Sk).
end for
```

Once S is initialized, the input key is no longer used. The next phase is key stream generation which is described by the pseudo-code as:

```
i = 0
k = 0
while (true)
  i = (i + 1) mod 256
  k = (k + Si) mod 256
```

```
Swap(Si, Sk)  
t = (Si + Sk) mod 256  
key = St.  
end loop
```

For encryption, the value 'key' is XORed with the next byte of plaintext. For decryption, the value 'key' is XORed with the next byte of cipher text.

2.1.3 Empirical Randomness Tests

Every random sequence should be tested carefully before putting into extensive use. Empirical tests [Knuth D.E., 1997] are used for this purpose. In these tests one manipulates the groups of numbers of the sequence resulting in certain statistics. Then these statistics are applied to statistical tests like Chi-square test [Menezes A. et al., 1997], which is the best known of all the statistical tests to accept hypothesis whether the generated random sequence has the similar distribution of a purely random sequence or not. Studies are carried out using the following five tests, because these tests are widely used for determining whether the binary sequences possess the characteristics that a truly random sequence would exhibit.

1. Frequency Test (mono bit test)
2. Serial Test (two bit test)
3. Poker Test
4. Runs Test
5. Autocorrelation Test

2.1.3.1 Frequency Test (mono bit test)

In this test, we determine the number of zeros and ones in the generated random sequence. Let n_0 and n_1 denote the number of zeros and ones respectively.

The statistic used is

$$X_1 = \frac{(n_0 - n_1)^2}{n}$$

where n is total number of bits in the sequence. The X_1 approximately follows the χ^2 distribution with one degree of freedom.

2.1.3.2 Serial Test (two bit test)

In this test, we determine whether the number of occurrences of 00, 01, 10 and 11 as subsequences of random sequence S . Let n_0 and n_1 be same as frequency test. Let n_{00} , n_{01} , n_{10} and n_{11} denote the number of occurrences of 00, 01, 10, and 11 respectively in S .

$$\text{Then } n_{00} + n_{01} + n_{10} + n_{11} = n - 1$$

The statistic used is

$$X_2 = \frac{4}{n-1} (n_{00}^2 + n_{01}^2 + n_{10}^2 + n_{11}^2) - \frac{2}{n} (n_0^2 + n_1^2) + 1$$

which approximately follows the χ^2 distribution with two degrees of freedom, if $n \geq 21$.

2.1.3.3 Poker Test

In this test the generated random sequence S is divided into $k = n/m$ non-overlapping parts each of length m , where m be a positive integer such that $\lfloor n/m \rfloor \geq 5 \cdot (2^m)$. Let n_i be the number of occurrences of the i^{th} type of sequence of length m , where $1 \leq i \leq 2^m$. The Poker test determines the sequences of length m , each appear approximately the same number of times in S .

The statistic used is

$$X_3 = \frac{2^m}{k} \left(\sum_{i=1}^{2^m} n_i^2 \right) - k$$

which approximately follows the χ^2 distribution with $2^m - 1$ degrees of freedom. If $m = 1$, the Poker test yields the Frequency test.

2.1.3.4 Runs Test

In this test, we determine whether the number of runs of various lengths in the sequence S is as expected in the random sequence. The expected number of gaps (or blocks) of length i in a random sequence of length n is $e_i = (n - i + 3) / 2^{i+2}$. Let k be equal to the largest integer i for which $e_i \geq 5$. Let B_i and G_i be the number of blocks and gaps respectively of length i , in S , for each i , $1 \leq i \leq k$.

The statistic used is

$$X_4 = \sum_{i=1}^k \left(\frac{(B_i + e_i)^2}{e_i} + \frac{(G_i - e_i)^2}{e_i} \right)$$

which approximately follows the χ^2 distribution with $2k-2$ degree of freedom.

2.1.3.5 Autocorrelation Test

In this test the correlations between the sequence S and its (non-cyclic) shifted version are checked. Let d be a fixed integer, where $1 \leq d \leq n/2 - 1$. The number of bits in S not equal to their d -shifts is,

$$A(d) = \sum_{i=0}^{n-d-1} s_i + s_{i+d}$$

where $+$ is XOR operation. The statistic used is,

$$X_5 = 2 \left(\frac{A(d) - \frac{n-d}{2}}{\sqrt{n-d}} \right)$$

which approximately follows normal distribution, $N(0,1)$, if $n - d \geq 10$.

2.1.4 Implementation of PRNGs

2.1.4.1 Shift Register Based Generators

2.1.4.1.1 LFSR

LFSR was implemented to generate any number of random numbers to produce the random sequence of any length. Also the required bits were taken from each random number to form the random sequence.

2.1.4.1.2 Geffe Generator

The implementation of Geffe generator contains three LFSRs of different lengths with different feedback polynomials. These three LFSRs are combined in a non-linear manner. If $LFSR_1$, $LFSR_2$ and $LFSR_3$ are the outputs of the three LFSRs, then the output of the Geffe Generator (result) is found out using the nonlinear function as given in the equation

$$\text{result} = (LFSR_1 \wedge LFSR_2) \oplus ((\neg LFSR_1) \wedge LFSR_3)$$

From the 'result' of this equation, any number of bits can be taken to form the random sequence. The procedure can be repeated to produce more random numbers so that the length of the random sequence produced can be increased as desired.

2.1.4.2 Arithmetic and Algebraic Operations Based Generators

2.1.4.2.1 LCGs

The following functions generate pseudo-random numbers using the LCGs. They are `drand48()`, `erand48()`, `lrand48()`, `nrand48()`, `rand48()` and `jrand48()`. These are available with the C libraries and are used for LCG testing. The user can give the number of random numbers and the number of bits required per random number as inputs to produce the random sequence. Analysis was done using 1000 different such streams produced from LCG with different initial seed for each random stream. All the LCGs considered were tested using the empirical tests listed in section 2.1.3.

2.1.4.2.2 $X^2 \bmod N$

The seed value X was chosen to be ≤ 10000 . The value N was chosen as 65745881, which is the product of two large primes P and Q . The values for P and Q are 8209 and 8009 respectively. Bits were taken from $X^2 \bmod N$ to generate the random sequence.

2.1.4.3 JEROBOAM

After performing the JEROBOAM algorithm the value obtained in the variable `cmb` (6th step in the algorithm) will be a 16 bit random number. This number was used to XOR with plain text. The algorithm was repeated for many random numbers to form the random sequence. Up to 16 bits were taken from each random number.

2.1.4.4 RC4

RC4 was implemented using the pseudo code given in section 2.1.2.4 to generate any number of random numbers and producing the random sequence of any length. Up to 8 bits were taken from each random number. The value 'key' shown in the pseudo code was used to XOR with the plaintext to form the cipher text.

2.1.5 Results

The summary of all the results obtained is presented in this section. The Frequency, Serial, Poker and Runs tests were analyzed using the Chi-square table, while Autocorrelation test was analyzed using Normal table.

2.1.5.1 Shift Register Based Generators

2.1.5.1.1 LFSR

All the five randomness tests were conducted for LFSR. The tests were found accepting or rejecting depending upon the factors like whether the polynomial used to specify the tap sequence was primitive or not. For the same polynomial, when different seeds were given, different results were obtained. For the same polynomial, same number of random numbers and same seed, if the number of bits taken from each random number is different then also different results were obtained. Analysis was done till 10000 random numbers were produced from a LFSR.

2.1.5.1.2 Geffe Generator

Three different LFSRs were used here. The polynomials can be of different lengths, tap sequences and seed values. Here also the randomness tests were found accepting or rejecting depending upon the factors explained for LFSR. These factors are applicable for all the three LFSRs. Analysis was done till 10000 random numbers were produced from Geffe. Results were found to be the same as LFSR for all the five randomness tests.

2.1.5.2 Arithmetic and Algebraic Operations Based Generators

The following generators were analyzed using 1000 different random streams having different initial seeds.

2.1.5.2.1 LCGs

erand48() and drand48()

Both erand48() and drand48() were showing the randomness property for 15 least significant bits for frequency and serial tests. As the number of random numbers produced was becoming more the randomness property was found decreasing. For about 1000 random numbers the randomness property was not showing considerable variation, but beyond 1000 it was found decreasing.

In poker test, both erand48() and drand48() were showing the randomness property for the 15 least significant bits. The test was found accepting even when the number of random numbers was increased (tested till 10000).

erand48() passed runs test for some specific range of bits and specific number of random numbers. i.e. 4 to 16 bits for 100 random numbers, 8 to 16 bits for 500 random numbers, 10 to 15 bits for 2000 random numbers and 13 to 15 bits for 10000 random numbers. drand48() did not pass this test.

Both erand48() and drand48() failed in autocorrelation test.

lrand48(), mrand48(), nrand48() and jrand48()

lrand48(), mrand48(), nrand48() and jrand48() were showing the randomness property for 18 least significant bits for frequency test and 16 least significant bits for serial test. Even when more random numbers were produced, the randomness property was found exhibiting (tested till 10000 random numbers).

In poker test, lrand48(), mrand48(), nrand48() and jrand48() were showing the randomness property for the 16 least significant bits. Here also the generators passed the tests when the number of random numbers had been increased till 10000.

Runs test was passed by nrand48() only, for some specific range of bits and specific number of random numbers. i.e. 4 to 16 bits for 100 random numbers, 8 to 16 bits for 500 random numbers, 10 to 16 bits for 2000 random numbers and 13 to 16 bits for 10000 random numbers. lrand48(), mrand48() and jrand48() did not pass the runs test.

lrand48(), mrand48(), nrand48() and jrand48() failed in autocorrelation test.

2.1.5.2.2 $X^2 \bmod N$

For $N = 65745881$ and $X \leq 10000$, randomness property was exhibiting for 26 least significant bits of the random number for both frequency and serial tests. It was also seen that as the number of random numbers was increased the randomness property was decreasing and the generator passed for 175 random numbers only. Here the results were confirming to what Vazirani proved [U.Vazirani and V.Vazirani, 1985], as per which $\log_2(N)$ least significant bits will show randomness property, that is $\log_2(65745881)$ is 25.97.

In poker test, $X^2 \bmod N$ was showing the randomness property for the 26 least significant bits. Analysis has shown that as the number of random numbers was increased the randomness property was getting reduced. Here it passed for 120 random numbers only.

In runs test, the result of analysis was same as in the Poker test. $X^2 \bmod N$ failed in autocorrelation test.

2.1.5.3 JEROBOAM

All the five randomness tests were carried out in JEROBOAM. If the key value given as input to the program is not correct, then it will simply exit without doing any of the steps in the algorithm. But if the key is acceptable according to the key setup described, JEROBOAM is producing random numbers. The 16-bit number produced from the 6th step of the algorithm was tested with the above tests and it passed for all of them. Hence the basic

features of this algorithm can be explored further for developments in this area.

2.1.5.4 RC4

All the five randomness tests were carried out in RC4. The value 'key' produced using the pseudo code as shown in section 2.1.2.4 was tested with the above tests and it passed for all of them.

In addition to these, the time required to generate random sequences of different lengths were also found out for JEROBOAM and RC4, since they have passed all the randomness tests.

2.1.6 Performance Evaluation

The summary of the performance evaluation is presented here. The time required for producing different random bit sequences of same length in JEROBOAM and RC4 were compared. The results are shown in Table 2.1

On comparing the memory required for executable files of the two generators, JEROBOAM was found occupying lesser space compared to RC4. The size required for optimized code for JEROBOAM was 6341 bytes, while for RC4, it was 8077 bytes.

Table 2.1: Timing Analysis of JEROBOAM and RC4

PRNGs	Number of random numbers in each run	Number of bits from each random number	Total number of bits	Time (Sec.)
JEROBOAM	200000	16	3200000	0.07
	400000	16	6400000	0.14
	600000	16	9600000	0.21
	800000	16	12800000	0.28
	1000000	16	16000000	0.37
RC4	400000	8	3200000	0.01
	800000	8	6400000	0.02
	1200000	8	9600000	0.03
	1600000	8	12800000	0.04
	2000000	8	16000000	0.05

2.2 Design and Development of Novel Stream Cipher: MAJE4

2.2.1 Motivation for Design of a New Stream Cipher

Unlike the other type of symmetric cryptographic algorithms called block ciphers, stream ciphers encrypt / decrypt each bit independently. Stream ciphers are much faster than block ciphers and they have greater software efficiency. Due to these features, stream ciphers have been the choice for several communication standards like IEEE 802.11b [Sultan Weatherspoon,

2000] and Bluetooth [Specification of the Bluetooth system, 2001]. Hence it is required to have more studies and developments in this area.

A new stream cipher, which efficiently generates pseudo-random bits that are identical to truly random bits, forms the goal here. The stream cipher has been named as MAJE4. It works as shown in Fig. 2.1

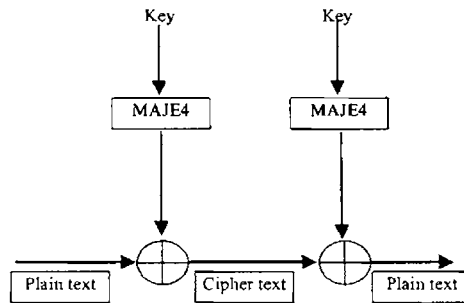


Fig. 2.1 Working of the Stream Cipher: MAJE4

2.2.2 Design Considerations of the Stream Cipher, MAJE4

1. MAJE4 design should work efficiently on 32-bit processors.
2. It should pass all the empirical tests described in section 2.1.3.
3. The encryption sequence should have a large period. A pseudo random number generator uses a function that produces a deterministic stream of bits that eventually repeats itself. The longer the key, the longer it takes for a brute force attack [Richard Clayton, 2001] and more difficult to do the cryptanalysis.

4. It should have a flexible security choice with key sizes of 128 or 256 bits.
5. The key stream should approximate the properties of a true random stream as possible.
6. It must be suitable for hardware or software and using only primitive computational operations commonly found on microprocessors.
7. It has to be simple and fast. The algorithm must be easy to implement. The task of determining the strength of the algorithm also has to be simple.
8. It should have low memory requirement to make it suitable for devices with restricted memory.
9. It should use mixed operators. The use of more than one arithmetic and / or Boolean operator complicates cryptanalysis. Primitive operators like + and ^ may be used since these operators do not commute resulting in a difficult cryptanalysis.

2.2.3 Description of MAJE4

The mathematical operators used are

1. Addition: Addition of words, denoted by +
2. Bitwise exclusive OR: This operation is denoted by ^.

3. Right shift operation: The right shift of word x right by y bits is denoted by $x \gg y$.

All the design considerations mentioned in section 2.2.2 were taken care while designing MAJE4 stream cipher. Here the randomness property has been tested with the primary empirical tests described in section 2.1.3. Since MAJE4 uses only primitive computational operators like $+$, \wedge , \gg etc, it is suitable for hardware and software implementations. The algorithm MAJE4 is easy to implement and fast also. The nonlinearity is obtained by alternative usage of $+$ and \wedge operators, which complicates cryptanalysis.

Key setup: One can choose between a 128-bit key and 256-bit key, which are stored as follows:

128-bit key: The four 32-bit words, ie. $key_{[0]}$, $key_{[1]}$, $key_{[2]}$ and $key_{[3]}$ are considered for storing the key.

256-bit key: The key is stored in eight 32-bit words $key_{[0]}$, $key_{[1]}$, $key_{[2]}$, $key_{[3]}$, $key_{[4]}$, $key_{[5]}$, $key_{[6]}$ and $key_{[7]}$.

Algorithm 2.1: MAJE4

Step 1: Assign the key length kl either as 128-bit or 256-bit.

Step 2: if $kl = 128$ then

$kl_n = 2, div = 4$

else

$kl_n = 3, div = 8$

Step 3: if $kl = 128$ then consider two lsb's of $key_{[0]}$ and find its decimal equivalent and store in the variable 'in'.

else

if $kl = 256$ then consider three lsb's of $Key_{[0]}$ and find its decimal equivalent and store in the variable 'in'.

Step 4: $ran = key_{[0]} \wedge key_{[in]}$

Step 5: if $kl = 128$ then consider two lsb's of ran and find its decimal equivalent and store in the variable 'in1'.

Step 6: if $kl = 256$ then consider three lsb's of ran and find its decimal equivalent and store in the variable 'in1'.

Step 7: check the 16th bit in ran ,

if it is 1 then

$$newran = (key_{[in1]} + key_{[in1+1 \bmod div]}) \wedge (key_{[in1+2 \bmod div]} + key_{[in1+3 \bmod div]})$$

else

$$newran = (key_{[in1]} \wedge key_{[in1+1 \bmod div]}) + (key_{[in1+2 \bmod div]} \wedge key_{[in1+3 \bmod div]})$$

Step 8: The output 32-bit word is $newran$, which can be used to XOR with the corresponding word in the plain text.

Step 9: Advance all the keys as

$$key_{[i]} = key_{[i]} * key_{[i]} + key_{[i]} \ggg 20$$

Step 10: go to step3

2.2.4 Randomness Tests

The analysis of MAJE4 is done using the empirical tests explained in section 2.1.3, as these tests can be effectively used for determining whether the binary sequences possess the specific characteristics that a truly random sequence need to have. The results of analysis are explained in section 2.2.5.

2.2.5 Results

Here the Frequency, Serial, Poker and Runs tests are analysed using the Chi-square table and Autocorrelation test is analysed using Normal table, as specified for each randomness tests. The fast stream cipher MAJE4 successfully passed all the five empirical tests for every run. Tables 2.2, 2.3 and 2.4 show the results of the specified tests.

Table 2.2: Statistical Analysis using Autocorrelation Test

Number of random numbers generated	Total number of bits produced	Statistical Analysis	
		128-bit key	256-bit key
		Autocorrelation test	Autocorrelation test
300	9600	2.0855	1.5007
500	16000	2.2158	1.8581
800	25600	2.0762	2.3439
1000	32000	2.4944	1.6045
2000	64000	1.6368	1.4902

Table 2.3: Statistical Analysis using Frequency, Serial, Poker and Runs Tests with 128-bit Key

Number of random numbers generated	Total no. of bits produced	Statistical Analysis			
		128-bit key			
		Frequency test	Serial test	Poker test	Runs test
500	16000	1.3690	3.6252	294.27	24.00
1000	32000	1.5961	3.5682	566.25	28.43
1500	48000	1.4083	3.4354	538.08	19.97
4000	128000	0.0632	1.2458	2099.7	29.93
6000	192000	0.2475	1.8224	2022.3	30.12
8000	256000	0.4100	2.9614	4101.1	32.33
10000	320000	0.0903	3.7286	4159.0	30.25

Table 2.4: Statistical Analysis using Frequency, Serial, Poker and Runs Tests with 256-bit Key

Number of random numbers generated	Total no. of bits produced	Statistical Analysis			
		256-bit key			
		Frequency test	Serial test	Poker test	Runs test
500	16000	0.3610	4.2347	269.69	12.79
1000	32000	0.8820	5.5466	501.05	15.86
1500	48000	0.3100	5.9851	481.64	18.87
4000	128000	0.0031	1.6390	2044.4	27.17
6000	192000	0.0316	2.3485	2011.3	27.32
8000	256000	0.0082	2.3659	4109.3	21.40
10000	320000	0.0630	1.6504	4116.4	16.16

2.2.6 Performance Evaluation

The summary of performance evaluation of MAJE4 was carried out by comparing with JEROBOAM stream cipher and is presented in Table 2.5

2.2.6.1 Timing Analysis

From the timing analysis it can be noted that when JEROBOAM 128-bit and MAJE4 128-bit are compared, MAJE4 128-bit is almost 9 times faster as shown in Fig.2.2.

Table 2.5: Timing Analysis

PRNGs	Key length	No. of random numbers generated	No. of random bits per each random number	Total no. of bits produced (Mbps)
JEROBOAM	128-bit	26,80,000	16	40.89
MAJE4	128-bit	1,15,39,399	32	352.15
MAJE4	Variable 128-bit	58,34,000	32	178.03
MAJE4	Variable 256-bit	43,99,999	32	134.27

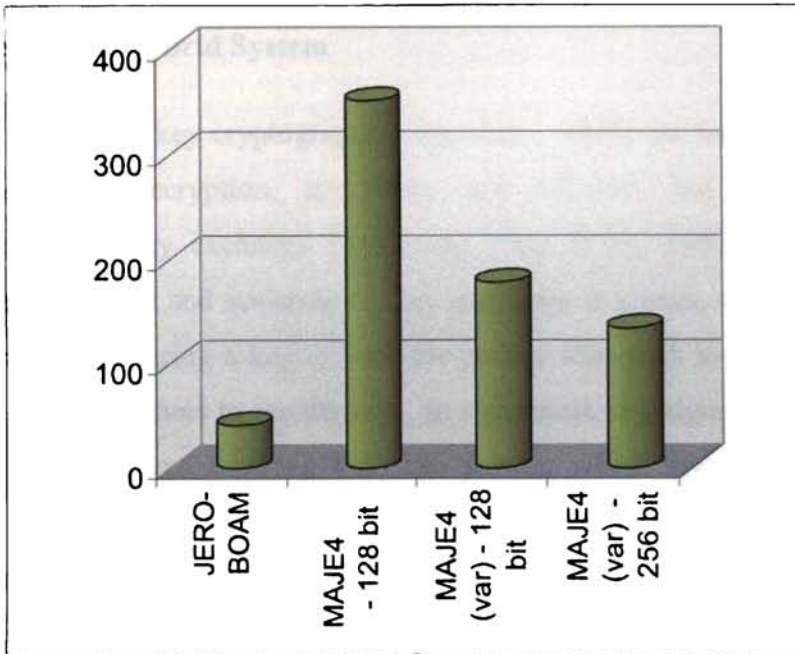


Fig. 2.2: Comparison of Number of Random bits Produced per Second

2.2.6.2 Memory Requirements

On comparing the memory required for executable files of JEROBOAM 128-bit and MAJE4 128-bit, MAJE4 was found consuming lesser space compared to JEROBOAM. The memory size for JEROBOAM, MAJE4 128-bit and MAJE4 128 / 256 bits are 6341 bytes, 5435 bytes and 5678 bytes respectively.

