

*Republic of Iraq  
Ministry of Higher Education  
and Scientific Research  
Al-Nahrain University  
College of Science*



# *Improving the Security of HTML Files*

*A Thesis Submitted to the College of Science,  
Al-Nahrain University in Partial Fulfillment  
of the Requirements for the Degree of  
Master of Science in  
Computer Science*

**Submitted by**

**Nadia Fadhil Al-Bakri**

**(B.Sc. 1988)**

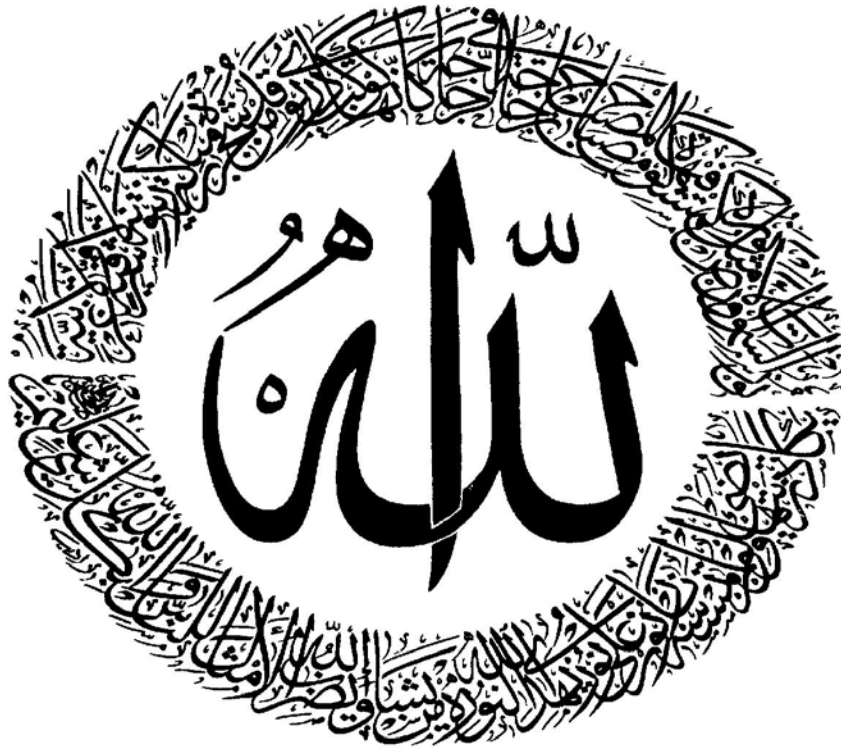
**Supervised by**

**Dr. Loay E. George**

**September 2009**

**Ramadan 1430**

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ



صدق الله العظيم

من سورة النور 34

## Supervisor Certification

I certify that this thesis was prepared under our supervision at the Department of Computer Science/College of Science/Al-Nahrain University, by **Nadia Fadhil Ibrahim** as partial fulfillment of the requirements for the degree of Master of Science in Computer Science.

Signature:



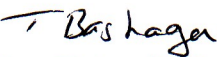
Name : **Dr. Loay E. George**

Title : **Assistant Professor**

Date : *10 / 8 / 2009*

In view of the available recommendations, I forward this thesis for debate by the examination committee.

Signature:



Name : **Dr. Taha S. Bashaga**


Title : **Head of the department of Computer Science,  
Al-Nahrain University.**


Date : *10 / 8 / 2009*


## ***Examining committee certification***

We certify that we have read this thesis and as an examining committee, Examined the student in its content and what is related to it and that in our opinion it meets the standard of a thesis for the degree of Master of Science in computer science.


### ***Examining Committee Certification***

Signature:   
Name : Dr. Abdul-Karim A-R Kadhim  
Title : Assis. Prof. (chairman)  
Date : 25/11/2009

Signature:   
Name : Dr. Jane Jaleel Stephan  
Title : Assis. Prof. (member)  
Date : 25/11/2009

Signature:   
Name : Dr. Abeer M. Yousif  
Title : Lecturer (member)  
Date : 25/11/2009

### ***Supervisor Certification***

Signature:   
Name : Dr. Loay E. George  
Title : Assis. Prof  
Date : 25/11/2009

### ***The Dean of the College Certification***

Approved by the Council of the College of Science

Signature:  
Name : Dr. Laith Abdul Aziz Al-Ani  
Title : The Dean of College of Science, Al-Nahrain University.  
Date : / /2009

# *Dedication*

*To the greatest father a person could ever have ...*

*What ever I am ... What ever I become ... I owe it  
to you.*

*To my mother the most generous woman ...*

*To my husband for his patience, and support...*

*To my beloved brother ...*

*To my beautiful daughters: Dania and Zena ...*

*You are the cause of my success ...*

*Nadia*

## *Acknowledgment*

*Praise be to God whose will and guidance enables me to complete my research.*

*I would like to express my deepest thanks to my supervisor Dr.Loay E. George for his continued invaluable guidance, for so many fruitful discussions and sharing his ideas and for honest efforts given throughout the progress of this research.*

*Grateful thanks for the Dean of the College of Science Dr. Laith Abdul Aziz Al-Ani for his attention and encouragement.*

*Grateful thanks for the Head of the Department of Computer Science Dr.Taha S.Bashaga for the continuous support during the period of my studies.*

*Also, I wish to thank all the doctors and staff in computer science department at AL-Nahrain University for their help and giving me advices.*

*I wish to express my special thanks to my true friend Dr.Nidaa Flaih for her encouragement and friendship.*

*Nadia*

# *Abstract*

Due to its ease and low cost, Internet is considered as the most important media for securely transferring information contained in WebPages. These WebPages are established using Hypertext Markup Language (HTML).

The main objective of this research is to provide a way to prevent the illegitimate reading of HTML file contents. The security requirements we considered in securing HTML file contents are confidentiality, which is accomplished using encryption methods and integrity detection that is accomplished using message digest-5 (MD5) hash function. In the established HTML Secure System (HSS), three encoding methods were used; they are: random number generator, one-way hash function and Base64 encoding.

**Two random number generators** have been used and tested for generating random numbers for stream ciphering the text, color and font that may found within HTML files (plain text). **The first random generator** is the traditional Linear Feedback Shift Register (LFSR), the periodicity of its key stream is about  $2^{32}$  bit long (4GByte). **The second random generator** is a proposed random number generator; the periodicity of its key stream is about  $2^N$ , where N ( $N > 32$ ) is a variable value depends on the length of the internal linear generators used in its scheme. Tests have been conducted to study the performance of the two generators; it is found that the use of the proposed random generator reduces the elapsed time required for encryption and decryption processes in range between (30% to 40%) in comparison with the time required by LFSR generator. In addition, the proposed random generator produces longer key stream. Five statistical random tests were performed on the cipher text for the two generators; the results indicated an acceptable level of randomness.

# Table of Contents

## **Chapter One: Introduction**

1.1 Overview.....	1
1.2 HTML's Role on the Web.....	1
1.2.1 Markup Languages.....	2
1.2.2 HTML Obstructs.....	3
1.3 Web Security Aspects.....	4
1.3.1 Web Security Breach.....	4
1.3.2 Securing Information in Transit.....	4
1.4 Cryptography and Web Security.....	5
1.5 Literature Review.....	6
1.6 Aim of Thesis.....	8
1.7 Thesis Outline.....	8

## **Chapter Two: Theoretical Concepts**

2.1 Introduction.....	10
2.2 Beneath the World Wide Web.....	10
2.2.1 Hyper Text Markup Language (HTML).....	11
2.2.2 Browsers.....	23
2.2.3 Types of Browsers.....	24
2.2.4 Hyper Text Transfer Protocol (HTTP).....	24
2.3 Web Security.....	26
2.3.1 Cryptography.....	27
2.3.2 Messages and Encryption.....	27
2.3.3 Cryptographic Protocols.....	29



2.3.4 Symmetric Cryptosystems.....	30
2.3.5 Public-Key Cryptosystems.....	31
2.3.6 Stream Cipher.....	33
2.3.7 Pseudo-Random Number Generator (PRNG).....	35
2.3.8 Linear Feedback Shift Register (LFSR).....	35
2.3.9 Randomness Statistical Tests.....	36
2.3.10 The $\chi^2$ Distribution.....	40
2.4 Cryptographic Hash Function.....	40
2.5 Message Digest 5 (MD5).....	41
2.6 Encoding.....	43
2.7 Base64 Encoding.....	<b>Error</b> 44
<b>Bookmark not defined.....</b>	

### **Chapter Three: The Improved HSS-Scheme**

3.1 Introduction.....	50
3.2 System Model.....	51
3.3 Encoding Module.....	51
3.3.1 Lexical HTML Analyzer.....	53
3.3.2 Encrypting HTML File.....	56
3.3.3 Random Number Generator.....	58
3.3.4 HSS Encoder.....	64
3.3.5 MD5 (Message Digest) Function.....	68
3.3.6 Base64 Encoder.....	70
3.4 Decoding Module.....	73
3.5 HSS Decoder.....	73

### **Chapter Four: Implementation and Testing**

4.1 Introduction.....	82
-----------------------	----

4.2 System Implementation.....	83
4.3 System Evaluation.....	96
4.3.1 Randomness Tests.....	96
4.3.2 The Elapsed Encoding Time.....	112

## **Chapter Five: Conclusions and Future Works**

5.1 Introduction.....	116
5.2 Conclusions.....	116
5.3 Future Works.....	118
List of References.....	119
Appendix A ... MD5 Algorithm .....	A1
Appendix B... Common Character Entities.....	B1
Appendix C...The Selected Percentiles of the $\chi^2$ (Chi-Square) Distribution	C1

## List of Abbreviations

<b>Abbreviation</b>	<b>Meaning</b>
<b>ASCII</b>	American Standard Code for Information Interchange
<b>EBCDIC</b>	Extended Binary Coded Decimal Interchange Code
<b>HSS</b>	HTML Secure System
<b>HTML</b>	Hyper Text Markup Language
<b>HTTP</b>	Hyper Text Transfer Protocol
<b>HTTPS</b>	Hyper Text Transport Protocol over SSL
<b>HR</b>	Human Resources
<b>LFSR</b>	Linear Feedback Shift Register
<b>MAC</b>	Message Authentication Code
<b>MD5</b>	Message Digest 5
<b>PRNG</b>	Pseudo Random Number Generator
<b>SHTTP</b>	Secure Hyper Text Transport Protocol
<b>SSL</b>	Secure Sockets Layer
<b>URL</b>	Uniform Resource Locator
<b>WWW</b>	World Wide Web
<b>W3C</b>	World Wide Web Consortium
<b>XML</b>	Extensible Markup Language
<b>XOR</b>	Exclusive Or