2.3 Development of a Hybrid System MARS4 using MAJE4

2.3.1 Need for a Hybrid System

Symmetric key cryptographic algorithms, which use the same key for encryption and decryption, are faster and efficient, but they have a disadvantage in key exchange [Kencheng Zeng et al., 1991], [Mustak E. Yalcin et al., 2004] and scalability. Key exchange is a term, which refers to the means of delivering a key to both the parties who wish to exchange data without allowing others to see the key. In symmetric key algorithm, the same key has to be shared between both the parties, which need a secure key transfer. In asymmetric key algorithm, the public key is known to all and the need for private key transfer doesn't arise. The scalability problem can be explained by considering the case of n persons communicating to each other. The number of key pairs required in symmetric key algorithm is $n*(n-1)/2$ [Atul Kahate, 2005], whereas in asymmetric key algorithm it is n key pairs. For example if 1,000 people want to securely communicate with each other, only 1,000 public keys and the corresponding private keys are required in asymmetric key algorithm. This is in severe contrast to the symmetric key operation where 1,000 participants need 499,500 key pairs, thus leading to a scalability problem. Asymmetric key cryptographic algorithms not only solve the major problem of key exchange and scalability but also achieve the purpose of non-repudiation [Fujisaki E. and Okamoto T, 1999a], [Williams H.C., 1980], [Bellare M. and Rogaway P, 1995]. The dawn of asymmetric key cryptography does not indicate the end of symmetric key cryptography. In practice, the symmetric key and asymmetric key systems are not in competition. Most cryptographic schemes on which e-commerce operations

rely use a hybrid of these systems. Here the asymmetric key system is used for the distribution of a secret key, which can be a long-term key or specific to a particular communication session. Then the securely distributed secret key is used to encrypt and decrypt messages in a communication channel between two users. The performance of secret key cryptography over that of asymmetric key, and the appeal of key distribution inherent to asymmetric key cryptography, are the main reasons for the wide adoption of these hybrid systems [Fujisaki E. and Okamoto T, 1999b].

As shown in Fig. 2.3 the plain text is encrypted with the fast symmetric encryption algorithm MAJE4 and symmetric key K1 to form the cipher text and then the symmetric key K1 of MAJE4 is encrypted with public key K2 of asymmetric algorithm RSA [Park Stephen K. and Keith W. Miller, 1988]. Then the cipher text and encrypted symmetric key K1 are sent together by the sender to the receiver. In the receiver side, first RSA algorithm is run with its private key K3 to recover the symmetric key K1. Then by using K1 and MAJE4 the entire cipher text is converted into plain text.

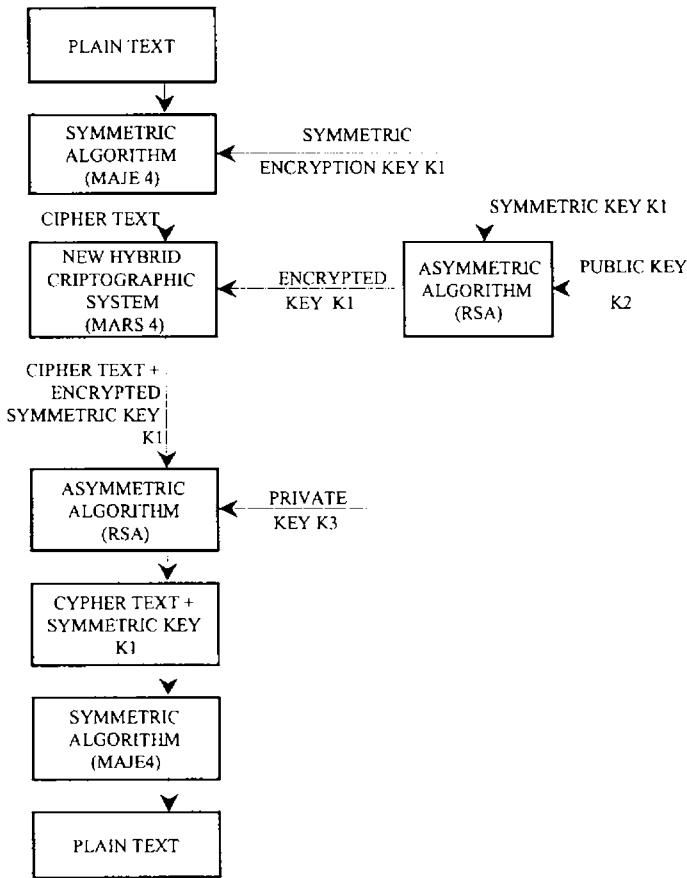


Fig. 2.3 A Novel Hybrid Cryptographic System: MARS4

2.3.2 Objectives for MARS4

The following objectives are considered while combining the two cryptographic algorithms with a view to obtaining the merits of both the systems.

1. The method should be completely secure.
2. The encryption / decryption process should not take longer time.
3. The generated cipher text should be compact in size.
4. The solution should scale to a large number of users easily, without introducing any additional complications.
5. The key exchange problem should be solved by the new method.

2.3.3 Description of MARS4

All the above-mentioned objectives were considered while proposing MARS4. The main features of MAJE4 are explained in section 2.2.2. The following are the main features of RSA and MARS4.

RSA

Main features

1. RSA is computationally easy for a party B to generate the key pair (Public key KS_b , Private key KR_b).
2. It is computationally easy for a sender A, knowing the public key KS_b and the message to be encrypted M , to generate the cipher text $C = E_{KS_b}(M)$.
3. It is computationally easy for the receiver B, to decrypt the resulting cipher text using the private key to recover the original message $M = D_{KR_b}(C) = D_{KR_b}[E_{KS_b}(M)]$
4. It is computationally infeasible for an opponent, knowing the public key KS_b alone to determine the private key KR_b .

5. It is also computationally infeasible for an opponent, knowing the public key K_{S_b} and a cipher text C , to recover the original message M .
6. The encryption and decryption functions can be applied in either order. $M = D_{K_{R_b}}[E_{K_{S_b}}(M)] = D_{K_{S_b}}[E_{K_{R_b}}(M)]$

Algorithm 2.2: RSA

Step 1: Choose two large prime numbers P and Q .

Step 2: Calculate $N = P * Q$.

Step 3: Select the public key (encryption key) E such that it is not a factor of $(P-1)$ and $(Q-1)$.

Step 4: Select the private key (decryption key) D such that the following equation is true:

$$(D * E) \bmod (P-1) * (Q-1) = 1$$

Step 5: Encrypt the plain text PT to form the cipher text CT as follows

$$CT = PT^E \bmod N$$

Step 6: Send CT as the cipher text to the receiver.

Step 7: Decrypt the cipher text CT to form the plain text PT as follows

$$PT = CT^D \bmod N$$

The crux of RSA is that factoring N to find P and Q is not at all easy but it is quite complex and time consuming.

MARS4

MAJE4 and RSA can be combined to have MARS4 as a very efficient security solution. Assume that A is the sender of a message and B is the receiver. MARS4 works as follows.

Algorithm 2.3: MARS4

- Step 1: A encrypts the original plain text message (PT) with the help of MAJE4 and the symmetric key (K1) and forms the cipher text (CT).
- Step 2: Encrypt K1 with the public key (K2) of B using RSA.
- Step 3: Attach the encrypted K1 to the CT and send it to B.
- Step 4: B now uses the RSA algorithm and its private key (K3) to decrypt K1.
- Step 5: Then B uses K1 and the MAJE4 algorithm to decrypt the CT for yielding the original plain text (PT).

As specified in the objectives of MARS4 in section 2.3.2, the symmetric key algorithm MAJE4 is faster and it can produce 352 Mbps. Also the generated cipher text is of the same size as the plain text. Instead, if we had used the asymmetric key encryption as in RSA, then the operation would have been quite slow, especially if the plain text was of larger size. Also the cipher text produced is of larger size than the size of the plain text. Now since the encryption of only 128-bit private key is done, RSA encryption process would not take too long and the encrypted key will not consume more space also. This feature of RSA is used for solving the major problem of key exchange. MARS4 is thus having the advantages of both MAJE4 and RSA.

2.3.4. Results

Tables 2.6 to 2.10 show the results of MAJE4, RSA and MARS4 run with plain text of different sizes. The memory sizes of the plain text to be encrypted as well as the cipher text, the time taken for encryption and decryption are shown in these tables.

Table 2.6: Time Taken for Encryption or Decryption of Files of Various Sizes using MAJE4

File size of plain text (bytes)	File size of cipher text (bytes)	Time taken (Sec.)		
		Encryption	Decryption	Total
30144	30144	0.01	0.01	0.02
60003	60003	0.02	0.02	0.04
90070	90070	0.03	0.03	0.06
120014	120014	0.04	0.04	0.08
150060	150060	0.05	0.05	0.10

Table 2.7: Time Taken for Encryption or Decryption of Files of Various Sizes using RSA (N=187)

File size of plain text (bytes)	File size of cipher text (bytes)	Time taken (Sec.)		
		Encryption	Decryption	Total
30144	101336	0.03	0.04	0.07
60003	201672	0.06	0.09	0.15
90070	302665	0.09	0.14	0.23
120014	403183	0.12	0.19	0.31
150060	504041	0.15	0.24	0.39

Table 2.8: Time Taken for Encryption or Decryption of Files of Various Sizes using RSA (N=3431)

File size of plain text (bytes)	File size of cipher text (bytes)	Time taken (Sec.)		
		Encryption	Decryption	Total
30144	140080	0.05	0.07	0.12
60003	278891	0.10	0.14	0.24
90070	418769	0.15	0.22	0.37
120014	557735	0.20	0.29	0.49
150060	697224	0.25	0.36	0.61

Table 2.9: Time Taken for Encryption or Decryption of Files of Various Sizes using RSA (N=44377)

File size of plain text (bytes)	File size of cipher text (bytes)	Time taken (Sec.)		
		Encryption	Decryption	Total
30144	166888	0.03	0.09	0.12
60003	332224	0.06	0.18	0.24
90070	498816	0.09	0.27	0.36
120014	664584	0.12	0.36	0.48
150060	830967	0.15	0.46	0.61

Table 2.10: Time Taken for Encryption or Decryption of Files of Various Sizes using MARS4

File size of plain text (bytes)	File size of cipher text + key (bytes)	Time taken (Sec.)		
		Encryption	Decryption	Total
30144	30262	0.01	0.01	0.02
60003	60121	0.02	0.02	0.04
90070	90188	0.03	0.03	0.06
120014	120132	0.04	0.04	0.08
150060	150178	0.05	0.05	0.10

2.3.5 Performance Evaluation

The performance evaluation was done by comparing the time taken and the memory required for encryption and decryption using MAJE4, RSA and MARS4 algorithms.

2.3.5.1 Timing Analysis

As shown in Table 2.11, the time consumed for the MAJE4 symmetric algorithm and the MARS4 hybrid algorithm can be seen as the same. Hence the advantage of symmetric algorithm, which is the speed of encryption and decryption, is preserved in the hybrid system also. Whereas RSA 8-bit algorithm is consuming 0.07 seconds, RSA 12-bit and 16-bit are taking 0.12 seconds each. RSA 12-bit and RSA 16-bit are taking the same time since the data is processed byte by byte. There is a difference of 0.05 seconds between RSA 8-bit and RSA 16-bit. On considering this difference and estimating the time required for RSA 128-bit, the time required is obtained as 0.82 seconds. Thus MARS4 is found to be 41 times faster than RSA.

Table 2.11: Time Required for Encryption and Decryption using MAJE4, RSA, and MARS4.

Algorithm	Key length	File size of plain text (bytes)	Time taken (sec.)
MAJE4	128-bit	30144	0.02
RSA	8-bit	30144	0.07
RSA	12-bit	30144	0.12
RSA	16-bit	30144	0.12
MARS4	128-bit	30144	0.02

2.3.5.2 Memory Requirements

When the memory requirements for the cipher text of MAJE4 and MARS4 are compared as shown in Table 2.12, both are found consuming almost the same amount of memory. In RSA 8-bit the memory requirement is almost 4 times greater than that of MARS4 and in RSA 12-bit, it is 5 times greater. Also RSA 16-bit is 6 times greater in size than MARS4. Hence for each additional 4-bit key in RSA, the memory size can be found increasing by about the memory size of the given plain text as shown in Fig. 2.4. Thus if RSA 128-bit key is used, the memory size will be about 34 times greater than that of MARS4.

Table 2.12: Memory Requirements for MAJE4, RSA and MARS4

Algorithm	Key length	File size of plain text	File size of cipher text
MAJE4	128-bit	30144	30144
RSA	8-bit	30144	101336
RSA	12-bit	30144	140080
RSA	16-bit	30144	166888
MARS4	128-bit	30144	30262

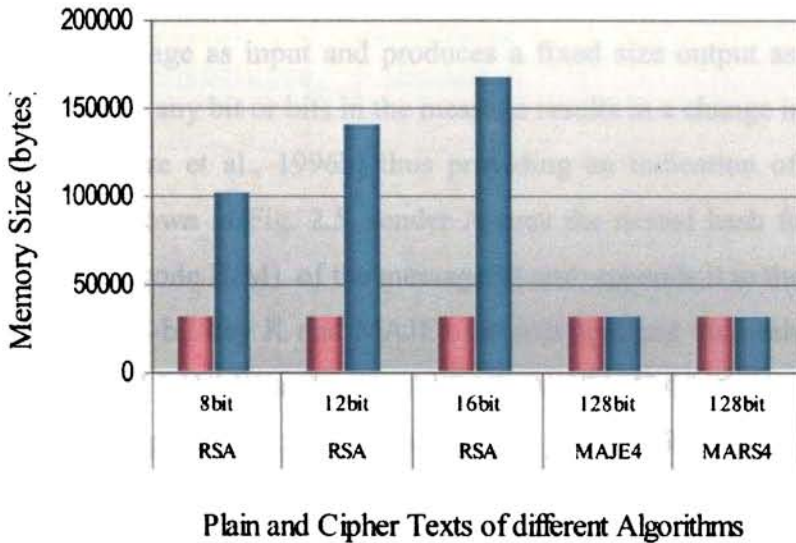


Fig. 2.4: A Comparison of Memory Sizes for MAJE4, RSA and MARS4

2.4 Message Integrity Enhancement of Nested Hash Functions using MAJE4

2.4.1 Introduction

MAJE4 has been proven to be an effective algorithm for providing confidentiality, in the previous sections. The integrity of a message is another concern before the network security service. As a different application of MAJE4 in providing message integrity using nested hash functions, encryption of the hash code with MAJE4 has been considered.

Hash function is a function of all the bits of the message. It accepts a variable size message as input and produces a fixed size output as the hash code. A change in any bit or bits in the message results in a change in the hash code [Mihir Bellare et al., 1996b] thus providing an indication of message tampering. As shown in Fig. 2.5, sender A uses the nested hash function to compute the hash code $H(M)$ of the message M and appends it to the message M . Using the 128-bit key K and MAJE4 the message and the hash code are encrypted as $E_k[M \parallel H(M)]$ and sent to the receiver B. Using the same key K and MAJE4 the cipher text is decrypted back to produce the message and the hash code. Now B re-computes the hash code of the received message using the same nested hash function. Thus B validates the integrity of the message by comparing the hash code received from A with that generated by B. If both hash codes are same then the transmission has happened securely.

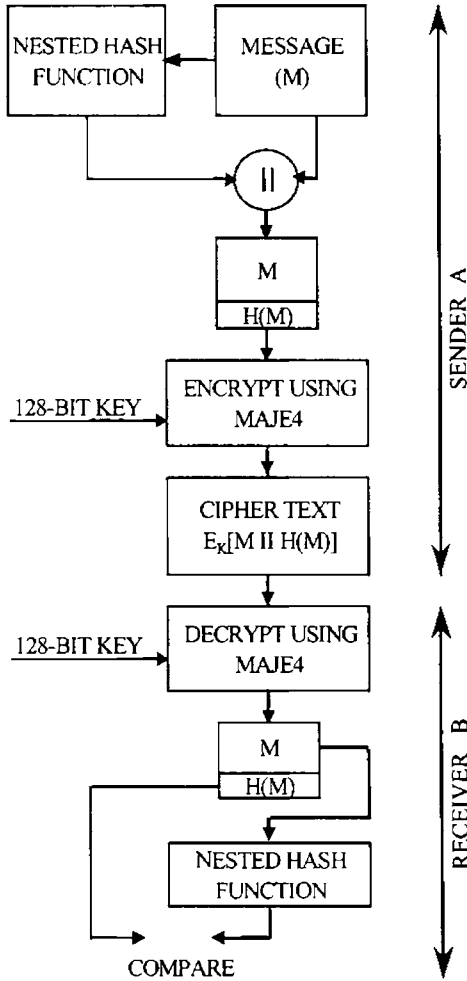


Fig. 2.5: Use of Combined Hash Code and Encryption

2.4.2 Nested Hash Functions

The Merkle-Damgaard model is a good one for the design of hash functions [Ivan Damgard, 1990], [Ralph C. Merkle, 1990]. This model simplifies the management of large inputs and the production of a fixed length output using a function F . The message is viewed as a collection of m -bit blocks. $M = M[1], \dots, M[n]$ with $M[i] = m$ bits for $i = 1, 2, \dots, n$. Assume the length $|M|$ of M as a multiple of m bits, which can be achieved by a suitable padding. Enough number of zeros is added to bring the length of message to multiple of m bits. The blocks are then processed sequentially using the function F . The result of the hash function till then and the current message block are taken as the inputs. This operation is repeated over the entire message blocks to find the hash code of the message M . Algorithm 2.4 is used to compute the hash code.

Algorithm 2.4: Nested Hash Function

- | | |
|---------|--|
| Step 1: | The message is viewed as a collection of 64-bit blocks. $M = M[1], M[2], \dots, M[n]$ with $M[i] = 64$ -bit, for $i = 1, 2, \dots, n$. |
| Step 2: | Check whether the length $ M $ is a multiple of 64 bits and whether n is an even number. If not, suitably append enough zeros to bring the length to a multiple of 64 bits and to make n even. |
| Step 3: | Apply the first function F_1 which is the add operation to the consecutive blocks. ($MB[1] = M[1] + M[2]$, $MB[2] = M[3] + M[4]$ and so on till $MB[n/2] = MB[n-1] + MB[n]$.) |
| Step 4: | Apply the second function F_2 which is an XOR operation, to the |

random initial value and to $MB[1]$ and form the initial hash code. Then F_2 is applied again to the initial hash code and to $MB[2]$ to form the next hash code and so on. Finally apply F_2 on the result of the hash code obtained so far and to $MB[n/2]$ to form the final hash code $H(M)$ of 64-bit length.

Step 5: Now $H(M)$ is added with M as the hash code.

Fig. 2.6 represents the algorithm 2.4. The random initial value used in step 4 provides protection to the hashing process to compute the hash code of the initial message block. The recipient can verify that the message is unaltered by using the same random initial value, which was used to compute the hash code of the message. If these hash codes match, then the message is believed to have arrived unchanged from the sender. Thus the initial random value prevents attackers from making undetectable changes to the message. Message of any length can be considered as the input while the output hash code is of fixed 64-bit length. The initial value used as K in the equation, $H(M) = F_2K(F_1(M))$ is random and hence the attackers will not be able to predict the initial value easily. The functions F_1 and F_2 are ADD and XOR operations [Mihir Bellare et al., 1995] which are easy to implement both in hardware and software. At the same time the nested usage of operators $+$ and \wedge complicates cryptanalysis. The security of the hash code mainly depends on the cryptographic properties of the hash function H . Here the non-linearity is obtained when functions F_1 and F_2 are nested. This provides added security.

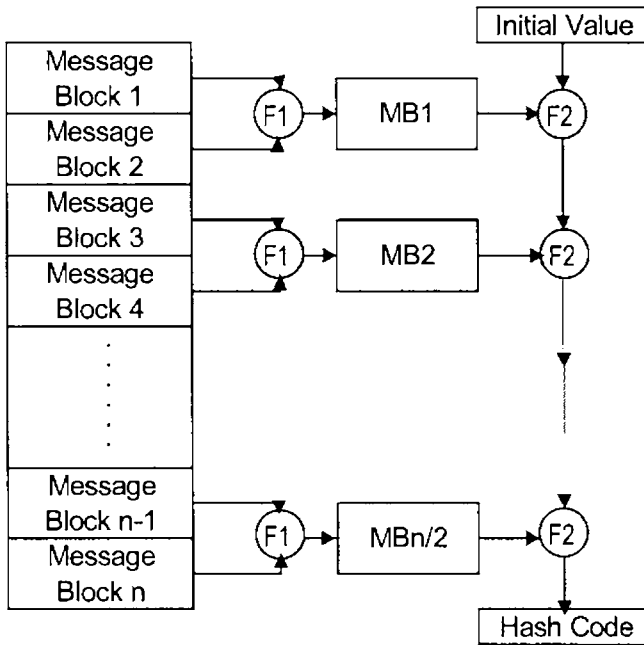


Fig. 2.6: Model of a Nested Hash Function

It is also observed that the length of a hash code in bits is directly related to the number of trials that an attacker has to perform before a message is accepted. For a hash code value of bit length m , the attacker has to perform on average 2^{m-1} random hash code verifications.