## List of Figures

Figure No.	Description	Page No.
2.1	The generic structure of HTML document	15
2.2	Font attributes ( face, size and color)	22
2.3	Webpage in Microsoft Internet Explorer	22
2.4	Encryption and decryption	28
2.5	Simplified model of symmetric encryption	30
2.6	Public-key encryption	32
2.7	Stream cipher diagram	34
2.8	PRNG using stream cipher	35
2.9	Linear Feedback Shift Register	36
2.10	Hash function mechanism	41
2.11	Message digest generation using MD5	42
2.12	One MD5 operation within a round	43
2.13	Base64 encoding diagram	45
2.14	Base64 encoding	47
2.15	Hexadecimal encoding	48
3.1	The structure of HSS	52
3.2	The process of filling the first location of R_Seed vector	62
3.3	HTML digest generation using MD5 hash function	69
3.4	The integrity check process	79
4.1	HSS interface	83
4.2	HSS main menu	83

<b>Figure No.</b>	<b>Description</b>	<b>Page No.</b>
4.3	File sub menu	84
4.4	Process sub menu	84
4.5	HTML file encryption sub menu	85
4.6	Input text box for secret key assignment	85
4.7	Menu of HTML files	86
4.8	The contents of an HTML file sample	87
4.9	The source code of HTML file sample	87
4.10	The content of encrypted HTML file sample using the proposed random generator	88
4.11	The source code of encrypted HTML file sample using the proposed random generator	88
4.12	The content of encrypted HTML file sample using LFSR generator	89
4.13	The source code of encrypted HTML file sample using LFSR generator	89
4.14	The color sub menu	90
4.15	The encrypted HTML file sample after color encryption	91
4.16	The source code of encrypted HTML file after color encryption	91
4.17	HTML file sample1	92
4.18	The encrypted colors using color names	93
4.19	The font sub menu	94
4.20	HTML file sample2	94
4.21	The encrypted HTML file (text, color, font size) encryption	95
4.22	The results of five randomness tests for the ciphered HTML file1 using the proposed random generator	98

<b>Figure No.</b>	<b>Description</b>	<b>Page No.</b>
4.23	The results of five randomness tests for the ciphered HTML file1 using LFSR generator	99
4.24	The results of five randomness tests for the ciphered HTML file2 using the proposed random generator	101
4.25	The results of five randomness tests for the ciphered HTML file2 using LFSR generator	102
4.26	The results of five randomness tests for the ciphered HTML file3 using the proposed random generator	103
4.27	The results of five randomness tests for the ciphered HTML file3 using LFSR generator	104
4.28	The results of five randomness tests for the ciphered HTML file4 using the proposed random generator	106
4.29	The results of five randomness tests for the ciphered HTML file4 using LFSR generator	107
4.30	The results of five randomness tests for the ciphered HTML file5 using the proposed random generator	108
4.31	The results of five randomness tests for the ciphered HTML file5 using LFSR generator	109
4.32	The results of five randomness tests for the ciphered HTML file6 using the proposed random generator	111
4.33	The results of five randomness tests for the ciphered HTML file6 using LFSR generator	112
4.34	The variation of elapsed encoding time versus the file size	113
4.35	The variation of elapsed decoding time versus the file size	113

## List of Tables

Table No.	Description	Page No.
2.1	Text formatting elements	17
2.2	HTML simple formatting elements	18
2.3	Body attributes and their functions	19
2.4	Sixteen common color values	20
2.5	The list of attributes of the font tags	21
2.6	The base64 encoding	45
4.1	The randomness tests result for the ciphered HTML file1 using the proposed random generator	97
4.2	The randomness tests result for the ciphered HTML file1 using LFSR generator	99
4.3	The randomness tests result for the ciphered HTML file2 using the proposed random generator	100
4.4	The randomness tests result for the ciphered HTML file2 using LFSR generator	101
4.5	The randomness tests result for the ciphered HTML file3 using the proposed random generator	102
4.6	The randomness tests result for the ciphered HTML file3 using LFSR generator	104
4.7	The randomness tests result for the ciphered HTML file4 using the proposed random generator	105
4.8	The randomness tests result for the ciphered HTML file4 using LFSR generator	106
4.9	The randomness tests result for the ciphered HTML file5 using the proposed random generator	107
4.10	The randomness tests result for the ciphered HTML file5 using LFSR generator	109
4.11	The randomness tests result for the ciphered HTML file6 using the proposed random generator	110
4.12	The randomness tests result for the ciphered HTML file6 using LFSR generator	111
4.13	The elapsed encoding and decoding time values when using one of the two generators	114

# *Chapter One*

## *Introduction*

### **1.1 Overview**

Web sites are now a key asset to organizations of all sizes, providing information and services to clients, marketers, suppliers, and employees. Unfortunately, these developments have opened new security threats to the enterprise networks, and opened the door to an increasing number of threats to individual and business computers. There is a growing trend of hackers attacking networks via home and remote users. These attacks can be range from partial loss of data to making the system no usable; to privacy can be completely violated. In the year 2005, the rate of threats had increased by almost 50%, as cyber criminals joined forces to create targeted malware attacks for financial gain [Rad07].

Authentication, message integrity and encryption are very important in cultivating, improving, and promoting Internet security. To protect users from Internet based attacks and to provide adequate solutions when security is imposed. cryptographic techniques must be employed to solve these problems. Message integrity is required because data may be altered as it travels through the Internet. Without confidentiality by encryption, information may become truly public [Rhe03].

### **1.2 HTML's Role on the Web**

The World Wide Web (WWW) is composed of millions of Web pages. A browser (when requested) can present one Webpage at a time. A Webpage is generally a single HTML document, which might include text,



graphics, sound files, and hypertext links. Each HTML document is considered as a single Webpage, regardless of the length of the document or the amount of information included [Sta96]. HTML is a text document format, somewhat like word processing or desktop publishing formats, but considerably less complicated and based on more open standards [NaSt00].

HTML was originated by Tim Berners-Lee, with revisions and editing by Dan Connolly and Karen Muldrow. Up until the time when HTML Working Group (i.e., a Group chartered in 1994 with the task of defining the HTML standard) took over responsibility for the standard, it was largely an informal effort. Tim Berners-Lee still very much involved in the evolution of the standard and served as director of the World Wide Web Consortium (W3C), which is group of corporations and other organizations with an interest in the World Wide Web. HTML was subsequently modified to act as a universally understood language; a kind of publishing mother tongue that all computers may potentially understand [Dar99].

### **1.2.1 Markup Languages**

Markup language is a formal set of special characters and related capabilities used to define a specific method for handling the display of files that include markup; HTML, XML and CSS are markup languages [TiBu05]. A markup language is not a compiled language, it is an interpreted language. It is interpreted by the browser when it is first read in, and the results are displayed in the browser window, once that's done, the processing is over until different markup is read in and the cycle repeats. Using Markup within a text file indicates the structure of the document, and indicates which parts of the document are headings, which parts are paragraphs, what belongs in a table, and so on.

There are two kinds of markup languages:

1. **The Control Code Markup:** using control code that characterizes typical word processing and page layout applications in the form of embedded property symbols that are not human readable. Word processing programs use binary codes. If a document is opened with a program that does not understand these codes, a bunch of unreadable characters will be seen.
2. **HTML-Style Markup:** using plain text characters that are both human and machine readable. Hypertext markup languages use human-readable codes in plain text, even though the codes are still meant to be computer processed [Cin99].

### 1.2.2 HTML Obstructs

It is difficult to say that there are disadvantages in having a Web site, since most people and companies use their own Websites to enhance their marketing and customer service efforts, not supplant them. There are a few hurdles to leap, and they should definitely be considered [NaSt00]:

- **Security:** Transmitting data via Internet technology, including the Web, is inherently a rather unsecured process. For data to be transmitted over the Web it has to pass through a number of different servers and hosts; and any of the offered information could potentially be read or held by anyone. This has been a strong argument against commerce on the Web, because people recognize the dangers in revealing personal information (for instance; credit card numbers).
- **Copyright Issues:** The lack of security holds true for the Web designer. Nearly anything could be created on the Web can easily be read or copied by anyone has Web access. This is intimidating both to artists and publishers who want to make sure that Internet access doesn't, in some way, devalue their published (and profitable) efforts.

## **1.3 Web Security Aspects**

The World Wide Web (WWW) has both promises and dangers. The promise is that the Web can dramatically lower costs to organizations for distributing information, products, and services. The danger is that the computers linked to the Web are vulnerable.

**Web security** is a set of procedures, practices, and technologies used for assuring reliable and predictable operations of Web servers, Web browsers, other programs that communicate with Web servers, and the surrounding Internet infrastructure. Unfortunately, the sheer scale and complexity of the Web makes the problem of Web security dramatically more complex than the problem of Internet security in general [Gar01].

### **1.3.1 Web Security Breach**

A breach of security can be defined as illegal access to information that can result in disclosure and/or alteration of information. A Web security breach can take place in several forms, such as infringement on the network of an organization, alteration of organizational or individual information, virus attacks, and theft. A failure in network security could cost the organization some of its goodwill and reputation. No other organization would be interested in doing business with an organization that cannot protect its information and security system [Bha03].

### **1.3.2 Securing Information in Transit**

Much of the initial emphasis in the field of Web security is involved with the problem of protecting information as it traveled over the Internet from a Web server to the end user's computer. The concern was that someone eavesdropping on the network (at intermediate nodes) might copy sensitive information, or alter information in transit.

There are many ways to protect information from eavesdropping as it travels through a network; the most common ways are [Gar01]:

- Physically secure the network, so that eavesdropping becomes nearly impossible task.
- Hiding the information that needed to be secured within other information that appears innocuous.
- Encrypt the information so that it cannot be decoded by any party who is not in possession of the proper key.