2.4.3 Use of Hash Code and MAJE4

The following are the steps performed for obtaining the confidentiality and integrity using MAJE4 and nested hash function.

1. Sender encrypts the message M as well as the hash code $H(M)$ using 128-bit key K and the fast stream cipher algorithm MAJE4, then sends it to the receiver.
2. Receiver decrypts the message as well as the hash code using the same 128-bit key K and MAJE4 algorithm.
3. Receiver re-computes the hash code $H(M)$ over the message M and checks whether it matches with the received hash code.
4. If it matches with the hash code, then the message can be considered to have reached securely. Otherwise it can be understood that some distortion has happened.

2.4.4 Results

Tables 2.13 to 2.15 show the results of time requirements for MAJE4 and nested hash code when run with plain texts of different sizes. The memory sizes of the plain text to be encrypted, the cipher text, the time taken for encryption and decryption and the time taken for producing the hash code are given.

Table 2.13: Time Taken for Encryption or Decryption of Files of Various Sizes using MAJE4

File size of plain text (bytes)	File size of cipher text (bytes)	Time taken (Sec.)	
		Encryption	Decryption
135094	135094	0.05	0.05
270728	270728	0.10	0.10
541608	541608	0.20	0.20
812440	812440	0.30	0.30
1082953	1082953	0.40	0.40

Table 2.14: Time Taken for Producing the Hash Code of Files of Various Sizes using Nested Hash Function

File size of plain text (bytes)	Time taken for producing the hash code (Sec.)
135094	0.01
270728	0.02
541608	0.04
812440	0.06
1082953	0.08

Table 2.15: Total Time Taken for Producing the Hash Code and Encryption/Decryption of Files of Various Sizes using Nested Hash Function and MAJE4.

File size of plain text (bytes)	Total time taken for producing and recomputing hash code (Sec.)	Total time taken for encryption and decryption (Sec.)
135094	0.02	0.10
270728	0.04	0.20
541608	0.08	0.40
812440	0.12	0.60
1082953	0.16	0.80

In Table 2.15 it can be seen that only $1/5^{\text{th}}$ of additional time is required for producing the hash code along with encryption and decryption. More over if the message size is reasonably small or up to about 135 kilobytes then the time taken for producing the hash code is negligible. For large messages the additional time requirement is very less as shown in Fig. 2.7. The memory size required for executable code for nested hash code is 5899 bytes and for MAJE4 it is 5435 bytes. Hence a total of less than 12 Kilobytes memory is enough for providing both integrity and confidentiality of messages.

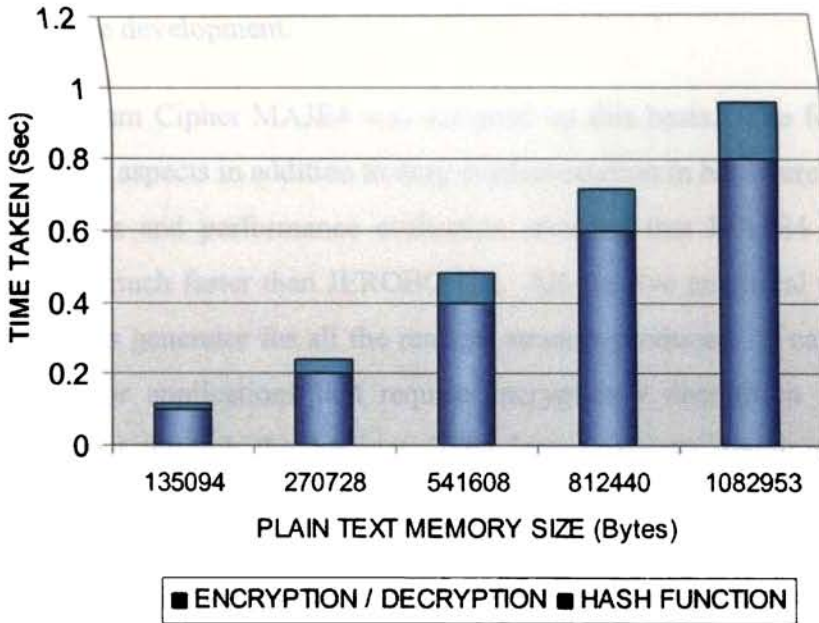


Fig. 2.7: Total Time Taken for Hash Code Generation & Verification and Encryption / Decryption

2.5 Conclusions

Popular PRNGs and Stream Ciphers were taken up for studies based on statistical analysis using five randomness tests. Results of analysis and further performance evaluation studies showed that JEROBOAM and RC4 are dependable as they passed all the five randomness tests. LFSR and Geffe passed or failed the tests depending upon the input polynomial, initial seed, number of bits taken from each random number etc. as mentioned in 2.1.5.1. LCG and $X^2 \bmod N$ failed for autocorrelation test. Among these, JEROBOAM is not reported to have undergone attacks, whereas RC4, LFSR, Geffe, LCG and $X^2 \bmod N$ generators have undergone attacks. Hence it was

concluded to make use of the intrinsic qualities of JEROBOAM as basic guidelines for future development.

A new stream Cipher MAJE4 was designed on this basis. The focus was on the security aspects in addition to easy implementation in hardware and software. Analysis and performance evaluation revealed that MAJE4 is a reliable generator much faster than JEROBOAM. All the five empirical tests were passed by this generator for all the random streams produced. It can be effectively used for applications that require encryption / decryption of a stream of data sent through the Internet. The lesser memory requirement makes it suitable for devices with limited memory.

Exploration of different applications for MAJE4 gave birth to MARS4, a hybrid cryptographic system. Performance evaluation studies proved MARS4 to be much faster than the popular RSA. The memory requirement for MARS4 is also less than RSA. MARS4 also provides a solution to the key exchange problem seen among symmetric key algorithms. Thus the advantages of both symmetric and asymmetric cryptographic algorithms are preserved in MARS4. It was proven to be a very sound technique for transferring messages where confidentiality is of importance to the users.

Integrity of message is an additional property that can be achieved along with confidentiality. With a very small increase in time, this could be achieved by using nested hash functions. The time required for messages with a memory of upto 135 Kbytes is found negligible. Also the additional memory size needed for implementing the hash function is only 5899 bytes. Nested hash functions with MAJE4 can be used for applications that require message

integrity and encryption / decryption of stream of data sent through the Internet.

When advanced cryptographic systems with lesser memory and good speed are made available, they become easier to implement and manage and more internet users can take advantage of their benefits.

Chapter 3

Studies on Hash Functions and Design & Development of a Novel Hash Function JERIM-320

Abstract:

Providing integrity of messages is one of the main desirable network security services as mentioned in Section 1.4. Use of MAJE4 along with nested hash function was considered in the previous Section as an application of providing integrity in addition to confidentiality. But message integrity is a stand alone service which needs to be ensured for different applications in the network. Hence to enhance security services, studies on existing hash functions have been carried out and a new hash function JERIM-320 with better security has been proposed in this chapter. Performance evaluation has been carried out by comparing with 5 popular hash functions by using practical implementation and also by using step computation methods. This work suggests JERIM-320 as an alternative for the present day hash functions. The randomness property of JERIM-320 is analysed by means of the statistical experiments. This is done to ensure the integrity of messages as well as to generate a long unpredictable key stream with better performance using JERIM-320 and hence to expose it for applications requiring pseudo-random number generations also.

3.1 Study of Hash Functions

In recent years, due to the prospering use of internet applications, ensuring confidentiality, integrity and authenticity of information is becoming an increasingly important issue. At present, most of the current planetary data archives are stored online in rewritable media. As such, the data are vulnerable to accidental changes and deletions as well as intentional changes by virus, trojans and the like. Viruses typically modify the host files that they infect, and hence one way of virus detection involves checking files for signs of unauthorized modification [Raphabel C et al., 2006]. When two persons are communicating over an insecure channel, they need a method by which the original information sent by the sender can be accepted by the receiver without an uncertainty on possible alteration or leakage; this is called ensuring the integrity of message. When a person 'A' sends a message to another person 'B', the hash code computed using the hash function is appended to the message. After receiving the message, 'B' re-computes the hash code using the same hash function and compares with the received hash code. If both are the same, then 'B' can confirm that the message has started off from the intended sender and it has not been tampered with, during the transmission; this is shown in Fig. 3.1.

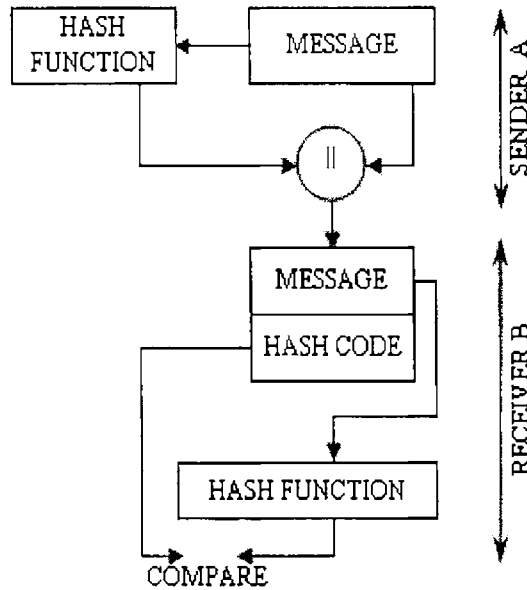


Fig. 3.1: Hash Code Generation and Verification

Hash functions are important components in many cryptographic applications and security protocol suites. The most important uses are in the protection of information authentication and as a tool for digital signature schemes. The succeeding paragraphs present the observations of an overall review of cryptographic hash functions.

1. Desirable properties [William Stallings, 2003] for hash functions:
 - a. Hash functions can be applied to messages of any length, x .
 - b. They produce an output of fixed length.
 - c. For any given x , it is easy to compute $H(x)$ making both the hardware and software implementation easy.
 - d. For any given value h , it is computationally infeasible to find x such that $H(x) = h$. (Preimage resistance) [Erhan K, 2007]

- e. For any given block x , it is computationally infeasible to find $y \neq x$ with $H(y) = H(x)$. (Second preimage resistance) [Erhan K, 2007]
 - f. It is computationally infeasible to find any pair (x, y) such that $H(x) = H(y)$. (Collision resistance). [Erhan K, 2007]
2. Most popular hash functions are designed using Merkle-Damgaard model [Ivan Damgard, 1989], [Ralph C. Merkle, 1989]. This model simplifies the management of large inputs and produces a fixed length output using a function HF. The message is viewed as a collection of m -bit blocks:

$M = M_{\{1\}} \dots M_{\{n\}}$ with $M_{\{i\}} = m$ bits for $i = 1, 2, \dots, n$.

The hash function H can be described as follows:

$HF_0 = IV$; $HF_i = f(HF_{i-1}, M_{\{i\}})$, where $1 \leq i \leq n$;

$H(M) = HF_n$.

Here f is the compression function of H , HF_i is the chaining variable between stage $i-1$ and stage i , and IV denotes the initial chaining value. This iterative construction in the model provides a moderate goal of extending the domain of collision resistant functions. Many hash functions such as MD4 [Rivest R.L., 1990], MD5 [Rivest R.L., 1992] and SHA-family [NIST FIPS-180-2, 2002] are based on this concept.

3.2 Review of Popular Hash Functions

In this section, the hash functions SHA-1, SHA-256, RIPEMD-160, RIPEMD-320 and FORK-256, their similarities and differences are briefly described.

3.2.1 Similarities

The general skeleton of these hash functions shows their similarities and consists of the following steps:

1. Initialization: A few constant values are defined in this step. These constants include initial chaining values (IVs), order of accessing message words, additive constants and the number of bits for rotation in each step.
2. Preprocessing: The message to be hashed has to be of length divisible by 512. Otherwise padding bits are used to append the message, with a single bit of value '1' followed by the required number of 0's. This makes the message length equal to 64 bits less than a multiple of 512-bit blocks, each of which consists of sixteen 32-bit words.
3. Processing: This is the heart of the algorithm, where each 512-bit block is processed in one step. Each step consists of the following sub steps
 - a. Initialize working variables with the current values of the chaining variables.
 - b. Update the working variables using some computation in rounds. Each round has almost the same computation in all its steps.
 - c. Update the chaining variables
4. Completion: The final hash value is composed to form the hash code by updating the chaining variables.

3.2.2 SHA-1

The SHA series of algorithms which stand for ‘Secure Hash Algorithm’ were developed by NIST [NIST FIPS-180-2, 2002]. SHA algorithms are based upon the MD4 and MD5 [Rivest R.L., 1992] algorithms developed by Ron Rivest. SHA was released by the National Security Authority as a US Government Standard in 1993. SHA-0 is commonly known as SHA, it was the first materialization of the secure hashing algorithm. This first version was withdrawn soon after release due to weaknesses in the design. SHA-1 released a couple of years later fixed these problems. It was released in 1995, and is similar to MD4 and MD5 hashing algorithms. It is considered as MD5’s successor and is slightly more secure than MD4 & MD5. SHA-1 is also slower than MD5 and produces a 160-bit hash. The SHA-1 algorithm is featured in a large number of security protocols and applications. Both SHA and SHA-1 produce a hash value of 160 bits.

In hash functions, the given message is divided into 512-bit blocks and again each block is divided into sixteen 32-bit sub blocks. The sixteen 32-bit words M_i , where $0 \leq i \leq 15$, are then linearly expanded into eighty 32-bit words W_i as

$$W_i = \begin{cases} M_i, & \text{for } 0 \leq i \leq 15, \\ W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16} & \text{for } 16 \leq i \leq 79. \end{cases}$$

The state update transformation operates on five 32-bit registers, which are initialized with the current value of the chaining variable. It

consists of 80 steps, divided into 4 rounds of 20 steps each. A single step of the state update transformation is shown in Fig. 3.2. In each step the function f is applied to the state variables B_i , C_i , and D_i . The function f depends on the step numbers: 0 to 19 (round 1) use f_1 and steps 40 to 59 (round 3) use f_2 . f_3 is applied in the remaining steps (round 2 and 4). The functions are defined as:

$$f_1(B, C, D) = (B \wedge C) \vee (\neg B \wedge D)$$

$$f_2(B, C, D) = (B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$$

$$f_3(B, C, D) = B \oplus C \oplus D,$$

where \wedge denotes the logical AND operation, \vee denotes the logical OR operation, \oplus corresponds to addition modulo 2, and $\neg B$ is the bitwise complement of B . The state update transformation also uses step constants K_i .

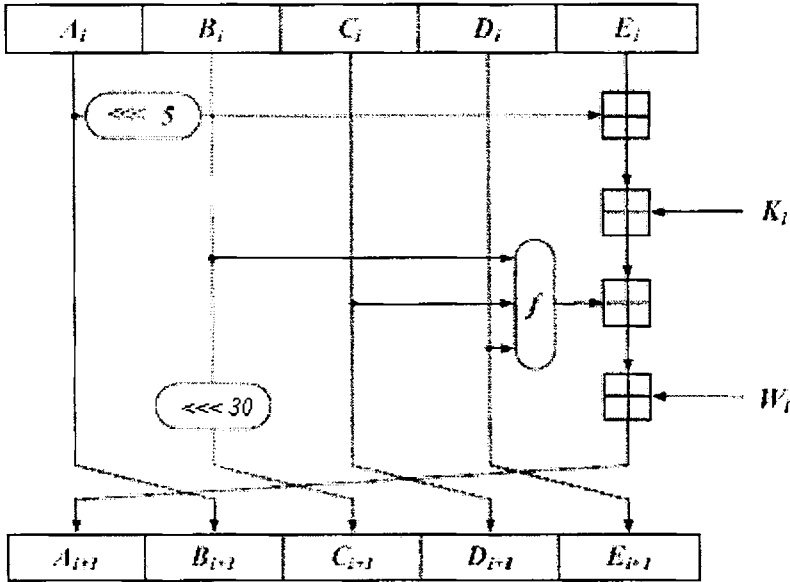


Fig. 3.2: A Single Step Operation of SHA-1

3.2.3 SHA-256

SHA-256 is a part of the SHA-2 family of products having the capability for larger output strings. SHA-256 [NIST FIPS-180-2, 2002] operates on chaining variables of 256 bits. The message expansion takes as input a vector m with 16 words M_i and outputs 64 32-bit words W_i , generated according to the following formula:

$$W_i = \begin{cases} M_i, & \text{for } 0 \leq i \leq 15, \\ \sigma_1(W_{i-2}) + W_{i-7} + \sigma_0(W_{i-15}) + W_{i-16} & \text{for } 16 \leq i \leq 63. \end{cases}$$

The functions $\sigma_0(x)$ and $\sigma_1(x)$ are defined as follows:

$$\sigma_0(x) = ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x)$$

$$\sigma_1(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x)$$

where $ROTR^a$ denotes cyclic rotation by 'a' positions to the right, and SHR^a denotes a logical shift by 'a' positions to the right.

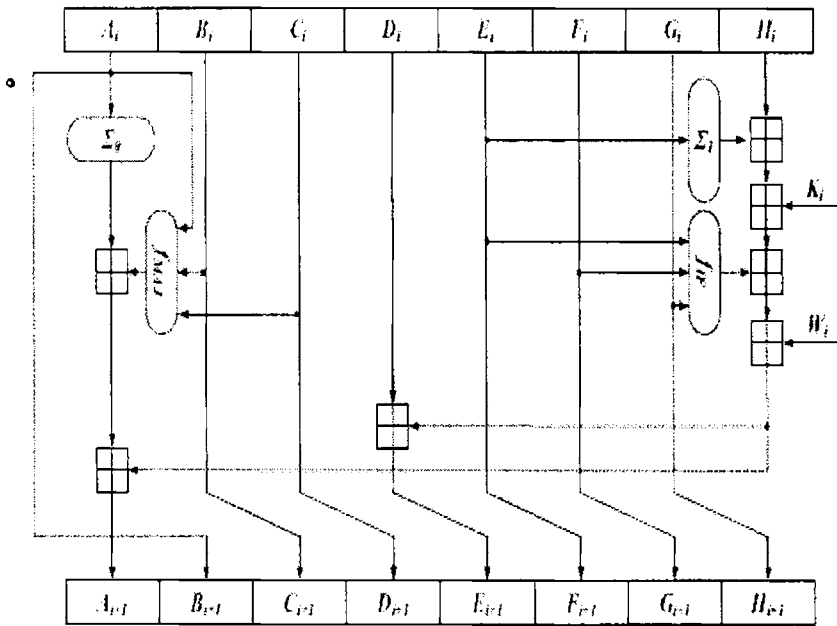


Fig. 3.3: A Single Step Operation of SHA-256

The compression function consists of 64 identical steps. One step is shown in Fig. 3.3. The step transformation employs the bitwise Boolean functions f_{MAJ} and f_{IF} and two linear functions $\Sigma_0(x)$ and $\Sigma_1(x)$ given by:

$$f_{MAJ}(A, B, C) = (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C)$$

$$f_{IF}(E, F, G) = (E \wedge F) \oplus (\neg E \wedge G)$$

$$\Sigma_0(x) = ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x)$$

$$\Sigma_1(x) = ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x).$$

The i -th step uses a fixed constant K_i and the i -th word W_i of the expanded message.

3.2.4 RIPEMD-160

RIPEMD-160 (RACE Integrity Primitives Evaluation Message Digest) is a 160-bit hash function developed in Europe by Hans Dobbertin, Antoon Bosselaers and Bart Preneel, and published in 1996 [Dobbertin H. et al., 1996]. It produces a 160-bit hash value. Like its predecessor RIPEMD, it consists of two parallel streams. While RIPEMD consists of two same parallel streams of MD4, the two streams are designed differently in the case of RIPEMD-160. The message expansion of RIPEMD-160 is a permutation of the 16 message words in each round, where different permutations are used in each round of the left and the right stream. In each stream, 5 rounds of 16 steps each are used to update the five 32-bit registers. Fig. 3.4 shows one step transformation. The step transformation employs five bitwise Boolean functions f_1, \dots, f_5 in each stream:

$$f_1(B, C, D) = B \oplus C \oplus D$$

$$f_2(B, C, D) = (B \wedge C) \vee (\neg B \wedge D)$$

$$f_3(B, C, D) = (B \vee \neg C) \oplus D$$

$$f_4(B,C,D) = (B \wedge D) \vee (C \wedge \neg D)$$

$$f_5(B,C,D) = B \oplus (C \vee \neg D)$$

Where \oplus denotes the bitwise XOR operation, \wedge denotes the logical AND operation, \vee denotes the logical OR operation and \neg denotes the bitwise complement operation. The order of the Boolean function is different in each stream. A constant K_i is added in every step; the constant is different for each round and for each stream. Different rotation values ‘S’ are used in each step and in both streams. After the last step, the initial value and the values of the right and the left streams are combined, resulting in the output of one iteration.

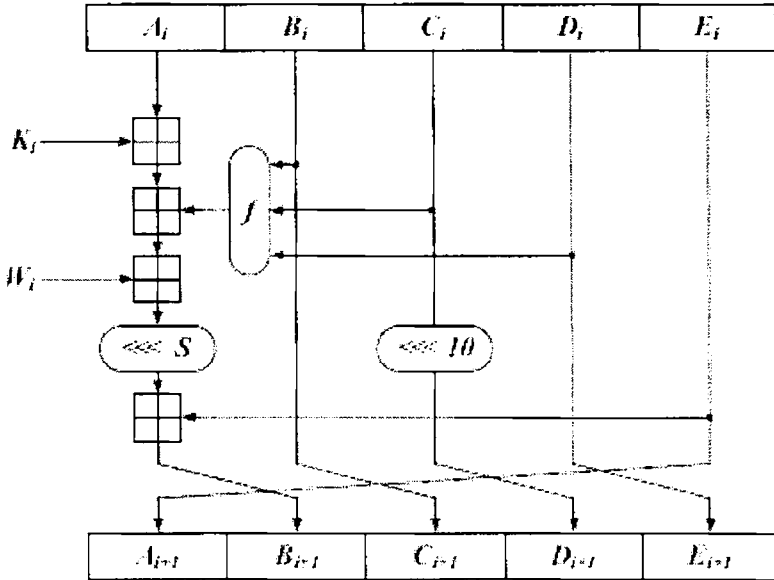


Fig. 3.4: A Single Step Operation of RIPEMD-160

3.2.5 RIPEMD-320

RIPEMD-320 [Dobbertin H. et al., 1996] is an iterative hash function that operates on 32-bit words. This round function takes a 10-word chaining variable and a 16-word message block as inputs and maps them into a new chaining variable. All operations are defined on 32-bit words. This is an extension of RIPEMD-160, and is intended for applications that require a longer hash result without having the need for a larger security level than RIPEMD-160. In short, RIPEMD-320 is a double width string variant of the popular RIPEMD-160.

3.2.6 FORK-256

The design of the FORK-256 hash function is proposed by Hong et al. in [Hong D. et al., 2006]. It follows the iteration principle proposed by Merkle [Ralph C. Merkle, 1989] and Damgard [Ivan Damgard, 1989], and its compression function hashes a 512-bit message block M at each iteration and uses a 256-bit chaining variable CV_n . The name of FORK-256 comes from the fact that the internal state is modified simultaneously in four parallel streams, and the four corresponding outputs h_1, \dots, h_4 are recombined as $h' = (h_1 + h_2) \oplus (h_3 + h_4)$ and produces the output $CV_{n+1} = h' + CV_n$ of the compression function. To process the 512-bit message M with the chaining variable CV_n , M is first divided into sixteen 32-bit words M_0, M_1, \dots, M_{15} . The processing applied in each of the four streams is the same and consists of eight iterations of a step transformation on an internal state. The internal state consists of eight 32-bit words denoted by A, B, C, D, E, F, G and H ,

and the step transformation involves several parameters such as varying constants $\alpha_{j,r}$ and $\beta_{j,r}$, and two 32-bit words M_i and M_j from M . These words M_i and M_j are chosen depending on the stream number and the round number according to the rules of message ordering. Basically, a permutation σ_j is applied in stream j to select the sub-blocks $M_{\sigma_j(2i)}$ and $M_{\sigma_j(2i+1)}$ at round i . The step transformation itself is pictured in Fig. 3.5. The two non-linear functions f and g used in each step are defined as follows:

$$f(x) = x + (x \lll 7 \oplus x \lll 22), \quad g(x) = x \oplus (x \lll 13 + x \lll 27)$$

Here ' \oplus ' denotes bitwise XOR, '+' denotes integer addition, and $w \lll k$ denotes the word w cyclically rotated by k bit positions to the left.

In Fig.3.5, $A_{j,r}, \dots, H_{j,r}$, words of the internal registers of stream j are shown, after step r . The words A_0, \dots, H_0 , denote the common initial state of the registers. There are eight rounds in each of the four streams.

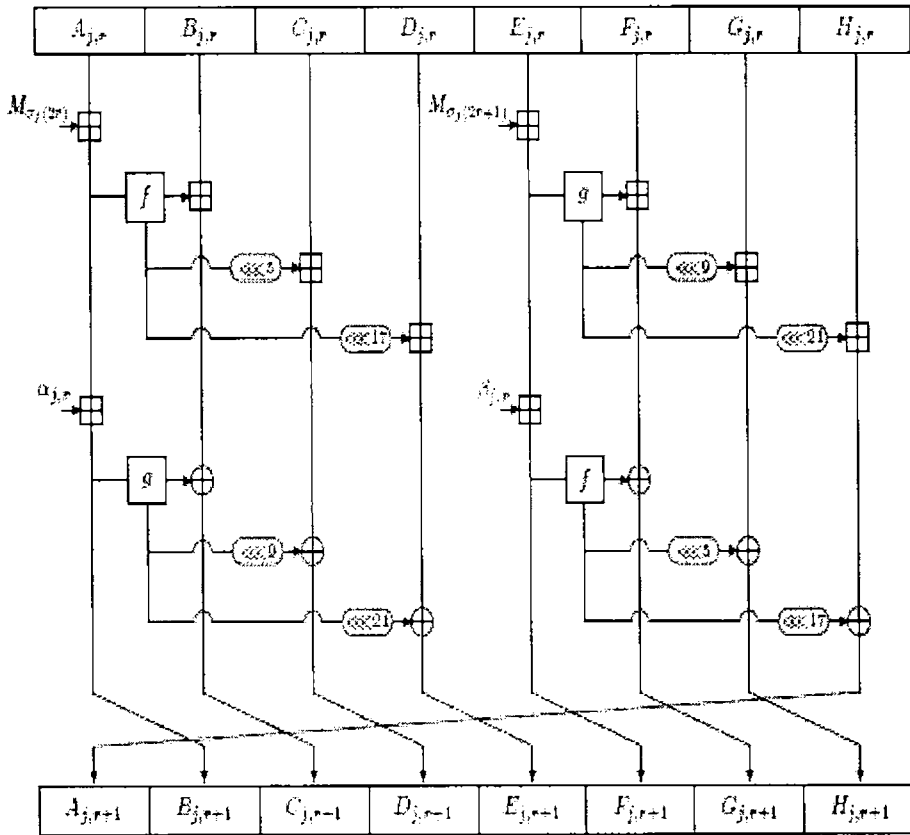


Fig. 3.5: A Single Step Operation of FORK-256

3.2.7 Differences

The main differences between the five hash functions SHA-1, SHA-256, RIPEMD-160, RIPEMD-320 and FORK-256 are summarized as follows:

1. Sixteen 32-bit message words are used as 80 words in SHA-1 and 64 words in SHA-256. In each compression step, one word is then referred in

a serial fashion. But RIPEMD-160, RIPEMD-320 and FORK-256 use only 16 words and in different accessing orders.

2. For each 512-bit block, SHA-1 has 4 rounds of 20 steps each, SHA-256 has 4 rounds of 16 steps each, and RIPEMD-160 and RIPEMD-320 have 5 rounds each of 16 steps and FORK-256 has single round with 8 steps.
3. Working variables in the five hash functions are not updated in the same way in compression steps.
4. RIPEMD-160 and RIPEMD-320 use two parallel processing streams while FORK-256 uses four parallel processing streams for each 512-bit message block. Only one processing stream each is used by the other two.
5. After processing each 512-bit message block, SHA-1, SHA-256, RIPEMD-320 and FORK-256 update the chaining variables in the same way, whereas RIPEMD-160 uses a different way to update them.
6. The final hash values of SHA-1 and RIPEMD-160 are 160-bit long; for SHA-256 and FORK-256 the values are 256-bit long; whereas for RIPEMD-320, it is 320-bit long.

3.3 Design of a Novel Hash Function: JERIM-320

3.3.1 Motivation and Design Factors

The successful use of cryptographic algorithms for detection of file tampering lies in the fact that any small change in the source file should result in a significant change in the hash code. MD5, SHA-1 and RIPEMD

algorithms are popularly used for generating hash codes. But these algorithms have been broken at various levels [Biham E. et al., 2005], [Chabaud F. and Joux A., 1998], [Dobbertin H., 1996], [Biham E. and Chen R., 2004]. Collisions in the hash code have proved that a file may be modified without a corresponding change in the hash code. Generally a function which has a good diffusion property [Coskun B. and Memon N., 2006] cannot be so light, but most step functions have been developed to be light for efficiency. This is why MD4 type hash functions including SHA-1 are vulnerable to Wang et. al.'s collision finding attacks [Xiaoyun Wang and Hongbo Yu, 2005]. If a longer hash function such as RIPEMD-320 or SHA-512 [NIST FIPS-180-2, 2004] is used, the collisions are less likely and the benefits of greater security supersede the computational compromise of the longer hash function.

The SHA-2 [Philip Hawkes, et al., 2004] hash functions are quite resistant against those attack techniques which have been used to attack MD4, MD5 and SHA-1. The SHA-2 functions are possible short term alternatives to SHA-1. No attacks against SHA-2 functions have been noticed.

An alternative to this is RIPEMD-family [Dobbertin H., 1996], which has a different design approach for providing secure hash code. The attacker who tries to break members of RIPEMD family should try simultaneously at two ways where the message difference passes. This design strategy is still considered successful in so far as no effective attack on RIPEMD family has been reported except the first proposal of RIPEMD. The RIPEMD family has heavier hash functions compared to MD4 family. For example, the first proposal of RIPEMD consists of two lines of MD4. The number of steps in

RIPMD-160 is almost same as that in SHA-0. No specific attack against RIPMD-160 or RIPMD-320 has been reported.

As a result of a large number of attacks on hash functions such as MD5 and SHA-1 of the so called MD4 family, there is an increasing need for developing alternate designs based on new principles for future hash functions. Several attacks on hash functions are focused on alleviating the difference of intermediate values which are caused by the difference in the message. In this context, a hash function can be considered secure, if it is computationally hard to alleviate such difference in its compression function.

Based on these considerations a new hash algorithm JERIM-320 has been designed. In the design criteria, more emphasis is given to security over speed because of the practically negligible effect of increase in the time requirement even though it is also considered as one of the measures of performance. The efficiency of the new hash function is its design based on potential parallelism.

The properties envisaged during the design of JERIM-320 are:

- a. It should be highly secure
- b. It should have a higher hash length to resist against the birthday attack [Wagner D., 2002].
- c. It should have a structure resistant to all known attacks including Wang et. al's attack [Xiaoyun Wang and Hongbo Yu, 2005].

- d. It should have a reasonable performance with respect to speed of operation.

The size of the hash value and that of the intermediate state are selected as 320 bits. This value has been chosen for the following reasons:

- a. Since 32-bit words are generally used; the size should be a multiple of 32.
- b. Most of the successful shortcut attacks on existing hash functions are found to be at the intermediate state rather than at the final value. The attacker typically chooses two colliding values for an intermediate block, and is propagated to a collision of the full function. But, these attacks would not have been successful, if the intermediate values were larger.

3.3.2 Description of JERIM-320

The basic notations used in JERIM-320 are shown in Table 3.1.

Table 3.1: Basic Notations in JERIM-320

Notation	Description
$X \wedge Y$	Addition of X and Y modulo 2 (XOR)
$X + Y$	Addition of X and Y
$X \vee Y$	Bitwise OR operation of X and Y
$X \wedge Y$	Bitwise AND operation of X and Y
$\neg X$	Bitwise NOT operation of X
$X \lll n$	Bit-rotation of X by n bits to the left

3.3.2.1 Input Block Length and Padding

An input message is processed as 512-bit blocks. Padding is used to make the length of the original message equal to a value which is 64 bits less than the exact multiple of 512 bits. The padding consists of a single 1-bit, followed by as many 0-bits, as required. After padding, the original length of the message is calculated and added at the end of the message as a 64-bit value. In the case of very long message, the length of the message is calculated as the original message length modulo 2^{64} .

3.3.2.2 Structure of JERIM-320

Fig. 3.6 shows the outline of the compression function of JERIM-320. It consists of four parallel branches B1, B2, B3 and B4. The initial chaining variable CV_i is given as input to the compression functions. CV_i consists of 10 registers A,B,C,D,E,F,G,H,I and J. These chaining variables in each branch are initialized as given below.

$$A1 = A2 = A3 = A4 = 0xb54ff53a$$

$$B1 = B2 = B3 = B4 = 0x67452801$$

$$C1 = C2 = C3 = C4 = 0xabcadb84$$

$$D1 = D2 = D3 = D4 = 0xc2d3e0f1$$

$$E1 = E2 = E3 = E4 = 0x2e72d96c$$

$$F1 = F2 = F3 = F4 = 0x4ab0cd91$$

$$G1 = G2 = G3 = G4 = 0x9a056873$$

$$H1 = H2 = H3 = H4 = 0x5ca28c67$$

$$I1 = I2 = I3 = I4 = 0xa14fe235$$

$$J1 = J2 = J3 = J4 = 0x863d421c$$

Each successive 512-bit message block M is divided into sixteen 32-bit sub blocks M_0, M_1, \dots, M_{15} as $\Sigma_i(M)$ and given as input to all four branches. The following computation is done to update CV_i to CV_{i+1} as $CV_{i+1} = CV_i \wedge ((B1output \wedge B2output) + (B3output \wedge B4output))$. Finally the message is transformed into the 320-bit hash value.

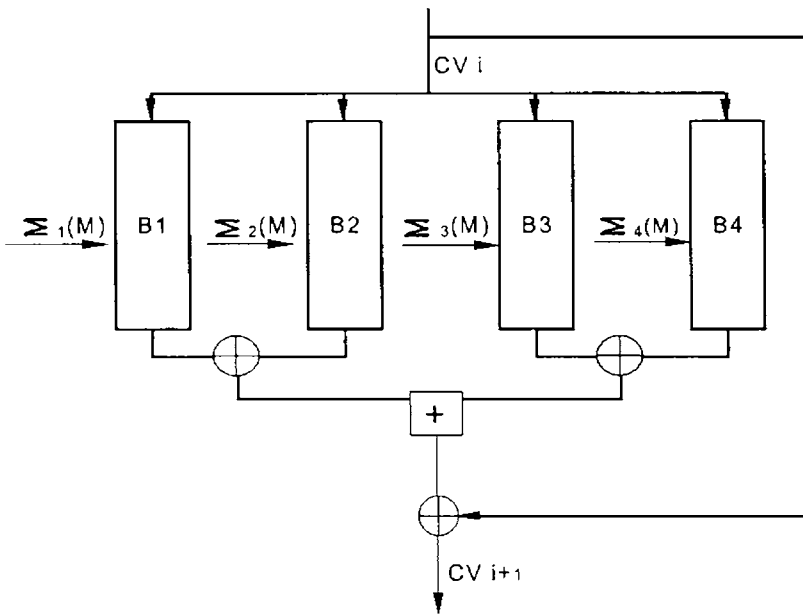


Fig. 3.6: Outline of the Compression Functions of JERIM-320

3.3.2.3 Single Step Operations

Five rounds are used in JERIM-320 for each 512-bit message block. The sixteen 32-bit sub blocks of the 512-bit block in each round are processed in four parallel branches. The inputs to each single step operations are the

sixteen sub blocks, the chaining variables $A_1, B_1, \dots, J_1, A_2, B_2, \dots, J_2, A_3, B_3, \dots, J_3, A_4, B_4, \dots, J_4$ of each branch and the constants $K_{[t]}$. Order of message words in each branch and each round is shown in Table 3.2 and Table 3.3. Shift values, Boolean functions and Constants in each branch and each round are shown in Table 3.4, Table 3.6 and Table 3.7 respectively. There are 16 single step iterations in each round and in all the four branches as shown in Fig. 3.7. The output of each iteration is copied again into the chaining variables $A_1, B_1, \dots, J_1; A_2, B_2, \dots, J_2; A_3, B_3, \dots, J_3; A_4, B_4, \dots, J_4$ and so on.

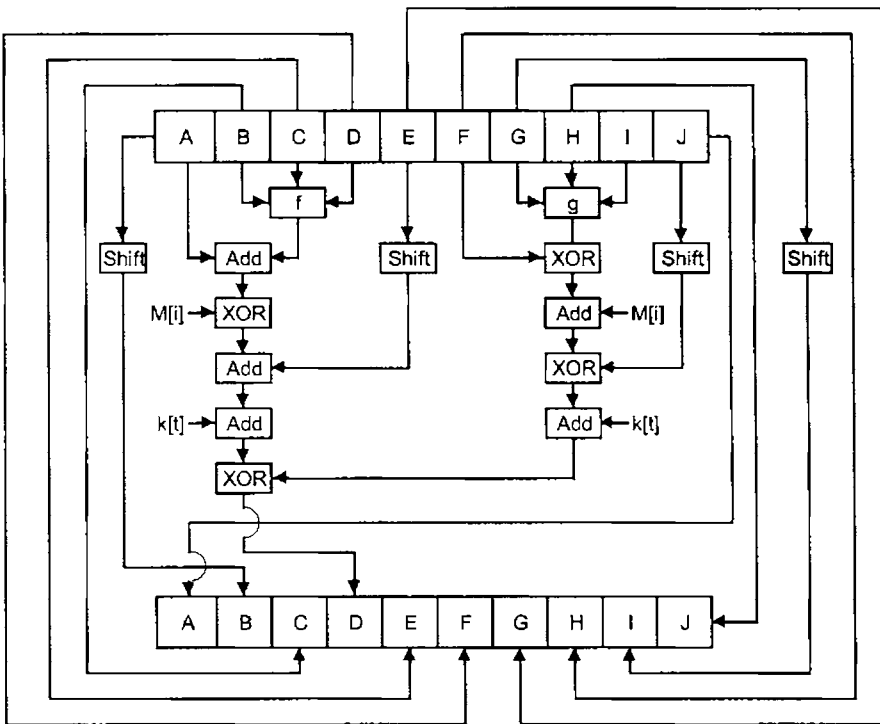


Fig. 3.7: A Single Step Operation of JERIM-320

3.3.2.4 Order of the Message Words

The order in which the blocks are combined is important to prevent the collisions. In order to resist Wang et. al attack, different message orders have been used in different branches as shown in Table 3.2.

Table 3.2: Order Rule of Message Words in Different Branches

I	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
B1 _(i)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
B2 _(i)	15	12	8	10	9	14	11	13	4	3	7	5	6	1	0	2
B3 _(i)	7	14	15	2	11	1	5	10	9	0	3	4	12	8	13	6
B4 _(i)	4	8	3	12	6	9	13	0	1	7	14	15	2	11	5	10

The conditions considered for defining the order of message words [Hong D. et al., 2006] are:

1. Each word is applied twice in the upper and lower parts of the table.
2. Each word is applied twice in the left and right parts of the table.
3. Hence each word is considered 4 times and is indexed by 0 to 15.

To make the attacks more difficult, the order of message in each round for the four branches are considered differently by considering cyclic shift of i as shown in Table 3.3. Each column of its argument is shifted cyclically and independently, so that column i is shifted left by i positions.

Table 3.3: Message Order in Different Rounds

Round	Message order using cyclic shift of i
Round1	$i=0$ to 15
Round2	$i=3$ to 15, $i=0$ to 2
Round3	$i=7$ to 15, $i=0$ to 6
Round4	$i=9$ to 15, $i=0$ to 8
Round5	$i=13$ to 15, $i=0$ to 12

3.3.2.5 Shifts

The variable shift values as shown in Table 3.4 provide better immunity against attacks such as differential collision [Chabaud F. and Joux A., 1998]. The generalization of inner collisions to a full compression is harder with variable shift amounts. The design criteria [Dobbertin H. et al., 1996] are:

1. The shifts are chosen between 5 and 15.
2. Every message block should be rotated over different amounts.

Since the message order in each round for the four branches are considered differently by considering cyclic shift of i , the order in which shift constant is used in each branch and in each round is also varying.

Table 3.4: Amount of Shifts in each Round for Different Message Blocks

Round	M ₀	M ₁	M ₂	M ₃	M ₄	M ₅	M ₆	M ₇	M ₈	M ₉	M ₁₀	M ₁₁	M ₁₂	M ₁₃	M ₁₄	M ₁₅
1	15	14	12	13	9	8	7	6	11	12	14	15	6	5	8	9
2	14	15	13	11	8	7	9	5	12	14	13	11	5	6	7	8
3	13	12	14	15	7	5	6	9	13	15	12	14	9	8	5	7
4	12	13	11	14	6	6	5	8	14	13	11	12	8	7	6	5
5	11	11	15	12	5	9	8	7	15	11	15	13	7	9	9	6

3.3.2.6 Boolean Functions

The ten different boolean functions used are given in Table 3.5. In each single step operation there are two boolean functions f and g . In each round there are different f and g boolean functions as shown in Table 3.6, which help to resist attacks. The SAC (Strict Avalanche Criterion) [Yuan li and Cusick T.W., 2005] property of boolean functions also helps to defy attacks.

Table 3.5: Boolean Functions

$f_1(x, y, z) = g_4(x, y, z) = x \wedge y \wedge z$
$f_2(x, y, z) = g_3(x, y, z) = (x \wedge y) \vee (\neg x \wedge z)$
$f_3(x, y, z) = g_7(x, y, z) = (x \wedge \neg y) \wedge z$
$f_4(x, y, z) = g_1(x, y, z) = (x \wedge z) \vee (y \wedge \neg z)$
$f_5(x, y, z) = g_9(x, y, z) = x \wedge (y \vee \neg z)$
$f_6(x, y, z) = g_2(x, y, z) = (x \vee y) \wedge (\neg x \vee z)$
$f_7(x, y, z) = g_6(x, y, z) = (x \wedge \neg y) \wedge z$
$f_8(x, y, z) = g_5(x, y, z) = (x \vee z) \wedge (y \vee \neg z)$
$f_9(x, y, z) = g_{10}(x, y, z) = x \wedge (y \wedge \neg z)$
$f_{10}(x, y, z) = g_8(x, y, z) = x \wedge (\neg y \wedge z)$

Table 3.6: Boolean Functions used in each Round.