Of these techniques, encryption is the only technique that is practical on a large-scale public network. Physically securing the Internet is impossible. Information hiding only works if the people who the information are hidden from do not know it is hidden. Additionally, encryption can prevent outside alteration, or make it obvious when the information has been changed.

## **1.4 Cryptography and Web Security**

Cryptography is the fundamental technology used to protect information as it travels over the Internet. Every day, encryption is used to protect the content of Web transactions, email, newsgroups, chat, Web conferencing, and telephone calls as they are sent over the Internet. Without encryption any crook, thief, Internet service provider, telephone company, hostile corporation, or government employee who has physical access to the network media (that carry confidential data) can eavesdrop upon the exchanged data. Cryptography can be used for more than scrambling messages. Increasingly, systems that employ cryptographic techniques are used to control access to computer systems and to sign digital messages. Cryptographic systems have also been devised to allow the anonymous exchange of digital money and even to facilitate fair online

voting. For all of its users, cryptography is a way of ensuring certainty and reducing risk in an uncertain world [Gar01].

## 1.5 Literature Review

A few published researches were found, relatively, close to the work presented in this research. But in the world, there are many companies adopted this field of work .The following are among them:

1. **Ants Soft, (2000), [Ant00]**, this software company had released a software tool "**HTML Protector**" which had offered a protection to Webpage content by preventing unauthorized users from viewing HTML source code. This software offered a three JavaScript based encryption methods to be chosen from. A variety of other features (like right click disable, printing disable and password protection) were offered.
2. **Protware, (2003) [Pro03]**, this software company had released "**HTML Guardian**", which is a solution for total Website protection. It prevents stealing and reuse of some codes published in websites. "HTML Guardian" had offered a design to protect images on the website. Additional features were offered (like disabling right click, disabling page printing and disabling text selection).
3. **Ozcan, (2003), [Ozc03]**, in his master thesis (titled "*Design and development of practical and secure E-mail system*") presented an E-mail system that performs public key distribution and management in a unique way. The system was named "*Practical and Secure E-mail System*" (*PractiSES*). A central authority, which is trusted by all users, takes the responsibility of key distribution and management in PractiSES. The PractiSES Client module is an E-mail application

that is designed for end users. On top of regular E-mail client features, PractiSES Client can also be used to exchange E-mails among users in encrypted and/or signed fashion. PractiSES is designed according to the phases of "*Object Oriented Analyses and Design (OOAD)*".

4. Alwash, (2003), [Alw03], in his master thesis (titled "*Design and implementation of an E-mail security system*"), he presented a secure mail system called (*SecureMail*) that provided the Email sender with a confidentiality property. This protection was done using *Rijndael* encryption algorithm. Also, *Frog2prince* algorithm was applied as a linguistic steganography on the encrypted message, such that the cipher text was hidden in an innocuous English text.
5. Al Moosaway, (2005), [AIM05], in her master thesis (titled "*Encryption and compression of HTML files*") suggested the use of Elgamal public-key encryption algorithm to encode the Webpage's documents. The resulted files are further processed using some compression algorithms (such as LZ77, LZP). These compression algorithms require a dictionary shared by the compressor and the decompressor. The basic idea of these algorithms is to define a unique encryption for each word in the dictionary. After the implementation of the above two compression algorithms, the resulted files were processed using some ready made compression applications (such as WINRAR and WINZIP).
6. Abass, (2006), [Aba06], in his master thesis (titled "*An encryption based security system for E-mail*"), used the Blowfish E-mail system that uses Blowfish algorithm, This system is made to encrypt the text and send it as E-mail encrypted text. The system decrypt E-mail

encrypted text using a key called "the usable key". The applied algorithm uses a single secret key to perform both encryption and decryption tasks. Also, the system can encrypt bitmap image files (\*.bmp), and sends it as attached file to the recipient who will open the file and decrypt it.

## **1.6 Aim of Thesis**

The aim of this research is to define the way of applying cryptography to prevent illegitimated person from viewing HTML file (Webpage) off line. The legitimated persons can decrypt the encrypted HTML file (Webpage) by their shared secret key. The structure of the encrypted HTML file is compatible with HTML file standard structure and can be viewed by any Web browser but the displayed contents of the Webpage looks unreadable.

## **1.7 Thesis Outline**

The remainder part of this research consists of four chapters, and they are organized as follows:

- ❖ **Chapter 2 (*Theoretical Concepts*):** This chapter introduces the theoretical concepts of HTML, cryptography and base64 encoding.
- ❖ **Chapter 3 (*The Improved HSS-Scheme*):** This chapter describes the structure of the proposed system. The encoding and decoding modules are described in details.
- ❖ **Chapter 4 (*Implementation and Testing*):** This chapter presents the interfaces of the suggested system and shows the results of the tests to evaluate the performance of the proposed system using different HTML files.

- ❖ **Chapter 5 (*Conclusions and Future Work*):** This chapter introduces the derived conclusions, and some suggestions for future works are given.



# *Chapter Two*

## *Theoretical Concepts*

### **2.1 Introduction**

Over the last past years, Internet has grown from being a small research-based conglomeration of computers and networks to a major communication environment. Before 20 years, it was rare for a company to have a Webpage, companies now have multiple pages. Not only those do major corporations have WebPages, but mostly every organization, club, group, and individual has a page as well. Unfortunately, with this increased Web activity, comes the usual growing pain; which is due to the availability of private or confidential information, computer espionage, and malicious users wanting to break into the various Web sites for one reason or another [Fis00].