Branch	Round1	Round2	Round3	Round4	Round5
B1	f1,g1	f2,g2	f3,g8	f4,g4	f5,g5
B2	f10,g10	f9,g9	f8,g3	f7,g7	f6,g6
B3	f6,g6	f7,g7	f8,g8	f9,g9	f10,g10
B4	f5,g5	f4,g4	f3,g3	f2,g2	f1,g1

3.3.2.7 Constants

Here twenty different constants are used as shown in Table 3.7. These constants represent the first 32 bits of the fractional parts of the cube roots of the first twenty prime numbers. Different constants introduce asymmetry to each round. By using constants, resistance to the attacker who tries to find a good differential characteristic [Niels Ferguson, 1998] can be achieved with a higher probability.

Table 3.7: Constants used in each Round

Branch	Round1	Round2	Round3	Round4	Round5
B1	428a2f98x	71374491x	B5c0fbcfx	E9b5dba5 _x	3956c25bx
B2	59f111f1x	923f82a4x	Ab1c5ed5x	D807aa98x	12835b01x
B3	243185bex	550c7dc3x	72be5d74x	80deb1fex	9bdc06a7x
B4	c19bf174x	e49b69c1x	efbe4786x	0fc19dc6x	240ca1ccx

With these, the description of the design features of JERIM-320 is completed.

3.4 A Bird's Eye View on Hash Functions

A brief overview of the above discussions is summarized in Table 3.8.

Table 3.8: Hash Functions at a Glance

Algorithm	Block size (bits)	Word size (bits)	Output size (bits)	Rounds	Serial / parallel iteration	Max. message size (bits)	Operations	Collision
SHA-1	512	32	160	80	Serial	$2^{64} - 1$	+, and, or, xor, rotl, not	Yes (year 2005)
SHA-256	512	32	256	64	Serial	$2^{64} - 1$	+, and, or, xor, shr, rotr, not	None yet
RIPEMD-160	512	32	160	80	Parallel (2 lines)	$2^{64} - 1$	+, and, or, xor, rotl, not	None yet
RIPEMD-320	512	32	320	80	Parallel (2 lines)	$2^{64} - 1$	+, and, or, xor, rotl, not	None yet
FORK-256	512	32	256	8	Parallel (4 lines)	$2^{64} - 1$	+, and, or, xor, shr, shl, rotl, rotr	Yes (year 2007)
JERIM-320	512	32	320	80	Parallel (4 lines)	$2^{64} - 1$	+, and, or, xor, rotl, not	None yet

3.5 Detailed Comparison of JERIM-320 with FORK-256

Among the algorithms considered for study in section 3.2, FORK-256 can be considered to have the most similar design with JERIM-320 since both have four parallel lines of message processing. Hence a detailed comparison of JERIM-320 with FORK-256 has been done; this is shown in Table 3.9.

Table 3.9: Comparison of JERIM-320 with FORK-256

1. Differences				
Sl. No.	Properties	JERIM-320	FORK-256	Advantages of JERIM-320
1.	Message digest Length	320 bits	256 bits	Makes brute force attack more difficult.
2.	Given a message digest, number of operations required to find the original message.	2^{320}	2^{256}	Finding preimage or second preimage requires more operations.
3.	Speed	23.37 Mbps	48.05 Mbps	The speed of JERIM-320 is also acceptable considering the higher degree of security.
4.	Message block processing	20 times	4 times	Helps to resist attacks
5.	Ordering of message words	Message ordering in each round for the four branches are considered differently by considering cyclic shift of i	Ordering rule of message words in different branches is same always.	More resistance to Wang et. al attacks
6.	Shift values	Variable shift values in different rounds	Constant shift values	Provide better immunity against differential collision attack.

7.	Boolean functions	Ten different Boolean functions are used.	Two Boolean functions	The SAC property of Boolean functions help to resist against attacks.
8.	Constants	Twenty different constants	Sixteen different constants	Increases asymmetry and hence more secure.
9.	Message Block	JERIM-320 can use either the same or two different sub blocks in a single step operation	FORK-256 uses two different message sub blocks in single step operation	Provides flexibility on security
10.	Number of rounds and single step iterations in each branch.	Five rounds of 16 single step iterations making a total of 80 iterations	Single round of 8 single step iterations only.	Increases the complexity and makes attack more difficult
2. Similarities				
Sl. No.	Properties	JERIM-320	FORK-256	-
1.	Parallel branch	It consists of four parallel branches	Same as JERIM-320	-
2.	Input block and padding	Padding is used to make the original message length equal to a value 64 bits less than the exact multiple of 512 bits.	Same as JERIM-320	-

The comparison given in Table 3.9 clearly reveals the supremacy of JERIM-320 over FORK-256 due to the following properties

1. 320-bit hash length
2. eighty number of iterations on the message
3. twenty times processing of each message block
4. different message ordering in each round for all the four branches
5. introduction of cyclic shift
6. different order rule of message words for different branches
7. variable shift values, more number of Boolean functions and more number of constants.

These enhancements make JERIM-320 capable of resisting attacks to a much higher degree. It can be concluded that JERIM-320 is more secure than FORK-256.

3.6 Security Analysis

3.6.1 JERIM-320

1. The main difficulty in cryptanalyzing JERIM-320 comes from the fact that the same message blocks are given as input to each of the four streams in a permuted fashion. The attacker who tries to break JERIM-320 should aim simultaneously at four ways where the message difference passes, which would make the attacks more difficult.
2. By using one message sub block twice at each single step, it has been made difficult to construct a differential characteristic with high

probability.

3. To avoid an attack that depends on brute-force methods [Richard Clayton, 2001] the output from the hash function has been made sufficiently long.
4. While combining the outputs from the four branches, orthogonal operations (+ and ^) are used to create confusion and diffusion which adds to the security.
5. There is a strong avalanche effect [Subariah Ibrahim, et al., 2005] hence a change in a single message bit affects all the registers after five rounds.
6. All shortcut attacks on MD / Snefru [Bart Preneel, et al., 1998] target one of the intermediate blocks. Increasing the intermediate value to 320 bits helps to prevent these attacks.
7. The single step operation ensures that changing a small number of bits in the message affects many bits during the various passes. Together with the strong avalanche, this helps JERIM-320 to resist attacks like Dobbertin's differential attack [Dobbertin H., 1996] on MD4.

3.6.2 Comparison with FORK-256

1. An independent analysis resulting in a 1-bit near collision attack against a reduced version of FORK-256 has been published [Matusiewicz K. et al., 2007b]. Then they have shown how to use this result to attack the complete FORK-256 hash function. There are eighty rounds of single step operations in JERIM-320 to make all such chances difficult.

2. In FORK-256, the use of four streams with message reordering as a means to protect against differential analysis [Matusiewicz K. et al., 2007b] is ineffective since the same difference is applied to every message block and the same differential pattern is occurring simultaneously in the four streams. This is taken care in JERIM-320 by using a cyclic shift of i in the message ordering in each round so that different message orderings are used in different rounds. This causes the JERIM-320 algorithm to be much more resistant than FORK-256.
3. In FORK-256 the same compression functions f and g are used in all the four branches. Also some weakness in FORK-256 compression function has been published on two branches of the algorithm [Matusiewicz K. et al., 2007a]. This is overcome in JERIM-320 by using different compression functions in different branches and also in different rounds. Here if an attacker constructs an intended differential characteristic for one branch function, the different compression functions will cause unintended differential pattern in the other branch functions, thus finding specific differences for patterns would be difficult.
4. In FORK-256, the differences in the words of the internal state register do not diffuse identically. Thus, only the differences in the words A and E will spread to the other registers in the next round. As a result, a near collision occurs in FORK-256 [Matusiewicz K. et al., 2007b]. This factor is taken care in JERIM-320 by using the non linear functions f and g . Moreover these non linear functions are different in each branch and in each round. Also the shift values in each branch, for all the iterations are different which helps to change the internal values.

3.7 Performance Evaluation

3.7.1 Practical Implementations

In this section the performance evaluation of the hash functions is done using practical implementations and by using single step computations. The total number of operations, memory requirements and the speed performance of JERIM-320 using one message block in single step operation and using two message blocks in single step operation were compared with FORK-256, SHA-1, SHA-256, RIPEMD-160 and RIPEMD-320. A detailed comparison with FORK-256 is given in section 3.7.1.1 due to the similarity in main structure. Also a separate comparison with RIPEMD-320 is given in section 3.7.1.2, since it matches with the hash length of JERIM-320. Comparisons with the other hash functions SHA-1, SHA-256, RIPEMD-160 are given in section 3.7.1.3.

3.7.1.1 Comparison with FORK-256

As shown in Table 3.10 the total number of operations used in a single step operation of FORK-256 is 1.3 times than that in JERIM-320.

Table 3.10: Comparison between the Number of Operations of JERIM-320 and FORK-256

Operation	JERIM-320 (using one message block in single step operation)	JERIM-320 (using two message blocks in single step operation)	FORK-256
Addition	42	42	97
Bitwise operation ($\wedge, \vee, \Delta, \neg$)	187	187	112
Shift operation	33	33	137
Total number of operations	262	262	346

As shown in Table 3.11, the memory requirement of JERIM-320 is less than that of FORK-256. JERIM-320 uses 80 iterations for each message block, where as FORK-256 uses only 8 iterations. In each branch there are ten chaining variables in JERIM-320, but FORK-256 has only 8 variables. Moreover each message block is processed 20 times in JERIM-320 where as in FORK-256 it is only 4 times. Due to these, obviously the speed of operation will be slightly less for JERIM-320 than FORK-256 as shown in Table 3.11. The speed of JERIM-320 using one message block in single step operation is nearly 3.4 times less than that of FORK-256 and JERIM-320 using two message blocks in single step operation is 2.05 times less than that of FORK-256. But the multiple iterations and processing on the message blocks in JERIM-320 will result in much higher security. The speed of JERIM-320 is still very much acceptable.

Table 3.11: Performance Comparison between JERIM-320 and FORK-256

Algorithm	Speed (Mbps)	Memory requirement (bytes)
JERIM-320 using one message block in single step operation	14.01	12003
JERIM-320 using two different message blocks in single step operation	23.37	12039
FORK-256	48.05	12149

3.7.1.2 Comparison with RIPEMD-320

A comparison of JERIM-320 and RIPEMD-320 with respect to the total number of operations is shown in Table 3.12, while that with respect to memory requirements and the speed of performance is shown in Table 3.13.

RIPEMD-320 provides the ability for longer hash strings and is a double width string variant of the popular RIPEMD-160. But both of these have only two lines of message processing and each message block is processed ten times only. Hence RIPEMD-320 and RIPEMD-160 are almost equally susceptible to attacks in the long run. As shown in Table 3.12, the total number of operations used in JERIM-320 is 4.03 times more than that in RIPEMD-320. Due to this, although the speed of JERIM-320 is slightly lower, the multiple operations on the message blocks will result in higher security.

Table 3.12: Comparison between the Number of Operations of JERIM-320 and RIPEMD-320.

Operation	JERIM-320 (using one message block in single step operation)	JERIM-320 (using two message blocks in single step operation)	RIPEMD-320
Addition	42	42	20
Bitwise operation ($\wedge, \vee, \wedge, -$)	187	187	36
Shift operation	33	33	9
Total number of operations	262	262	65

As shown in Table 3.13, the memory requirement of JERIM-320 is 1.3 times more than that of RIPEMD-320. JERIM-320 makes use of four parallel lines of message processing and hence the variables and computations required in JERIM-320 are more. The speed of JERIM-320 using one message block in single step operation is nearly 2.5 times less than that of RIPEMD-320 and that of JERIM-320 using two message blocks in single step operation is 1.5 times less than that of RIPEMD-320. These are because of the increased number of Boolean functions, constants and the more number of lines of message processing used in JERIM-320. The number of Boolean operations in RIPEMD-320 is five while that in JERIM-320 is ten. Similarly the number of constants in RIPEMD-320 is ten while for JERIM-320 it is twenty. Also each block in RIPEMD-320 is processed ten times while in JERIM-320, it is

twenty times. With all these, JERIM-320 provides higher security to make its overall performance good enough for acceptance by the internet community.

Table 3.13: Performance Comparison between JERIM-320 and RIPEMD-320

Algorithm	Speed (Mbps)	Memory requirement (bytes)
JERIM-320 using one message block in single step operation	14.01	12003
JERIM-320 using two different message blocks in single step operation	23.37	12039
RIPEMD-320	35.63	8927

3.7.1.3 Comparison with SHA-1, SHA-256, RIPEMD-160

As shown in Table 3.14 the total number of operations used in JERIM-320 is 7 times that of SHA-1, 3.7 times that of SHA-256 and 4 times that of RIPEMD-160. This is because of the hash function JERIM-320 making use of four parallel lines of message processing and hence the variables and computations in JERIM-320 will be more compared to other hash functions mentioned here.

Table 3.14: Comparison between the Number of Operations of SHA-1, SHA-256, RIPEMD-160 and JERIM-320

Operation	SHA-1	SHA-256	RIPEMD-160	JERIM-320
Addition	12	20	20	42
Bitwise operation ($\wedge, \vee, \wedge, \neg$)	18	27	36	187
Shift operation	7	23	9	33
Total number of operations	37	70	65	262

As shown in Table 3.15, the memory requirement for JERIM-320 is more and the speed is less than that of SHA-1, SHA-256 and RIPEMD-160. This is because of the increased number of Boolean functions, the need for other operations like add, shift as well as the greater number of lines of message processing used in JERIM-320.

Table 3.15: Performance Comparison between SHA-1, SHA-256, RIPEMD-160 and JERIM-320

Algorithm	Speed (Mbps)	Memory requirements (bytes)
SHA-1	60.89	6533
SHA-256	55.93	7214
RIPEMD-160	35.89	8679
JERIM-320	14.01	12003

3.7.2 Single Step Computation

The single step computation for comparison of speed of the six hash functions is as follows:

The step operation of SHA-1 consists of 4 additions, 2 shift and a Boolean function. The Boolean function consists of 3 unit operations, and the step operations consist of 80 steps (4 rounds * 20 iterations). That is $1 \text{ (stream)} * 80 \text{ (steps)} * 9 \text{ (step operations)} = 720 \text{ (unit operations)}$

The step operation of SHA-256 consists of 7 additions, 2 summations and 2 Boolean functions. Each Boolean function and summation consists of 3 unit operations, and the step operation consists of 64 steps. That is $1 \text{ (stream)} * 64 \text{ (steps)} * 19 \text{ (step operations)} = 1216 \text{ (unit operations)}$

RIPEMD-160 consists of 4 additions, 2 circular shifts and a Boolean function. The Boolean function consists of 3 unit operations.

$2 \text{ (streams)} * 80 \text{ (steps)} * 9 \text{ (step operations)} = 1440 \text{ (unit operations)}$.

RIPEMD-320 consists of 4 additions, 2 circular shifts and a Boolean function. The Boolean function consists of 3 unit operations.

$2 \text{ (streams)} * 80 \text{ (steps)} * 9 \text{ (step operations)} = 1440 \text{ (unit operations)}$.

FORK-256 consists of 10 additions, 6 XORs, 8 circular shifts and 2 Boolean function. The Boolean function consists of 1 unit operations.

$4 \text{ (streams)} * 8 \text{ (steps)} * 26 \text{ (step operations)} = 832 \text{ (unit operations)}$.

The step operation of JERIM-320 consists of 5 additions, 4 XORs, 4 shift and 2 Boolean functions. Each Boolean function consists of 3 unit operations, and the step operation consists of 80 steps (5 rounds * 16 iterations). That is $4 \text{ (streams)} * 80 \text{ (steps)} * 19 \text{ (step operations)} = 6080 \text{ (unit operations)}$.

From the above computations it can be seen that the number of unit operations in JERIM-320 is 8.4 times than in SHA-1, 5 times than in SHA-256, 4.2 times than in RIPEMD-160 and RIPEMD-320 and 7.3 times than in FORK-256. Due to this, the hash code produced in JERIM-320 will be much more secure than the other hash functions.

3.8 Statistical Analysis for the Dual Functioning of JERIM-320

3.8.1 Introduction

Cryptographic hash functions can also contribute to be a good foundation for a PRNG [John Viega, 2003]. Several designs have been using MD5 or SHA-1 in this capacity. HMAC-SHA-1 is generally considered to be indistinguishable from a PRF (Pseudo Random Function), a function selected at random from arbitrary strings to 160-bit outputs.

A PRNG can be implemented using the hash function JERIM-320 in the following way:

The ten chaining variables in the JERIM-320 hash function are initialized with values which are considered as the seed value for the generator. The message whose integrity is to be validated is given as input to JERIM-320. The message is processed as 512-bit blocks and for each 512-bit block an intermediate hash code is generated. This intermediate hash code is considered as the Pseudo random number. Again the values of chaining variables are replaced with the intermediate hash code values. Then the next 512-bit message block is considered and so on. The number of random numbers it can produce depends upon the input message length. A good hash

function will be able to produce random values with sufficient speed and quality. Since good hash functions can be substituted as PRNGs, we focus on analyzing the randomness property of the hash function JERIM-320.

3.8.2 Randomness Tests

The statistical analysis of JERIM-320 is done mainly using the tests listed in section 2.1.3. These tests are commonly intended for determining whether the binary sequences possess some specific characteristics that a truly random sequence is likely to exhibit.

3.8.3 Results

Here Frequency, Serial, Poker and Runs tests are analysed using the Chi-square table and Autocorrelation test is analysed using Normal table, as specified for each randomness tests. Tables 3.16, 3.17 and 3.18 show the results of the specified tests.

Table 3.16: Statistical Analysis using Frequency and Serial Tests

Input File Size (bytes)	No. of intermediate hash codes as PRN	No. of bits in each intermediate hash code	Total no. of bits considered	Frequency Test	Serial Test
69	20	32	640	0.2250	2.3649
407	70	32	2240	0.0875	2.0880
4207	660	32	21120	0.0547	2.0706
7514	1180	32	37760	0.7653	5.1989
10669	1670	32	53440	0.2102	2.9519

Table 3.17: Statistical Analysis using Poker and Runs Tests

Input File Size (bytes)	No. of intermediate hash codes as PRN	No. of bits in each intermediate hash code	Total no. of bits considered	Poker Test	Runs Test
69	20	32	640	12.60	7.82
407	70	32	2240	40.94	4.93
4207	660	32	21120	245.23	14.48
7514	1180	32	37760	529.67	15.93
10669	1670	32	53440	1043.65	14.14

Table 3.18: Statistical Analysis using Autocorrelation Test

Input File Size (bytes)	No. of intermediate hash codes as PRN	No. of bits in each intermediate hash code	Total no. of bits considered	Autocorrelation Test
69	20	32	640	1.4823
407	70	32	2240	2.7360
4207	660	32	21120	1.6176
7514	1180	32	37760	2.4346
10669	1670	32	53440	1.5878

3.8.4 Performance Evaluation

Details of the performance evaluation are shown in Table 3.19. The JERIM-320 can handle messages of length 9.7 Mbps for ensuring the message integrity and simultaneously producing random numbers. Since each 512-bit block produces an intermediate hash code of 320-bit, the number of random bits JERIM-320 produce are 6.06 Mbps. The JERIM-320 hash function successfully passes all the five empirical tests for every run. The memory

requirement for JERIM-320 is only 11462 bytes. Hence JERIM-320 can be simultaneously used for providing message integrity and for generating pseudo random numbers with reasonable speed.

Table: 3.19: Performance Evaluation of JERIM-320 as Hash Function and as PRNG

Algorithm	Message length for ensuring integrity (Mbps)	Number of Random bits produced (Mbps)	Memory requirement (bytes)
JERIM-320	9.7	6.06	11462

From the results of analysis and performance evaluation it can be seen that JERIM-320 is also a reliable random number generator. All the five empirical tests are passed by this generator for all the random streams produced. Hence JERIM-320 can be used not only to ensure message integrity but also as an efficient pseudo random number generator. The dual services of JERIM-320 make it very useful in cryptographic applications.

3.9 Conclusions

Detailed studies on different popular Hash Functions have been done, desired properties were identified and a new hash function called JERIM-320 with improved security and reasonable speed has been designed. Various cryptographic hashes are analyzed with JERIM-320 using practical implementations and using single step computations. The core strength of JERIM-320 is the four parallel lines with five rounds which provide a strong nonlinear avalanche plus more number of register operations that increase

diffusion and make differential attacks harder. Other salient features of JERIM-320 which help to provide improved security are the larger number of Boolean functions and constants and repeated processing of each message block with increased number of operations in each single step. These enhancements make JERIM-320 capable of resisting attacks to a much higher degree compared to the other ones. Since message integrity is an important security service in today's high-speed network protocols and also since the confidence level with respect to the current candidates is coming down, new hash schemes have become a necessity. A more secure hash code JERIM-320 can definitely be a substitution.

3.10 Test Vectors

The following are few test vectors using one message block and two message blocks in single step operation of JERIM-320 hash function.

3.10.1 JERIM-320 using One Message Block in Single Step Operation

Message 1

abcdefghijklmnopqrstuvwxyABCDEFGHIJKLMNOPQRSTUVWXYZ0123
456789abcdefghij

Intermediate Values:

Branch1

a2e47dc1 f7de8b66 92610e5a 642d9193 d1b17ef4 8025af21 6be9dfb
49de3ccc 1502d6ae cf596c85

Branch2

2e47d19f 11f7b254 8db858c6 b906f89a 7d24c5df 49d2dbab 9ea4903e
8e6805c0 206273be 9dc7eb34

Branch3

7f4d8321 894e27b7 fddb05d aa293970 74c83bda d387cb05 ce397853
1f7e3eff 45323469 ab4c0c62

Branch4

e4d22ef1 ba06df58 e0c12344 4852c253 e82bc3ba 3e1f9d87 d8cd742a
f9d89672 dbbc3c25 6a654e78

Hash Code

1518cd49 1d4d177f f33c1d89 5786e925 fa38cf04 f6c6413e e5a1a61
78f83037 7072e78e be33818b

Message 2

jbcd efghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ01234
56789abcdefghijklmnop

Intermediate Values:

Branch1

2bb17fd8 88b2907b fb5597b9 e08e4cf7 d869113e ce86fffb 10f657de
39764f9c 1b563eb9 fa612803

Branch2

3ddd6242 27636afd 6b9313ed 1873d830 610d9b05 aaae8d4 93603614
64060d12 3e3b507c ea1999ee

Branch3

41c3a8e5 e32e6792 f43821e2 bff7dd18 6285f80 7f31331a 5a190dc 769f1f85
919a3d1f 8c9cc946

Branch4

9da92331 d9f50dad 4fd24ffc b7610f52 f4e9f3a0 f07089d5 9a4bf89a
84827901 a2ad3d87 2a8b786e

Hash Code

44a71b45 d3ebb45e 62a2aa2f f9c2c6df ee1d95d6 5905ef0c de51f3ff
3a72b49c 4cb3a37 e4fec1f8

Message 3

ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123
456789ABCDEFGHIJ

Intermediate Values:

Branch1

c77dfed4 614b7c41 3bd726a1 a0966262 fc6c97f9 41feb695 682433d4
b3d0dc73 a71d9ebc f7bc6ab0

Branch2

1f4552da 71223052 247af80 9af62dea 8505b776 a2da0ced abaeec81
784e8cb8 295cda13 2e80426f

Branch3

77691f2c dc67d24a 1b6c4a68 fad8fde2 77beb0a7 94c3fc9f 6ce06ba8
3e3420a7 a88b6497 be403656

Branch4

ccd691dc 29b1044d 49afd85e c612fa5b 30f546e6 c2c75448 508f0a53
43d85c70 5c26e1c1 d990b677

Hash Code

cb1d1549 295fle1b 515b5fa6 9f0c6d46 a71cb8d 163de581 2ea157cc
fb113d66 61ca0189 a49b148c

Message 4

JBCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz01234
56789ABCDEFGHIJ

Intermediate Values:

Branch1

44f55b66 9842f21 c13bc735 dc0acf25 6fdac5e6 8476000 c77ee292 eca37a0e
25b432fe 7a0efa40

Branch2

79686a9b 38974aac 4ca8937a cfe76c90 bb28f497 2c01d4 d338f286 1a270d49
d955d46c b2fae10f

Branch3

f83d127d e0c70ca6 cccb1814 d4f0a1a ea6cb6cb 6e52fccf ca13e26a f55f29dc
908f0b13 d98b9335

Branch4

2871fd6f d3e258 1da9c273 cff5f7c0 50586673 13d7a0a4 b2255cf7 3815484
a8e777f3 ad5f92b6

Hash Code:

595b4d64 44cb2fa7 5ee3d44c b2cbbfae 807aede2 6547cd46 b2f07391
364eb41 c34bb7fe 14bca5b9

3.10.2 JERIM-320 using Two Different Message Blocks in Single Step Operation

Message1

abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123
456789abcdefghij

Intermediate Values:

Branch1

4383170a ab2f4c9f 720700b2 58b7a70c f37cea35 fe9b3326 8b7871a7
99873e8 b6bf475a d50e448d

Branch2

76eef25 f6bf0918 fb240e3e 570221fd 2e19c9ef baaace8e 850ace28 25f46e36
2b54edcd 6bd8da7a

Branch3

2d9678c2 88c76853 f468278e af7fe30b 6a59130f d722a202 a6bbe9aa
9c08835e 779d9324 a0b53e0

Branch4

4d96f05a e837e968 93693d79 3a6773e 82b71d7b 7aab3f62 69c0c1f2
7906d4f5 997afe39 49834e4b

Hash Code:

94311971 3df02765 304f54e4 952ca42 f0f2a85 3a3ad78c 86ee3f9a 9c2e75f3
8c1d542b 1af336d6

Message 2

jbcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ01234
56789abcdefghij

Intermediate Values:

Branch1

66baf333 e3f3b3cd 33525789 7d588262 e733f1c4 2cb06831 e2b7c51f
e65c07e9 f252424e 2b5e0ece

Branch2

7fe61e0e 9b3911db 5982910 d64958eb e395685f 8e4cd3c9 1e434afe
fb49c850 343f7682 e8ebd290

Branch3

18bf4291 69a1ca34 b27811aa 72b5def8 f40b6d7f 37c2fea8 962bba12
237d9f10 50c66664 a7219835

Branch4

7f02f6c4 58652c44 a43a159b 77a2e22b a7488c47 6266f5a7 2deeea13
206c2be5 39784dea 9a9d1860

Hash Code:

7f43b0b0 7ba8b8ce b32d1099 4d6b329 e3dd6362 2434e2d0 e9d7a660
afb927a cab690b9 b2d5619f

Message 3

ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123
456789ABCDEFGHIJ

Intermediate Values:

Branch1

12505570 af4b1952 fbf3443b 30c9f41c 4515ae48 71cc5927 21527ec3
90175a19 b3668c63 d8d55f91

Branch2

213b2bea e6b6cdbc 74bd72c3 13fa35bb cd9a8edb 1aa0586b e99e7b73
fd1b6d96 320ce9fe 94c6d998

Branch3

886b235f cb7e593e f8150760 a41b6a73 332cb981 4570312a ae01be73
b96395d0 bef6bb6 1700f993

Branch4

5f63be3 a5710e8 4af02fdf 70104edf 9001025e 208adbaa a9b85847 b521
8c56de58 a73aeefd

Hash Code

1108ea66 3690671d 8c159bb4 215a864c 41a2652e be0bf0ad 475bee5a
60c21c72 51f315a6 e28451f2

Message 4

JBCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz01234
56789ABCDEFGHIJ

Intermediate Values:

Branch1

5240a705 747c9471 df6da2e4 1fc9add9 893f0f5f 8c725963 e1eb72f
dd2bc9d9 abe7fa8 f785ddc9

Branch2

341f1f1 af67dedd b955cfe6 13f45808 5a96cd1b d9cdd0b7 c53d61fe
b440b60c 9f2e7824 57d46a8

Branch3

7e65157b d661cd2a d3fee90a f24587af 19a9de96 3c8d183b 53c83a48
f30db4f9 bea3b716 f86592f6

Branch4

e995bf68 bad3e012 cc2afdeb dfbd4802 52769c41 f6f6d7ff 5d23b831
9a9e3aa4 8b4c4f16 101badb4

Hash Code:

2dd6daa 79c353ce d78f8e05 374ae7f0 2c3e0b64 c00098fb dd4dcc2e
e38ae930 316dd34f 664e84ea

Chapter 4

Design and Development of a New Message Authentication Code: MACJER-320

Abstract:

The security services of providing confidentiality and integrity have been dealt with in the previous chapters. The other two main security services demanded by the network community could be message authentication and non-repudiation. Message authentication ensures two or more parties to verify that the received digital content is sent from an authorized person, which is important for electronic communication such as e-mail, e-commerce etc. This chapter introduces a new message authentication code MACJER-320. It is developed using JERIM-320 and in combination with a 320-bit secret key. MACJER-320 is using JERIM-320, a 320-bit hash function, while the popular HMAC uses a 160-bit hash function, SHA-1. The performance evaluation of the two methods has been done by using practical implementation.

4.1 MACJER-320

4.1.1 Introduction

Verifying the integrity, authenticity and non repudiation of information is a major necessity in computer systems and networks. Message authentication code (MAC) is one of the fundamental cryptographic primitives used extensively in the construction of security services in networks for general digital data transfer offering authentication of sender, data integrity and to some extent the non-repudiation. In particular, two persons communicating over an insecure channel require a method by which information sent by one person can be received and validated as authentic by the other. When a person 'A' transmits a message to another person 'B', it appends to the message a value called the authentication tag, which is computed by the MAC algorithm as a function of the transmitted message and the shared secret key. At reception, 'B' recalculates the authentication tag on the received message using the same mechanism and key, and checks whether the value obtained is equal to the tag attached to the received message. If both are the same, then 'B' can confirm that the message has started off from the intended sender and that it has not been tampered with during the transmission. Here the sender and receiver share a secret key k . The sender uses k to generate a tag and sends it along with the message to the receiver. Only the party who shares the secret key can generate the same tag. This ensures the sender's authenticity and non-repudiation. The algorithm producing the MAC is designed to reflect any changes in the message. This ensures the data integrity. Such a mechanism is most commonly based on the

secret key shared between the parties, which take the form of a message authentication code. This is shown in Fig. 4.1.

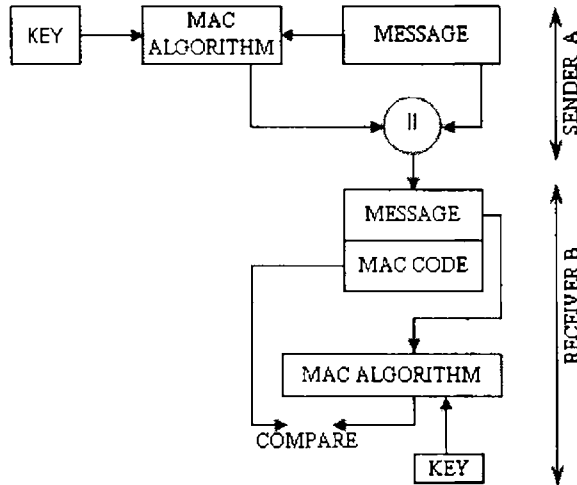


Fig. 4.1: Message Authentication Code (MAC)

MACs fall into two categories based on their fundamental building blocks [William Stallings, 2003]. One approach is to use symmetric block cipher in a cipher block chaining mode. Here MACs are constructed out of block ciphers like DES [NIST FIPSPUB 46-3, 1999] as seen in the DES-CBC MAC [Mihir Bellare et al., 1999], widely used in US and in International standards. The basic idea is to encrypt the message blocks using DES-CBC and output the final block in the cipher text as the checksum. Another popular approach is to use cryptographic hash functions like SHA-1 and MD5. This is particularly visible in the internet community, where the development of security protocols has lead to the need for simple, efficient and widely available MAC mechanisms. MACs with hash functions are more popular because they are faster than block ciphers in software implementation.

MAC algorithms are widely used in Internet security protocols (SSL/TLS, SSH, IPsec.) for encryption and authentication to support secure browsing, file transfer and remote login between the end users and servers (Bo Yang et al., 2006), in mobile communications (GSM and 3GPP) and in the financial sector for debit and credit transactions [Helena Handschuh, 2004]. Routing protocols have begun to use message authentication systems to verify the routing information transferred among routers. The security of the MAC algorithm depends on the difficulty for an unauthorized entity to produce a forgery that is, a new message with a valid MAC.

In short, a MAC can be thought of as a keyed hash, with the following properties:

1. Given any message, it is difficult to create a MAC without knowing the key.
2. Given a message and the corresponding MAC, it is difficult to create a new message with the same MAC.
3. Given any MAC, it is difficult to find a message that corresponds to it or matches it.

4.1.2. Motivation and Design Factors

The popular MAC mechanism used nowadays is the HMAC [NIST FIPSPUB 198, 2002] with MD5 or SHA-1 as the hash function. But the strength of MD5 [Rivest R.L., 1992] and SHA-1 [NIST FIPSPUB 180-2, 2002] has been called into question as a result of recent findings [Jongsung Kim et al., 2006]. Therefore development of new message authentication codes that involve the use of cryptographic hash functions with sound security

assumptions on the basic hash function are important in the current scenario. The outcome of an attempt in this line is a new message authentication code MACJER-320.

The following are the design objectives of MACJER-320:

1. To be able to use widely available hash functions without modifications.
2. To allow easy replacement of the embedded hash function in case faster or more secure hash functions are found or required.
3. To preserve the original performance of the hash function without incurring a significant drop.
4. To use and handle keys in a simple way.
5. To have a well understood cryptographic analysis of the strength of the authentication mechanism based on reasonable assumptions about the embedded hash function.

4.1.3 Description of MACJER-320

The variables used in the MACJER-320 construction are given in Table 4.1.

Table 4.1: Variables used in MACJER-320

K - Shared symmetric key.
M - Input message.
B - Number of bits in each block.
MDA - Message digest algorithm or Hash Function (JERIM 320)

H - Hash code
SH - Circular shifted hash code H
SK - Circular shifted key K

The step-by-step approach of MACJER-320 is given in Algorithm 4.1

Algorithm 4.1: MACJER-320

Step 1: Make length of K equal to B.

Here the initial key K is 320-bit long and the block length B is 512-bit. To make the length of K equal to the block length add as many 0 bits as required to the left of K. Hence add 192, 0 bits to the left of key K.

Step 2: Prefix and suffix the key along with the message.

Divide the key into two equal parts (256 bits each) and then prefix the message using 256 lsb bits of the key and suffix the message using 256 msb bits of the key.

Step 3: Apply the message digest algorithm or hash function.

Now, JERIM-320 is applied to the output of step 2 (i.e. to the combination of the 256 lsb bits of the key, the message and the 256 msb bits of the key) to produce the 320-bit hash code H.

Step 4: Circular shift hash code H and the initial key K.

Circular shift H by 13 bits and key K by 17 bits to the left to produce the shifted hash SH and the shifted key SK.

Step 5: XOR K with SH to produce KSH.

Now XOR K with SH to produce a variable called KSH.

Step 6: Add H with SK to produce HSK.

Now add H with SK to produce a variable called HSK

Step 7: XOR KSH with HSK to produce MAC.

XOR KSH with HSK to produce the final 320-bit message authentication code.

A pictorial representation of these seven steps are given in Fig. 4.2.

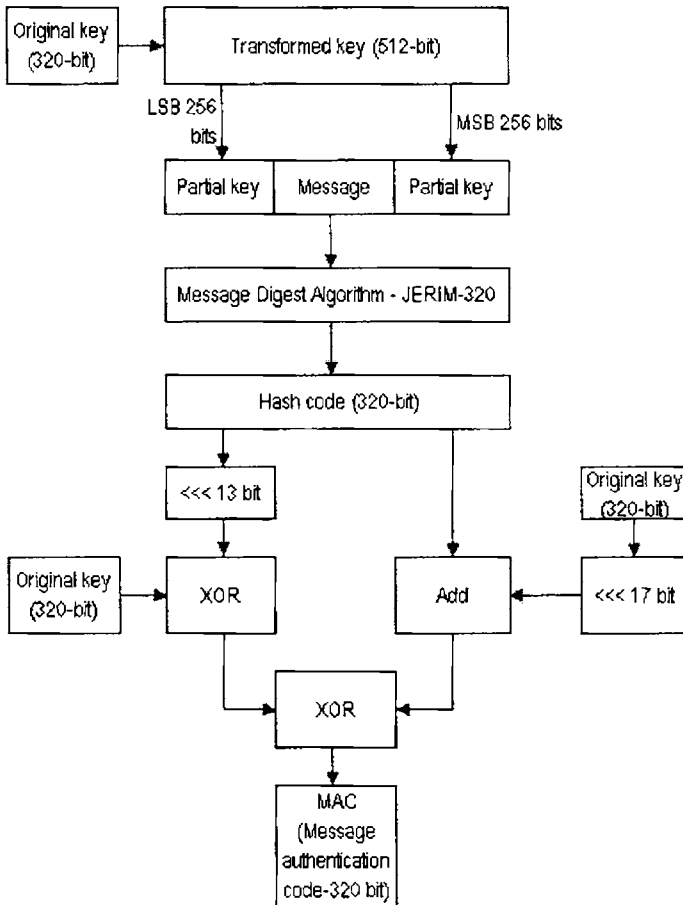


Fig 4.2: MACJER-320 Structure

4.1.4 Security Analysis

section 4.1.5 gives an analysis of the properties of MACJER-320, the new Message Authentication Code, to establish its significantly higher level of security than the popularly used ones.

4.1.5 Properties of MACJER-320

1. The security of the message authentication mechanism presented here depends mainly on the cryptographic properties of the hash function JERIM-320 as mentioned in section 3.6.1.
2. The length in bits of a message authentication code is directly related to the number of trials that an intruder has to perform before a message is accepted. For a MAC value of bit-length m , the intruder has to perform on average of 2^{m-1} random on-line MAC verifications before his strategy succeeds. Thus in MACJER-320, an intruder requires 2^{320-1} trials.
3. The message is enveloped with a secret prefix and a secret suffix before the hash code is computed. This hybrid method is stronger than either the prefix or the suffix variant [Gene Tsudik, 1992] and provides protection against message substitution attacks [Lifeng Lai et al., 2008] when used in conjunction with a strong hash function JERIM-320. Also the splitting of the key into two parts strengthens the key by increasing confusion at the cipher text level [Thomas Calabrese, 2006]
4. Another important property of this hybrid method is its resistance to birthday attacks [Wagner D., 2002]. Consideration of these attacks is

important since they strongly improve on exhaustive search attacks. Since these attacks require knowledge of the MAC value (for a given key) on about $2^{n/2}$ messages (where n is the length of the hash output) for values of $n \geq 320$ the attack becomes totally infeasible [Menezes A. et al., 1997].

5. When combining functions and operations together, the orthogonal operations like XOR and addition are used which cause confusion and diffusion in the MAC.
6. The shifting of the hash code and key was done to increase confusion thus strengthening the output.
7. XORing has the effect of randomizing the input almost completely and overcoming any regularity that may appear in the output.

4.1.6. Performance Evaluation

The total number of operations, memory requirements and the speed performance of JERIM-320 using one message block in single step operation and MACJER-320 were evaluated.

As shown in Table 4.2 and Table 4.3, the total number of operations used and the memory requirement in MACJER-320 are just 1.06 times than that in JERIM-320. This negligible increase in number of operations will not practically affect the speed of MACJER-320. As shown in Table 4.3, the speed of MACJER-320 is less only by 0.06 times that of JERIM-320, but authentication service could also be achieved along with message integrity while using MACJER-320. Moreover, the simple and inexpensive secret

prefix and secret suffix methods, the usage of orthogonal operators, the usage of shift operators and the usage of 320 bits secret key in MACJER-320 provide protection against differential attacks [Jiqiang Lu et al., 2006], when used in conjunction with the strong hash function JERIM-320.

Table 4.2: Comparison between the Number of Operations of JERIM-320 and MACJER-320

Operation	JERIM-320 (using one message block in single step operation)	MACJER-320
Addition	42	46
Bitwise operation ($\wedge, \vee, \Delta, \neg$)	187	193
Shift operation	33	41
Total number of operations	262	280

Table 4.3: Performance Comparison between JERIM-320 and MACJER-320

Algorithm	Speed (Mbps)	Memory requirements (bytes)
JERIM-320 using one message block in single step operation	14.01	12003
MACJER-320	13.15	12530

4.1.7 Test Vector

Key :

0x2345676,0x46565688,0x57239945,0x45111571,0x77783528,0x72885357,0
x17242468,0x53338223,0x42871903,0x97238156

Message:

abcdefghijklmnopqrstuvwxyzaBCDEFGHIJKLMNOPQRSTUVWXYZ

Message Authentication Code :

6c3e78aa 63c33ad1 48037b20 419e471c 67a3b429 c2c5c8b1 d909d7a8
3404118f 2beaddf4 8fbb08e7

Message :

jbcdefghijklmnopqrstuvwxyzaBCDEFGHIJKLMNOPQRSTUVWXYZ

Message Authentication Code :

980b9b98 d0aa2d30 580317bd 3ca59edb 2f00667 e938a0ef 1adb0270
20f9ee91 c4308808 c9e58889

4.2 Performance Evaluation between MACJER-320 and HMAC-SHA-1

A brief description of HMAC and the performance evaluation of MACJER-320 in comparison with the popular HMAC-SHA-1 have been done

in this section. The Secure Hash Algorithm (SHA-1) is referred to section 3.2.2.

4.2.1 HMAC

The different variables used in the HMAC algorithm are shown in Table 4.4.

Table 4.4: Basic Notations in HMAC

MD - Message digest / hash function
M - Input message
B - Number of bits in each block
K - Shared symmetric key
K1 - Transformed key K
Ipad - String 00110110 repeated b/8 times
Opad - String 01011010 repeated b/8 times
H - Hash code

The step-by-step approach of the HMAC message authentication code is given in Algorithm 4.2.

Algorithm 4.2: HMAC

Step 1: Make the length of K equal to B. Append enough zeros to the left end of K to create a B bit key K1.

Step 2: XOR K1 with Ipad to produce the B bit block S.

Step 3: Append M to S. That is the original message is simply appended to the end of S.

Step 4: Apply the Message digest algorithm / hash function to the output of Step 3 to produce hash code H.

Step 5: XOR K1 with Opad to produce the B bit block S1.

Step 6: Append the hash code H produced in Step 4 to S1.

Step 7: Apply the message digest algorithm / hash function to the output of Step 6 to produce the final MAC.

4.2.2 Security Analysis of MACs and Hash Functions

sections 4.1.4 and 3.6.1 describe an analysis of the properties of MACJER-320 and JERIM-320. The security analysis of HMAC-SHA-1 is explained in 4.2.2.1.