This chapter reviews three topics. The first one presents a general introduction to HTML language, some related concepts about HTTP (Hyper Text Transfer Protocol) and browsers. The second topic is the cryptographic principles and algorithms for security. The last topic deals with the conceptual meaning of Base64 encoding.

### **2.2 Beneath the World Wide Web**

The WWW is a web of pages hosted on Web servers around the world, connected by hyperlinks that connect one page to another. Those connections rely on a common set of protocols that allow any Web server machine to be configured in a certain way to distribute documents across the Internet in a standard way [NaSt00].

The WWW uses three technologies [Bro98]:

1. A markup language to handle the display of Web pages like HTML (Hyper Text Markup Language).
2. A protocol to transmit those pages like HTTP (Hyper Text Transfer Protocol).
3. A Web client program (i.e., Web browser) to receive the sent data from server, interprets it, and display the results like Internet Explorer.

The following sections will explain each of them.

### **2.2.1 Hyper Text Markup Language (HTML)**

HTML stands for Hyper Text Markup Language. This Markup language was invented by Tim Berners-Lee as a mean of distributing nonlinear text, called Hypertext, to multiple points across the internet. So, HTML is a cross-platform language that works on any type of computer, anywhere in the world. HTML provides a rich lexicon and syntax for designing and creating useful hypertext documents for the Web [NaTa98, [TiBu05].

#### **A. HTML Structure [TiBu05]**

HTML is a straightforward language for describing Webpage contents. HTML is not a programming language; it is a static descriptive language. Markup language is a set of markup tags. The markup tags describe how text should be displayed.

HTML markup has three types of components:

1. **Elements:** Identify different parts of an HTML page using tags.
2. **Attributes:** Information about an instance of an element.
3. **Entities:** Identify non-ASCII text characters, tag characters.

Every HTML markup that describes a Web page's content includes some combination of elements, attributes, and entities.

## 1. Elements [TiBu05]

Elements are the building blocks of HTML. They are used to describe every piece of text on the page. Elements are made up of tags and the content within those tags. A tag is the formal name for a piece of HTML markup that signals a command, usually enclosed in angle brackets `<>`. HTML tags are not case sensitive.

There are two main types of elements:

1. **Container elements:** Elements with content, they made up of a tag pair and the content that sits between the opening and closing tag in the pair. Start and end tag pairs look as seen below:

`<tag> content </tag>`

Content such as *paragraphs*, *headings*, *tables*, and *lists* always uses a tag pair.

2. **Empty elements:** Elements that insert something into the page using a single tag as seen below:

`<tag>`

## 2. Attributes [TiBu05]

Attributes allow an element to describe contents. Attributes let HTML designers to use elements differently depending on the circumstances. They are included within the start tag of the element and after the element name but before the ending sign, as seen below:

`<tag attribute="value" attribute="value">`

Every HTML element has a collection of attributes that can be used with it; attributes and elements cannot be mixed or matched. Some

attributes can take any text as a value while others have a specific list of values (such as options for coloring texts in the body section).

### **3. Character Entities [NaTa98]**

Character entities are numeric representations of characters to display the desired character on the page; regardless of document encoding. The text part of American Standard Code for Information Interchange (ASCII) defines a fairly small number of characters. It does not include some special characters, such as *trademark symbols*, *fractions*, and *accented characters*. HTML uses entities to represent these Non-text characters. The browser replaces the entity with the character it references. Each entity begins with an ampersand (&) and ends with a semicolon (;)

Character entities can be represented in three ways:

1. **Numerical entities that use decimal values:** To represent entity (symbol) using decimal values, the following syntax “&#Unicodevalue;” is used, where Unicode value is a unique number assigned to this symbol. The example below shows the decimal numerical entity for the division sign.

*<Tag> I often use the &#247; sign</Tag>*

2. **Numerical entities that use hexadecimal values:** To represent entities using hex values, the following syntax “&#xHexadecimalvalue;” is used, where the actual hexadecimal value of the character replaces *Hexadecimalvalue*. The following example shows the hexadecimal representation of the division sign entity.

*<Tag> I often use the &#xF7; sign </Tag>*

3. **Named entities:** Representing entities using standard HTML list names instead of numerical values. The example below shows the division sign entity but referring to it by its standard HTML list name.

`<Tag>I often use the &divide sign</Tag>`

In addition to non ASCII-characters representations, HTML assumes that some HTML markup characters, (such as: greater-than and less-than signs) are meant to be hidden and not displayed on the Webpage, this is because these symbols are instructions and will not show up on the Webpage. If these symbols on the Webpage were needed, the entities for them in the markup are included, as shown in the following example [TiBu05]:

`<p>The paragraph element identifies some text as a paragraph :</p>`

`<p>&lt; p &gt; This is a paragraph. &lt; /p &gt; </p>`

Where (&lt;) means (<) and (&gt;) means (>).