4.2.2.1 HMAC-SHA-1

1. In HMAC the XOR with Ipad results in flipping one-half of the bits of K. Similarly the XOR with Opad results in flipping the other-half of the bits of K, but a different set of bits. In effect, by passing S and S1 through the compression function of the hash algorithm, we have pseudo randomly generated two keys from K, which adds security to HMAC.

2. The recent attacks by Wang et al. [Xiaoyun Wang and Hongbo Yu, 2005] and Biham et al. [Biham E. et al., 2005] have undermined the confidence in the popular hash functions such as MD5 or SHA-1.
3. As outlined in the paper “keying hash functions for message authentication”, HMACs can be vulnerable to birthday, collision and other attacks. [Mihir Bellare et al., 1996b].
4. Other publications “On the security of HMAC and NMAC based on HAVAL, MD4, MD5, SHA-0 and SHA-1” [Jongsung Kim et al., 2006] and “Note on Distinguishing, Forgery and Second Preimage Attacks on HMAC-SHA-1 and a Method to Reduce the Key Entropy of NMAC” [Christian Rechberger and Vincent Rijimen, 2006] have shown how to use the differential distinguishers to devise a forger attack on HMAC.
5. The strongest attack known against HMAC is based on the frequency of collisions for the hash function. With this, HMACs have become more insecure [Mihir Bellare et al., 1996a].

In this scenario, the security provided by the HMAC is no more fully reliable, although the same is being widely used even now. This calls for a performance evaluation between MACJER-320 and HMAC-SHA-1 to explore the possible usage of MACJER-320 as an alternative.

4.2.3 Performance Evaluation

The performance evaluation of MACJER-320 and HMAC-SHA-1 is done using practical implementations. Evaluation of JERIM-320 and SHA-1 using single step computations has already been done in section 3.7.2. The

total number of operations, memory requirements and the speed performance of MACJER-320 using JERIM-320 hash function and HMAC using SHA-1 hash function are compared here.

The MACJER-320 produces a MAC of 320 bits where as HMAC-SHA-1 produces a MAC of 160 bits only. Hence MACJER-320 can definitely provide added security than HMAC-SHA-1.

As shown in Table 4.5 the total number of operations used in MACJER-320 is 3.7 times than that in HMAC-SHA-1. The hash function JERIM-320 in MACJER-320 makes use of four parallel lines of message processing and hence the variables and computations required in JERIM-320 will be more compared to the single stream hash function SHA-1 in HMAC. The multiple operations on the message blocks in MACJER-320 will result in much higher security with a negligible compromise in the speed of operation.

Table 4.5: Comparison between the Number of Operations of MACJER-320 and HMAC

Operation	MACJER-320 using JERIM-320	HMAC using SHA-1
Addition	46	24
Bitwise operation ($\wedge, \vee, \wedge, \neg$)	193	39
Shift operation	41	13
Total number of operations	280	76

As shown in Table 4.6, the memory requirement for MACJER-320 is more than that of HMAC-SHA-1 and the speed of MACJER-320 is less than that of HMAC-SHA-1. These are because of the increased number of Boolean functions, the need for other operations like add, shift as well as the greater number of lines of message processing used in JERIM-320 than in SHA-1.

Table 4.6 Performance Comparisons between MACJER-320 and HMAC

Algorithm	Speed (Mbps)	Memory requirements (bytes)
MACJER-320 using JERIM-320	13.15	12530
HMAC using SHA-1	57.58	8074

4.3 Conclusions

A new message authentication code called MACJER-320 has been designed with better security and reasonable speed. In MACJER-320, the simple and inexpensive secret prefix and secret suffix methods, orthogonal operators, shift operators and 320-bit secret key provide protection against differential attacks, when used in conjunction with a strong hash function JERIM-320. Moreover in MACJER-320 the hash function can be used as a black box, so that the replacement of the underlying hash function is easily supported. Also with a minute increase in time and memory requirement, additional security services like message authentication and non-repudiation could also be achieved along with message integrity.

The performance evaluation of MACJER-320 is done by comparing with the popular HMAC using practical implementations. MACJER-320 produces an output of 320 bit MAC code and hence it is more secure than the 160 bit MAC code produced by HMAC-SHA-1. Due to the more number of operations performed in each message block, the MAC code produced by MACJER-320 is definitely more secure compared to HMAC-SHA-1. Since message integrity and authentication services are very important in today's high-speed network protocols and in the light of confidence levels with the current candidates like SHA-1 are coming down, new MAC schemes are necessary and more secure MAC codes like MACJER-320 could be a good option.

Chapter 5

Use of MAJE4 and MACJER-320 in Secure Socket Layer / Transport Layer Security Protocol

Abstract:

The demand for information security in Internet based applications is by and large met by the Secure Socket Layer (SSL) / Transport Layer Security (TLS) protocol, which is in use widely. It provides protection against eaves droppings, tampering and forgery. The cryptographic algorithms RC4 and HMAC have been in use for achieving security services like confidentiality and authentication in the SSL / TLS. But recent attacks against RC4 and HMAC have raised questions in the confidence on these algorithms. Hence two novel cryptographic algorithms MAJE4 and MACJER-320 have been proposed as substitutes for them. The focus of this work is to establish the performance of these new algorithms and suggest them as dependable alternatives to satisfy the need of security services in SSL / TLS. The performance evaluation has been done by using practical implementation.

5.1 Introduction

SSL protocol has been universally accepted in the World Wide Web for authenticated and encrypted communication between clients and servers. It was originally developed by Netscape, its version 1.0 was never publicly released; version 2.0 was released in 1994 but contained a number of security flaws which ultimately led to the design of version 3.0 which was released in 1996 [Transport Layer Security, 2008]. At present, SSL is widely deployed in many intranets as well as over the public Internet in the form of SSL capable servers and clients and has become the de facto standard for Transport Layer Security. The Internet Engineering Task Force (IETF) standardized SSL as an IETF standard under the name of Transport Layer Security (TLS) protocol [Allen C. and Dierks T., 1997]. The few real world, practical applications of SSL & TLS are Client Server systems, Financial systems, Information systems to create remote access and administration applications, Travel industry to create online reservation systems and secure information transfer, etc. [Security Protocol, 1999]. Visa, MasterCard, American Express and many leading financial institutions have endorsed SSL / TLS for commerce over the Internet. Some early implementations of SSL used 40-bit symmetric keys because of US government restrictions on the export of cryptographic technology. The 40-bit key size limitation has mostly gone away and modern implementations use 128-bit (or longer) keys for symmetric key ciphers.

One of the reasons that SSL has outgrown other transport and application layer security protocols such as SSH, SET, and SMIME in terms of deployment is that it is application protocol independent [George Apostolopoulos et al., 2000]. Conceptually, any application that runs over

TCP can also run over SSL. There are many examples of applications such as TELNET and FTP running transparently over SSL. However, SSL is most widely used as the secure transport layer below HTTP. A large number of e-commerce sites dealing with private and sensitive information use SSL as the secure transport layer. This number is expected to grow, as more and more businesses and users embrace electronic commerce. As security becomes an integral feature for Internet applications and the use of SSL/TLS increases, its impact on the performance of the servers as well as the clients is going to be increasingly important. Browsers like Netscape Navigator and Internet Explorer can access SSL enabled web pages by using URLs that start with https: instead of http.

The main objectives for SSL are:

1. Authenticating the client and server to each other.
2. Ensuring data integrity
3. Securing data privacy.

5.2 Motivation

In applications using SSL, the confidentiality of information is ensured using strong encryption algorithms. For very fast encryption and decryption of data for transmission after an SSL connection has been established, RC4 is the most widely used algorithm. HMAC-SHA-1 has been recommended for message authentication in several network security protocols. The key reasons behind this are the free availability, the flexibility of chaining the hash

function and the reasonable speed, among others. Even though RC4 and HMAC-SHA-1 are the most widely used ciphers of secure web applications, the strength of RC4 and SHA-1 has been called into question as a result of recent findings. Hence it is required to have proven and new methods to meet the future requirements. The analysis of novel cryptographic algorithms MAJE4 and MACJER-320 and their performance in comparison with the popular ones RC4 and HMAC-SHA1 have been done in this context and they have been proposed as alternatives.

5.3 Security Analysis of Algorithms

section 2.1.3 shows that all the five randomness tests were passed by MAJE4. The security analysis of HMAC is explained in 4.2.2.1. Sections 3.6.1 and 4.1.4 describe the security analysis of JERIM-320 and MACJER-320. Security related findings of RC4 algorithm is given in section 5.3.1.

5.3.1 RC4

Some of the published attacks on RC4 are as follows:

1. The first known weaknesses in RC4 were reported in 1995 by Ross [Roos A., 1995] and Wagner [Wagner D., 1995]. They described several classes of keys that have specific weaknesses including predictable output or output that leaks key information. Later a related key attack was observed for long keys (2048 bits) [Grosul A.I. and Wallach D.S., 2000].

2. Since the output of RC4 stream cipher is used to encrypt the plain text by bitwise XOR, any observable bias in the output can be used as the basis for an attack. A correlation was detected by Golic [Golic J.Dj., 1997] between bytes at time t and $t+2$. Many stronger correlations were later reported by Fluhrer and McGrew [Fluhrer S.R. and McGrew D.A., 2001].
3. Attacks to guess the internal state and then check for consistency with known output have been studied independently by several researchers and results were published [Mister S. and Tavares S.E., 1999]; [Knudsen L. et al., 1998] and [Golic J.Dj., 2000].
4. The most significant attacks on RC4 have been based on exploiting the simplicity of the initialization algorithm to discover an observable bias in the first few bytes of the output sequence. A bias in the second output byte also has been reported [Martin I. and Shamir A., 2001]. The value zero occurs with twice the expected probability for a random sequence. A bias in the first byte was also reported [Mironov I., 2002].
5. Fluhrer S., Martin I. and Shamir A. published a report [Fluhrer S. et al., 2001] that describes several weaknesses in the key scheduling algorithm of RC4 and proposes attacks for exploiting those weaknesses.
6. Klein [Andreas Klein, 2007] showed an improved way of attacking RC4 using related keys that does not need the ‘resolved condition’ on the IVs and gets by with a significantly reduced number of frames.

7. Subhamoy Maitra and Goutam Paul gave an independent analysis [Subhamoy Maitra and Goutam Paul, 2007] of Klein's attack with results similar to multiple key bytes extension.
8. Vaudenay and Vuagnoux presented a similar attack at SAC 2007 [Serge Vaudenay and Martin Vuagnoux, 2007] which additionally makes use of the fact that the RC4 key is stretched to 256 bytes by repeating it. The same trick was reported by Ohigashi, Ozasa, Fujikawa, Kuwadako and Morii [Toshihiro Ohigashi et al., 2007] who developed an improved version of the attack.
9. The implication of these findings is that a buffer overflow attack [Crispin Cowan et al., 1999] or a similar attack can be used to learn a single state of the generator, which can then be used to predict all random values, such as SSL keys. This attack is more severe and more efficient than other known attacks.

These problems with RC4 have raised serious alert on the security of protocols like the SSL which are using RC4 for providing confidentiality.

5.4 Alternate usage of MAJE4 and MACJER-320 in SSL / TLS Protocol

In SSL, the confidentiality of the information is achieved by using RC4 and integrity and authentication by HMAC-SHA-1. In the light of threats to these methods as described in sections 5.3.1 and 4.2.2.1, an alternative can be suggested by employing MAJE4 and MACJER-320. The MAJE4 is a 128-bit or 256-bit key algorithm and the randomness property of the stream cipher was proven by five empirical tests like frequency test, serial test, poker test,

runs test and autocorrelation test in section 2.2.5. MACJER-320 is also a competent algorithm for providing message authentication as discussed in sections 4.1.3 & 4.1.4. Hence MAJE4 and MACJER-320 algorithms can be used effectively in place of RC4 and HMAC-SHA-1 for encryption and authentication.

5.4.1 MAJE4 & RC4

5.4.1.1 Timing Analysis & Memory Requirements

From the timing analysis it can be noted that when RC4 and MAJE4 are compared, MAJE4 is almost 1.2 times faster as shown in Table 5.1. On comparing the memory required for executable files, MAJE4 was found consuming lesser space compared to RC4. The memory size required for RC4 is 8077 bytes and for MAJE4 it is 5435 bytes.

Table 5.1: Timing Analysis & Memory Requirements

PRNGs	Key length	No. of random numbers generated	No. of random bits per each random number	Total no. of bits produced (Mbps)	Memory requirements (bytes)
MAJE4	128-bit	1,15,39,399	32	352.15	5435
RC4	128-bit	3,95,99,988	8	302.12	8077

5.5 Conclusions

The SSL designers have chosen to use the then available algorithms RC4 as fast stream cipher and HMAC as hash-based construction for its security services. But recent findings show that the confidence level in these algorithms is coming down. It is clear that a transition to a newer encryption and message authentication algorithms will be required in the near future, since the information handled is very sensitive. It is in this situation that more secure algorithms MAJE4 and MACJER-320 are suggested which can definitely become good substitutes.

Chapter 6

Summary of Results, Conclusions and Future Work

Abstract:

The summary of the results of the study of PRNGs / stream ciphers, hash functions and message authentication codes are presented in this chapter. Also the performance evaluation of novel algorithms MAJE4, MARS4 and MACJER-320 for achieving confidentiality, integrity and message authentication are summarized. The performance of these algorithms were compared with the current algorithms and an alternate usage of algorithms in SSL/TLS was suggested.

6.1 MAJE4

The study of various popular Pseudo Random Number Generators and Stream Ciphers have been carried out based on statistical analysis using five randomness tests. Extensive software implementations and statistical experimentation were conducted with a view to identifying a reliable PRNG or Stream Cipher. Results of analysis and performance evaluation studies showed that JEROBOAM and RC4 are dependable as they passed all the five randomness tests. Since RC4 had undergone attacks, it was concluded to make use of the inherent qualities of JEROBOAM as basic guidelines for future development.

The next goal was to design a stream cipher which generates a long unpredictable key stream with better performance. On this view, a new fast stream cipher MAJE4 was designed with a variable key size of 128-bit or 256-bit. The randomness property of MAJE4 was analysed as shown in section 2.2.5. The performance evaluation of the stream cipher was done by comparing with JEROBOAM. This is shown in Sec 2.2.6. Processing time and memory requirements were considered as the performance parameters for the comparison. The comparison studies revealed that MAJE4 is superior to JEROBOAM in terms of both the performance parameters. Details are given in sections 2.2.6.1 and 2.2.6.2.

6.2 MARS4

Having designed the stream cipher MAJE4, which is a symmetric key algorithm, the focus was on developing a new application by combining the

two cryptographic methods - the symmetric key algorithm and the asymmetric key algorithm, with the objective of getting advantages of both. Thus, a novel and fast hybrid technique MARS4 was developed using MAJE4 and the popular asymmetric key algorithm RSA. The performance evaluation of MARS4 was done in comparison with MAJE4 and RSA. Details of the results are explained in section 2.3.4. Results have proven that MARS4 which uses MAJE4 as the stream cipher, performs much better than the systems which use RSA. It was found to be much faster than the popular RSA and its memory requirement was also less than RSA. MARS4 also provides a solution to the key exchange problem seen among symmetric key algorithms, thus preserving the advantages of both symmetric and asymmetric cryptographic systems. In short, MARS4 can be projected to be a very sound technique for transferring messages where confidentiality is of importance to the users.

6.3 Nested Hash Function

Further work was aimed at providing integrity and confidentiality of messages in a swift and cost effective manner and is described in section 2.4. A nested hash function with lower computational and storage demands is developed for providing integrity to messages. Here the hash code and the message are encrypted using MAJE4 with a limited increase in processing time and memory as shown in section 2.4.4. Nested hash functions together with MAJE4 are recommended for internet applications that require both message integrity and confidentiality. When advanced cryptographic systems with good speed and lesser memory are made available, it becomes easier to implement and manage and hence more internet users can take advantage of these benefits.

6.4 JERIM-320

Message integrity is another distinct service which needs to be ensured during data transfer through the network. To enhance security services, detailed studies on different popular Hash Functions have been done, desired properties were identified and a new hash function called JERIM-320 with improved security and reasonable speed has been designed.

Performance evaluation of JERIM-320 has been carried out by comparing with 5 popular hash functions SHA-1, SHA-256, RIPEMD-160, RIPEMD-320 and FORK-256 by using practical implementation and also by using step computation. This is given in section 3.7. A detailed comparison with FORK-256, a separate comparison with RIPEMD-320 and comparisons with the other hash functions SHA-1, SHA-256, RIPEMD-160 are also done. From the results in sections 3.8.3 and 3.8.4, it can be seen that JERIM-320 is a reliable random number generator with sufficient speed. It can also form an alternative for the present day hash functions for providing data integrity. These dual services of JERIM-320 as a hash function as well as a PRNG makes it very useful in cryptographic applications.

6.5 MACJER-320

The security services of providing confidentiality and integrity have been dealt with in the previous sections. The other two main security services demanded by the network community are message authentication and non-repudiation.

A new message authentication code called MACJER-320 with better security and reasonable speed has been proposed. This was done using JERIM-320 in combination with a 320-bit secret key.

The performance studies of MACJER-320 were then conducted. Performance evaluation of the message authentication codes MACJER-320 and HMAC-SHA1 was also done using practical implementations as given in section 4.2.3. The multiple operations on the message blocks in MACJER-320 result in much higher security with a negligible compromise in the speed of operation. MACJER-320 was concluded to provide higher security than HMAC-SHA1. Moreover in MACJER-320 the hash function can be used as a black box, so that the replacement of the underlying hash function is easily supported. Since message integrity and authentication services are very important in today's high-speed network protocols and in the light of confidence levels with the current candidates like SHA-1 are coming down, new MAC schemes are necessary and more secure MAC codes like MACJER-320 could be a good option.

6.6 Use in SSL / TLS

The Secure Socket Layer (SSL) / Transport Layer Security (TLS) protocol is the most widely used security protocol in the Internet which meets the demand for information security. The cryptographic algorithms like RC4 and HMAC have been in use for achieving security services like confidentiality and authentication in the SSL. But recent attacks against RC4 and HMAC have raised questions on the confidence of these algorithms. It is clear that a transition to a newer encryption and message authentication

algorithms will be required in the near future, since the information handled is very sensitive. It is in this situation that more secure algorithms MAJE4 and MACJER-320 are suggested which can definitely become good substitutes. The focus of this area of the work is to establish the performance of these new algorithms and suggest them as dependable alternatives.

6.7 Research Conclusions

The work includes an elaborate study on the different security services that are necessitated for conveying information reliably through the network. After unravelling the efficient algorithms in different services, improvements have been attempted on these areas resulting in proposing new algorithms. Performance studies were conducted to establish the merits of these algorithms. Altogether, this work forms a comprehensive approach on the three major dimensions of security services namely confidentiality, integrity and authentication and contributes to an enhanced security system.

6.8 Future Work

Following are the few suggestions for future work:

1. The fourth main security service called non-repudiation blocks the sender's denial that the sender had not sent a particular message. Whereas authentication of identity may be sufficient for applications where the sender needs only to convince the recipient of his / her identity, the legal requirements of many e-commerce applications require non-repudiation to be sufficiently robust for the recipient to prove to a third party such as a judge or jury that the sender's denial

was false. Conventional crypto with a single shared secret key may be sufficient for two-party authentication, but public key technology is needed for three-party non-repudiation in open systems such as the Internet. Non repudiation with public key technology can be achieved by using digital signatures. This area may be explored further.

2. Other than SSL, there are security protocols used by the network users. One is the Standard for cryptography based authentication, integrity and confidentiality services at the IP datagram layer, usually called IPsec. Another one is Pont to Point Tunneling Protocol known as PPTP which is used to create Virtual Private Network communication across the Internet. This works at the IP Datagram layer. Next is the Secure Electronic Transaction. SET allows secure credit card transactions over the Internet. Another security protocol is S/MIME (Secure MIME) which guarantees the secure transmission, storage and authentication and forwarding of secret data at the application layer. SSH (Secure shell) Protocol that permits users secure remote access over a network from one computer to another is also a protocol which is widely used.

In all these protocols, the asymmetric key algorithm RSA has been used generally for providing confidentiality. Asymmetric key algorithms are usually slower and consume a lot of memory. During our studies, RSA was found to be less efficient in terms of processing time and memory consumption. Hence development of a secure asymmetric algorithm which is more efficient than the RSA will be a useful work.

3. The Hash function, JERIM-320 can be made available in several different strengths, by varying the number of rounds. Another innovation is possible by accommodating a variety of hash output sizes. These improvements will help to customize the security requirements.

Published Work of the Author

1. Sheena Mathew, K.Paulose Jacob, "A New Fast Stream Cipher: MAJE4", Proceedings of the International Conference IEEE INDICON 2005, pp 60-63, IIT Madras, Chennai.
2. Sheena Mathew, K.Paulose Jacob, "Performance evaluation of Pseudo Random Number Generators- A Statistical Analysis", Proceedings of International Conference on Resource Utilisation and Intelligent Systems, INCRUIS 2006, pp 224-229, Kongu Engg. College, Erode.
3. Sheena Mathew, K.Paulose Jacob, "A Novel Fast Hybrid Cryptographic System: MARS4", Proceedings of IEEE INDICON 2006, pp1-5, New Delhi.
4. Sheena Mathew, K.Paulose Jacob, "Message Integrity in the World Wide Web: Use of Nested Hash Function and a Fast Stream Cipher", Proceedings of 14th International Conference on Advanced Computing and Communication, pp 147-150, ADCOM 2006, NIT, Surathkal.

This paper has since been published in the

International Journal of Information Processing, pp 19-25, Vol. 1, No. 2, 2007.

5. Sheena Mathew, K.Poulose Jacob, "A Novel High Security Message Authentication Code: MACJER-320 and its Performance Evaluation", The Icfai Journal of Computer Sciences, pp 16-28, Vol II, No.1, January 2008.

Papers Accepted and awaiting publication:-

6. Sheena Mathew, K.Poulose Jacob, "JERIM-320: A New 320-bit Hash Function with Higher Security", International Journal of Computers, Systems and Signals.
7. Sheena Mathew, K.Poulose Jacob, "JERIM-320: A 320-bit Hash Function with Dual Applications - A Statistical Analysis", Journal of Discrete Mathematical Science & Cryptography.
8. Sheena Mathew, K.Poulose Jacob, "JERIM-320: A New 320-bit Hash Function Compared to Hash Functions with Parallel Branches", International Journal of Computer Science and Applications.

Papers Communicated:-

9. Sheena Mathew, K.Poulose Jacob, "Performance Evaluation of a Novel Hash Function JERIM-320 in Comparison with Popular Hash Functions".
10. Sheena Mathew, K.Poulose Jacob, "MACJER-320: A New Hash Based Message Authentication Code".
11. Sheena Mathew, K.Poulose Jacob, "Use of Novel Algorithms MAJE4 and MACJER-320 for achieving confidentiality and message Authentication in SSL & TLS".

Bibliography

1. Alan O. Freier, Philip Karlton and Paul C. Kocher, “The SSL Protocol Version 3.0”, Internet Draft, March 1996,
URL: <http://www.freesoft.org/CIE/Topics/ssl-draft/INDEX.HTM>
2. Allen C. and Dierks T., “The TLS Protocol Version 1.0”, Internet Draft, Internet Engineering Task Force, November 1997,
URL: <http://tools.ietf.org/html/rfc2246>
3. Andreas Klein, “Attacks on the RC4 stream cipher”, Journal of Designs, Codes and Cryptography, Springer Netherlands, 2008 URL:
http://www.quequero.org/uicwiki/images/Attacks_on_the_RC4_stream_cipher.pdf
4. Atul Kahate, Cryptography and Network Security, Tata McGraw-Hill, 2005
5. Bart Preneel, Vincent Rijmen, and Antoon Bosselaers, “Recent Developments in the Design of Conventional Cryptographic Algorithms” 1998,
URL: <ftp://ftp.esat.kuleuven.ac.be/cosic/bosselaer/algo.pdf>
6. Bellare M. and Rogaway P, “Optimal Asymmetric Encryption”, Proc. of Eurocrypt’94, LNCS 950, Springer-Verlag 1995, pp.92-111
7. Ben N. Venzke, “Economic Industrial Espionage”, July 14th 2002,
URL: <http://www.computerconsultants.com/news1.htm>

8. Biham E. and Chen R., “Near Collisions of SHA-0”, *Advances in Cryptology- CRYPTO*, LNCS 3152, Springer- Verlag, 2004, pp. 290-305
9. Biham E. and Chen R., Joux A., Carribault P., Lemuet C., and Jalby W., “Collisions of SHA-0 and Reduced SHA-1”, *Advances in Cryptology-EUROCRYPT*, LNCS 3494, Springer- Verlag, 2005, pp.36-57
10. Blum M., Blum L. and M. Shub , “A Simple Unpredictable Pseudo Random Number Generator,” *SIAM Journal of Computing*, 1986
11. Bo Yang, Ramesh Karri, David A. McGrew, “A High- Speed Hardware Architecture for Universal Message Authentication Code”, *IEEE Journal on Selected Areas in Communications*, Vol 24, No.10, 2006, pp. 1831-1839
12. Boyar J., “Inferring Sequences Produced by Pseudo-Random Number Generators”, *Journal of ACM*, Vol.36(1), Jan 1989, pp. 129-141
13. Braden R., “RFC 1122: Requirements for Internet Hosts— Communication Layers”, October 1989,
URL: <http://www.isi.edu/in-notes/rfc1122.txt>
14. Bruce Schneier, *Applied Cryptography*, John Wiley and Sons, 1996
15. Chabaud F. and Joux A., “Differential Collisions in SHA-0”, *Advances in Cryptology- CRYPTO*, LNCS 1462, Springer- Verlag, 1998, pp. 56-71

16. Christian Rechberger and Vincent Rijmen, “Note on Distinguishing, Forgery, and Second Preimage Attacks on HMAC-SHA-1 and a Method to Reduce the Key Entropy of NMAC”, 2006,
URL:
<http://citeseer.ist.psu.edu/cache/papers/cs2/338/http:zSzzSzszprint.iacr.orgzSz2006zSz290.pdf/note-on-distinguishing-forgery.pdf>
17. Coskun B. and Memon N., “Confusion/diffusion capabilities of some robust hash functions,” Proc. CISS, Conf. on Information Sciences and Systems, March 2006, pp. 1188 – 1193.
18. Crispin Cowan, Perry Wagle, Calton Pu, Steve Beattie, and Jonathan Walpole, “Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade”, 1999,
URL:<http://www.ece.cmu.edu/~adrian/630-f04/readings/cowan-vulnerability.pdf>
19. Dobbertin H., “Cryptanalysis of MD4”, Fast Software Encryption, LNCS 1039, Springer-Verlag, 1996, pp. 53-69.
20. Dobbertin H., Bosselaers A., Preneel B., “RIPEMD-160, a strengthened version of RIPEMD”. Fast Software Encryption, LNCS 1039, Springer- Verlag, 1996, pp. 71-82.
21. Erhan K, “Properties of Secure Hash Functions”, Denim Group, Nov. 2007, URL:http://denimgroup.typepad.com/denim_group/2007/11/properties-of-1.html
22. Fluhrer S., Mantin I., Shamir A., “ Weakness in the key scheduling

- Algorithm of RC4”, Proceedings in the selected Areas in Cryptography 2001, SAC’01, LNCS vol.2259, Springer-Verlag, 2001, pp. 1-24.
23. Fluhrer S.R. and McGrew D.A., “Statistical Analysis of the Alleged RC4 Keystream Generator”, Proceedings of Fast Software Encryption 2000, LNCS vol. 1978, Springer-Verlag, 2001, pp.19-30.
 24. Frank Rubin, “One-Time pad Cryptography”, 1997,
URL: <http://www.mastersoftware.biz/crypt005.htm>
 25. Frier A., Karlton P., and Kocher P., “The SSL 3.0 Protocol”, Netscape Communications Corporation, Nov. 1996,
URL: <http://home.de.netscape.com/eng/ssl3/draft302.txt>
 26. Fujisaki E. and Okamoto T., “How to Enhance the Security of Public-Key Encryption at Minimum Cost”, Proc. Of PKC’ 99, Springer-Verlag, LNCS 1560, 1999a, pp.53-68.
 27. Fujisaki E. and Okamoto T., “Secure Integration of Asymmetric and Symmetric Encryption Schemes”, Proc. Of Crypto’99, Springer – Verlag LNCS 1666, 1999b, pp. 535-554.
 28. Gene Tsudik, “Message Authentication with One-Way Hash Functions”, Proceedings of IEEE-INFOCOM 92, pp. 2055-2059.
 29. George Apostolopoulos, Vinod peris, Prashant Pradhan, Debanjan Sahi, “Securing Electronic Commerce: Reducing the SSL Overhead”, IEEE Network, 14(4), July 2000, pp. 8-16.

30. Golic J.Dj., "Linear statistical Weakness of alleged RC4 keystream generator", *Advances in Cryptology – Eurocrypt 97*, LNCS vol. 1233, Springer-Verlag, 1997, pp.226-238.
31. Golic J.Dj., "Iterative Probabilistic Cryptanalysis of RC4 Keystream Generator", *Proceedings of ACISP 2000*, LNCS vol.1841, Springer – Verlag, 2000, pp. 220-233.
32. Grosul A.I and Wallach D.S, "A Related Key Cryptanalysis of RC4", Manuscript from Department of Computer Science, Rice University, 6 June 2000.
33. Halton J. H., "A Retrospective and Prospective Survey of the Monte Carlo Method," *SIAM Review*, 1970, 12(1): pp.1-63,
URL:http://www.cs.fsu.edu/~mascagni/Halton_SIAM_Review_1970.pdf
34. Helena Handschuh, Gemplus, Bart Preneel and K.U.Leuven, "Minding Your MAC Algorithms", 2004,
URL:http://www.gemplusbanknote.com.br/smart/rd/publications/pdf/H_P04isb.pdf
35. Herve Chabanne and Emmanuel Michon, "Jeroboam", *Fast Software Encryptions Conference*, Springer LNCS, 1998.
36. Hong D., chang D., Sung J., Lee S., Hong S., Lee J., Moon D., Chee S., "A New Dedicated 256-Bit Hash Function : FORK-256", *Fast Software Encryption* , LNCS 4047, Springer- Verlag, 2006, pp. 195-209.

37. Ivan Damgard, "A design principle for hash functions"., Advances in Cryptology – CRYPTO '89 Volume 435 of Lecturer Notes in Computer Science, Berlin, NewYork, Tokyo, Springer – Verlag, 1990, pp. 416-427.
38. Jiqiang Lu, Jongsung Kim, Nathan Keller and Orr Dunkelman, "Differential and Rectangle Attacks on Reduced-Round SHACAL-1", 2006, URL:<http://www.ma.huji.ac.il/~nkeller/article-853.pdf>
39. John Viega, "Practical Random Number Generation in Software", Proc. 19th Annual Computer Security Applications Conference, 2003, URL: <http://www.acsac.org/2003/papers/79.pdf>
40. Jongsung Kim, Alex Biryukov, Bart Preneel, Seokhie Hong "On the Security of HMAC and NMAC Based on HAVAL, MD4, MD5, SHA-0 and SHA-1", Proceedings of SCN 2006, LNCS, Springer-Verlag.
41. Kaukonen K. and Thayer R., December 1999, "A Stream Cipher Encryption Algorithm "Arcfour"", URL:<http://www.mozilla.org/projects/security/pki/nss/draft-kaukonen-cipher-arcfour-03.txt>
42. Kencheng Zeng, Chung-Huang Yang, Dah-Yea Wei and T.R.N. Rao, "Pseudorandom Bit Generators in Stream-Cipher Cryptography", IEEE Computer, February 1991, pp. 8-17
43. Knudsen L., Meier W., Preneel B., Rijmen V. and Verdoolaeye S., "Analysis methods for (alleged) RC4", Advances in Cryptology – AsiaCrypt 98, LNCS vol.1514, Springer -Verlag, 1998, pp.327-341.

44. Knuth D.E., *The Art of Computer Programming: Vol.2, Seminumerical Algorithms*, Third Edition, Addison –Wesley, 1997.
45. Lifeng Lai, Hesham El Gamal and H. Vincent Poor, “Authentication over Noisy Channels”, 2008,
URL:http://arxiv.org/PS_cache/arxiv/pdf/0802/0802.2701v1.pdf
46. Mantin I. and Shamir A., “A Practical Attack on Broadcast RC4”, *Proceedings of Fast Software Encryption*, 2001, LNCS, vol.xx, pp.152-164, Springer-Verlag, 2001.
47. Matusiewicz K., Contini S., Pipeprzyk J., “Weaknesses of the FORK-256 compression function”, 2007a,
URL:<http://eprint.iacr.org/2006/317.pdf>
48. Matusiewicz K., Thomas Peyrin, Olivier Billet, Scott Contini and Josef Pieprzyk, “Cryptanalysis of FORK-256”, *Fast Software Encryption*, LNCS 4593, Springer- Verlag, 2007b, pp.19-38.
49. Menezes A., van Oorschot P. and Vanstone S., *Handbook of Applied Cryptography*, CRC Press, 1997,
URL:<http://www.cacr.math.uwaterloo.ca/hac/about/chap5.pdf>
50. Michael Chernick C., Charles Edington III, Matthew J. Fanto, Rob Rosenthal, “Guidelines for the Selection and Use of Transport Layer Security (TLS) Implementations”, June 2005,
URL:<http://csrc.nist.gov/publications/nistpubs/800-52/SP800-52.pdf>

51. Mihir Bellare, Roch Guerin, Philip Rogaway, “XOR MACs: New Methods for Message Authentication using Finite Pseudorandom Functions”, *Advances in Cryptology – Crypto 95 Proceedings, Lecturer Notes in Computer Science Vol. 963*, D.Coppersmith ed. Springer – Verlag, 1995.
52. Mihir Bellare, Ran Canetti, Hugo Krawczyk , “Message Authentication Using Hash Functions – The MAC Construction”, *RSA Laboratories CryptoBytes, Vol.2, No.1*, Springer- Verlag, 1996a.
53. Mihir Bellare, Ran Canetti, Hugo Krawczyk , “Keying Hash Functions for Message Authentication”, *Advances in Cryptology – Crypto 96 Proceedings, Lecturer Notes in Computer Science Vol. 1109*, Springer – Verlag, 1996b.
54. Mihir Bellare, Joe Kilian and Phillip Rogaway, “The Security of the Cipher Block Chaining Message Authentication Code”, July 1999, URL: <http://www.cs.ucdavis.edu/research/tech-reports/1997/CSE-97-15.pdf>
55. Mironov I., “(Not so) Random Shuffles of RC4”, *Advances in Cryptology –CRYPTO-2002, LNCS vol.2442*, Springer Verlag, 2002, pp. 304-319.
56. Mister S. and Tavares S.E., “Cryptanalysis of RC4-like Ciphers”, *Proceedings of SAC’98, LNCS vol. 1556*, Springer- Verlag, 1999, pp.131-143.

57. Mustak E. Yalcin, Johan A. K. Suykens and Joos Vandewalle, "True Random Bit Generation From a Double-Scroll Attractor", IEEE Transactions on Circuits and Systems, Vol. 51, No.7, July 2004, pp 1395-1404.
58. National Institute for Standards and Technology, "Data Encryption Standard (DES) FIPS PUB 46-3", U.S. Department of Commerce, Oct. 1999, URL: <http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf>
59. National Institute for Standards and Technology, "Digital Signature Standard (DSS) FIPS PUB 186-2", U.S. Department of Commerce, Jan. 2000, URL: <http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf>
60. National Institute for Standards and Technology, "Advanced Encryption Standard (AES) FIPS PUB 197", U.S. Department of Commerce, Nov. 2001, URL:<http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>
61. National Institute of Standards and Technology (NIST), "FIPS-180-2: Secure Hash Standard", 2002, URL:[http://csrc.nist.gov/publications/fips/fips 180-2/fips 180-2.pdf](http://csrc.nist.gov/publications/fips/fips%20180-2/fips%20180-2.pdf)
62. National Institute of Standards and Technology, Federal Information Processing Standards (FIPS) Publication 198, "The Keyed-Hash Message Authentication Code (HMAC)", March 2002, URL:<http://csrc.nist.gov/publications/fips/fips198/fips-198a.pdf>

63. National Institute of Standards and Technology, “Federal Information Processing Standards (FIPS) Publication 180-2”, Secure Hash Standard (SHS), February, 2004
64. National Institute for Standards and Technology, “DRAFT RSA Strong Primes - Digital Signature Standard (DSS) FIPS PUB 186-3”, U.S. Department of Commerce, Dec. 2007,
URL:http://csrc.nist.gov/publications/drafts/fips_186-3/fips186-3_Strong-Prime-Sections_Dec2007.pdf
65. Niels Ferguson, “Twofish Technical Report #1-Upper bounds on differential characteristics in Twofish”, 1998,
URL:<http://www.schneier.com/paper-twofish-differential.pdf>
66. Nowell Security Enforcement, “Preventing Industrial Espionage”, 2003, URL:<http://www.nowellgroup.com/index.php?page=espionage>
67. Park Stephen K. and Keith W. Miller, “Random Number Generators : Good ones are hard to find”, Communications of the ACM, October 1988, pp.1192-1201.
68. Philip Hawkes, Michael Paddon, and Gregory G. Rose, “ On corrective patterns for the SHA-2 family”, Cryptology ePrint Archive, Report 2004/207, August 2004, URL: <http://eprint.iacr.org/2004/207.pdf>
69. The prime pages, “Finding primes, proving primality”, Sept. 2006, URL: http://primes.utm.edu/prove/prove2_3.html

Bibliography

70. Raphabel C, W.Phan and David Wanger, "Security considerations for incremental hash functions based on pair block chaining", *Journal of Computers & Security* 25(2006), pp 131-136.
71. Ralph C.Merkle, "One way hash functions and DES", *Inadvances in Cryptology – CRYPTO '89 Volume 435 of Lecturer Notes in Computer Science*, Berlin, NewYork, Tokyo, Springer – Verlag, 1990, pp. 428-446.
72. Richard Clayton, "Brute force attacks on cryptographic keys", Oct. 2001, URL: <http://www.cl.cam.ac.uk/~rnc1/brute.html>
73. Richard Power, "Richard Power On Corporate Espionage", September 15th 2000, URL:<http://www.techtv.com/cybercrime/print/0,23102,12005,00.html>.
74. Ritter T., "The Efficient Generation of Cryptographic Confusion Sequences," *Cryptologia*, 15(2), 1991, pp. 81-139.
75. Rivest R.L., "The MD4 message digest algorithm", *Advances in Cryptology-CRYPTO*, LNCS 537, Springer-Verlag, 1990, pp.303-311.
76. Rivest R.L., "The MD5 message digest algorithm", (RFC 1320), Internet Activities Board, Internet Privacy Task Force, 1992.
77. Roos A., "A Class of weak keys in the RC4 stream cipher", *sci.crypt*, 1995.

78. RSA Laboratories, "PKCS #1 v2.1: RSA Encryption Standard", June 2002, URL: <ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf>.
79. Security Protocols Overview An RSA Data Security Brief, 1999, URL: <http://www.comms.scitech.susx.ac.uk/fft/crypto/security-protocols.pdf>
80. Seigenthaler T., "Decrypting a class of Stream Ciphers Using Cipher text Only", IEEE Transactions on computer, Vol C-34, No.1, Jan 1985, pp. 81-85
81. Serge Vaudenay and Martin Vuagnoux, "Passive-only key recovery attacks on RC4". In Selected Areas in Cryptography 2007, Lecturer Notes in Computer Science, Springer 2007
82. SmartCards Expo, 2007, URL:http://www.electronicstoday.org/e_security2007.htm
83. "Specification of the Bluetooth system", vol.1.1, Feb. 2001, URL:<http://www.bluetooth.com/dev/specifications.asp>
84. Standards for Efficient Cryptography Group (SECG), "SEC 1: Elliptic Curve Cryptography, Version 1.0", September 20, 2000. URL:http://www.secg.org/download/aid-385/sec1_final.pdf
85. Stanford University, "Network security resources and reporting problems", School of Earth Sciences, Oct. 2006, URL:<http://pangea.stanford.edu/computerinfo/resources/network/security/>

86. Subariah Ibrahim, Mohd Aizaini Maarof and Norbik Bashah Idris, "Avalanche Analysis of Extended Feistel Network", 2005, URL:http://eprints.utm.my/3258/1/Subariah_PARS05.pdf
87. Subhamoy Maitra and Goutam Paul, "Many keystream bytes of RC4 leak secret key information", Cryptology ePrint Archive, Report 2007/261, 2007, URL: <http://eprint.iacr.org/>.
88. Sultan Weatherspoon, "Overview of IEEE 802.11b Security", Intel Technology Journal Q2, 2000, pp. 1-5, URL:ftp://download.intel.com/technology/itj/q22000/pdf/art_5.pdf
89. Thomas Calabrese, "Information Security Intelligence Cryptographic Principles and Applications", Thomson Delmar Learning, India 2006.
90. Toshihiro Ohigashi, Hidenori Kuwakado and Masakatu Morii, "A Key Recovery Attack on WEP with Less Packets", Technical Report of IEICE, ISEC Nov., 2007.
91. Transport layer Security, Wikipedia, 2008 URL:http://en.wikipedia.org/wiki/Secure_Sockets_Layer
92. Vazirani U. and Vazirani V., "Efficient and Secure Pseudo Random Number Generation (extended abstract) Advances in Applied Cryptology," Proceedings of CRYPTO 84, Springer – Verlag, 1985, pp. 193-202.
93. Wagner D., " My RC4 weak keys", sci.crypt, September 1995.

94. Wagner D., "A Generalized Birthday Problem", Proceedings of Crypto '02, LNCS vol. 2442, Springer-Verlag, 2002.
95. WAP Forum, "Wireless Application Protocol Architecture Specification, WAP-100-WAPArch-19980430-a", WAP Forum Specifications 30-Apr-1998 version,
URL:<http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html>
96. WAP Forum, "Wireless Application Protocol Wireless Transport Layer Security Specification, WAP-199-WTLS-20000218-a", WAP Forum Specifications 18-Feb-2000 version,
URL:<http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html>
97. Wei Zeng D., Yang K. C. and Rao T., "Pseudo Random bit Generators in Stream Cipher Cryptography," IEEE Computer, 1991.
98. William Aiello, Sivaramakrishnan Rajagopalan and Ramarathnam Venkatesan, "Design of Practical and Provably Good Random Number Generators", SODA '95, ACM Jan 1995, pp. 1-9.
99. William Stallings, Network Security Essentials- Applications and Standards, Pearson Education, 2001.
100. William Stallings, Cryptography and Network Security- Principles and Practices, Pearson Education, 2003.

101. Williams H.C., “A Modification of the RSA Public key Encryption Procedure”, IEEE Trans. on Information Theory, Vol. IT-26, No.6, 1980, pp.726-729.
102. Xiaoyun Wang and Hongbo Yu, “How to break MD5 and other hash functions”, Advances in Cryptology – EUROCRYPT, LNCS 3494, Springer-Verlag, 2005, pp. 19-35.
103. Yuan li and T. W. Cusick, “Strict Avalanche Criterion Over Finite Fields”, 2005, URL: <http://eprint.iacr.org/2005/361.pdf>