The following five items should always be replaced with their respective entities because most of them are used in markup tags [TiBu05]:

1. Quotation marks (“and”).
2. The less than and greater than signs (< and >).
3. The ampersand (&).
4. Any characters not commonly found in English.
5. Mathematical symbols.

HTML usually consumes multiple white spaces into a single space, so the special escape sequence “&nbsp;” is used in case a real space is desired in HTML text by the designer [WiBa98].

Table (B.1) in appendix (B) shows all common character entities.

## B. HTML Document Parts

An HTML document parts must be built on a very specific framework. The entire document is enclosed in the `<HTML></HTML>` container tags. The first part of the document is encapsulated in the `<HEAD></HEAD>` container, which itself contains a `<TITLE></TITLE>` container. Finally, the body of the Webpage is contained in a `<BODY></BODY>` container [Bro98]. The simplest possible HTML document is given in Figure (2.1).

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
<HEAD>
<TITLE>A Very Basic HTML Document</TITLE>
</HEAD>
<BODY>
This is where the text of the document would be.
</BODY>
</HTML>
```

Figure (2.1) The generic structure of HTML document

In the above list of tags, the following types are shown:

### (1) **DOCTYPE** Tag [NaTa98]

The DOCTYPE tag always appears in the first line of HTML documents. It takes the general form of:

```
<!DOCTYPE HTML PUBLIC "public identifier">
```

Where *public identifier* defines the HTML version employed and the language used, as shown in Figure (2.1).

**(2) HTML Tag [NaTa98]**

The most fundamental tag used to create an HTML document is the `<HTML>` tag. This tag should be the first item in the document (after *DOCTYPE* tag), and the corresponding end tag (i.e., `</HTML>`) should be the last. Together, these tags indicate that the material contained between them represents a single HTML document. This is important because an HTML document is a plain text ASCII file, without these tags, a browser or other program will not be able to identify the document format and interpret it correctly.

**(3) HEAD Tag**

The head element is opened by the start tag `<HEAD>`, and closed by the end tag `</HEAD>`. This tag should follow the `<HTML>` start tag. The purpose of the head element is to provide information to the application that is interpreting the document. With the exception of the *TITLE* element, the elements within the head element are not seen by the reader of the document.

**(4) TITLE Tag**

A *TITLE* tag inside the *HEAD* element, `<TITLE>`, is a container for the text describing the document, so a closing tag is required. The text indicated by the *TITLE* tag will be displayed in the title bar of the Web browser.

**(5) BODY Tag**

The `<BODY>` container holds all of the information actually seen in the main browser window. Because HTML is a markup language, the body of the document is turned on with the start tag `<BODY>`. Everything that follows this tag is interpreted according to a strict set of rules that tell the

browser about the contents. The body element is closed with the end tag `</BODY>`.

### C. Block-Level and Inline Elements

Two different main groups of elements can be used within the BODY section of an HTML document: block-level elements and inline elements. The difference between the two groups is that when rendered visually in a web browser, the content of block-level elements starts on a new line, while the content of inline elements does not. The block-level elements may contain inline elements while inline elements may only contain text and other inline elements. Some examples for block-level elements are the elements used for headings (*H*), paragraphs (*P*), forms (*FORM*), divisions (*DIV*), tables (*TABLE*) and lists (*OL/UL/DL*).

Inline elements are those which are used to format text differently from the surrounding text inside the same parent element, such as the elements used for bold text (*B*), italic (*I*), underlined text (*U*), subscript (*SUB*), superscript (*SUP*) and many more. They can be nested, which means putting together one inside another; for example, `<B><I>` is bold italic `</I></B>` or `<I><B>` and so is `</B></I>`. Table (2.1) is a list of text formatting elements (block-level elements), and Table (2.2) is a list of simplest HTML formatting tags (inline elements) [TiBu05].

Table 2.1 Text formatting elements [Cin99]

Tag	Function	Attributes
BR	Specifies a line break.	CLEAR="none,left,right,all"
H1	Bold, very large font, centered. One or two blank lines above and below.	ALIGN="left,center,right,justify"
H2	Bold, large font, flush left. One or two blank lines above and below.	ALIGN="left,center,right,justify"



Tag	Function	Attributes
H3	Italic, large font slightly indented from the left margin. One or two blank lines above and below.	ALIGN="left,center,right,justify"
H4	Bold, normal font, indented more than H3. One blank line above and below.	ALIGN="left,center,right,justify"
H5	Italic, normal font, indented as H4. One blank line above.	ALIGN="left,center,right,justify"
H6	Bold, indented same as normal text, more than H5. One blank line above.	ALIGN="left,center,right,justify"
HR	A divider between sections of text; typically a full width horizontal rule.	ALIGN="left,center,right,justify" SIZE=(number of point rule) WIDTH=(percentage of screen size) NOSHADE
P	Indicates a new paragraph. This will cause a line break before with extra vertical space before. Whether or not there is space after the paragraph depends on how it ends, with a closing tag or a different type of text block.	ALIGN="left,center,right,justify"

Table (2.2) HTML simple formatting elements [Cin99]

Tag	Function	Example
B	Display text in bold.	<B>Buy now!</B>
I	Display text in italics.	<I>This is important.</I>
U	Display text underlined.	<U>Don't forget.</U>
S	Display text with strikethrough.	<S>No longer available.</S>
TT	Display text in monospace.	<TT>x = c*t</TT>

## D. BODY Element Attributes

The BODY element supports a large number of attributes. They are important for determining the general appearance of the document. Table (2.3) lists these attributes and their functions [Bro98].

Table (2.3) Body attributes and their functions [Bro98]

Attribute	Function
ALINK	Defines the color of an active link.
BACKGROUND	Points to the URL of an image to use for the document background.
BGCOLOR	Defines the color of the document background.
BGPROPERTIES	If this is set to FIXED, the background image does not scroll.
LEFTMARGIN	Sets the width of the left margin in pixels.
LINK	Defines the color of an unvisited link.
TEXT	Defines the color of the text.
TOPMARGIN	Sets the width of the top margin in pixels.
VLINK	Defines the color of an already visited link.

The following example shows the way of using BODY attributes:

```
<BODY bgcolor="#FFFFFF" text="#000000" link="#0000FF">
```

## E. HTML Colors

The first small step toward creating a document is to define the colors that will be used for any text in the Webpage, as well as to define a background color for the entire Webpage or some portion of it. If colors are not specified, the default colors are used [Bro98].

## F. Color Values

In HTML, color values are defined in two ways [TiBu05]:

## 1. Color Names

The HTML specification includes 16 color names that can be used to define colors in WebPages. Table (2.4) lists the 16 standard color names with their hexadecimal values.

## 2. Color Numbers

Color numbers allow using any color on WebPages by using Hexadecimal notation. Colors are defined in HTML using a hexadecimal coding system (the base 16 numbering system). The system is based upon the three components (Red, Green, and Blue) which lead to the common abbreviation of RGB. Each of the components is assigned a hexadecimal value between 00 and FF (0 and 255 in decimal numbers). These three values are then concatenated into a single value that is preceded by a pound sign (#) [Bro98].

Table (2.4) Sixteen common color values [Bro98]

Color Name	Hexadecimal Value	Color Name	Hexadecimal Value
Black	#000000	Green	#008000
Silver	#C0C0C0	Lime	#00FF00
Gray	#808080	Olive	#808000
White	#FFFFFF	Yellow	#FFFF00
Maroon	#800000	Navy	#000080
Red	#FF0000	Blue	#0000FF
Purple	#800080	Teal	#008080
Fuchsia	#FF00FF	Aqua	#00FFFF

When defining colors for HTML document elements color names or their values can be used. For example, the following lines are equivalent:

```
<BODY BGCOLOR="#FFFFFF">
```

```
<BODY BGCOLOR="WHITE">
```

## G. FONT Element

The method that HTML uses for providing control over the appearance of the text is the FONT element. The FONT element is a container that is opened with the `<FONT>` start tag and closed with the `</FONT>` end tag. The FONT container surrounds the text that a font is wanted to assign to. Table (2.5) shows how each of the attributes should be used. The browser needs to have the specified font to be installed on the computer. If the browser does not have the correct font, it will use the default font as set in its preferences. Figure (2.2) shows an example of how a font attributes (*FACE*, *SIZE*, and *COLOR*) is specified and Figure (2.3) shows the Webpage displayed in Microsoft Internet Explorer [Bro98].

Table (2.5) The list of attributes of the font tag [NaTa98]

Attribute	How It's Used	Sample Syntax
SIZE	This attribute can specify absolute sizes from 1 to 7, or relative sizes such as +3. Relative sizes which are larger than the defaults are recommended. That is, use sizes which begin with a + (plus) sign.	<pre>&lt;FONT SIZE="+2"&gt;Web design is fun!&lt;/FONT&gt; &lt;FONT SIZE="3"&gt;Web design is fun!&lt;/FONT&gt; &lt;FONT SIZE="2"&gt;Web design is fun!&lt;/FONT&gt;</pre>
COLOR	This attribute can specify the color by name (such as red), or by hexadecimal value (which gives much more control over the color).	<pre>&lt;FONT COLOR="#000099"&gt;Web design is fun!&lt;/FONT&gt; &lt;FONT COLOR="red"&gt;Web design is fun!&lt;/FONT&gt;</pre>
FACE	This attribute defines the typeface. More than one typeface can be specified and the browser looks for the first one specified. If the user's computer has none of the typefaces specified, the page will be displayed in the default face. Commas separate the names of the fonts.	<pre>&lt;FONT ACE="Arial,Helvetica"&gt;Web design is fun!&lt;/FONT&gt;</pre>

```

<HTML>
<HEAD>
<TITLE>Font face, Size,color Examples</TITLE>
</HEAD>
<BODY>
<FONT FACE= "Arial","Verdana", "Helvetica">
This is an example of. Arial selection </FONT><BR>
<FONT FACE= "Verdana","Arial", "Helvetica">
This is an example of Verdana selection. </FONT><BR>
<FONT FACE= "Helvetica","Verdana","Arial">
This is an example of Helvetica selection. </FONT><BR>
<FONT SIZE=+2>Size 5</FONT><BR>
<FONT SIZE=6>Size 6</FONT><BR>
<FONT COLOR="#FF0000">This text is red</FONT><BR>
</BODY>
</HTML>

```

Figure (2.2) Font attributes (face, size and color)

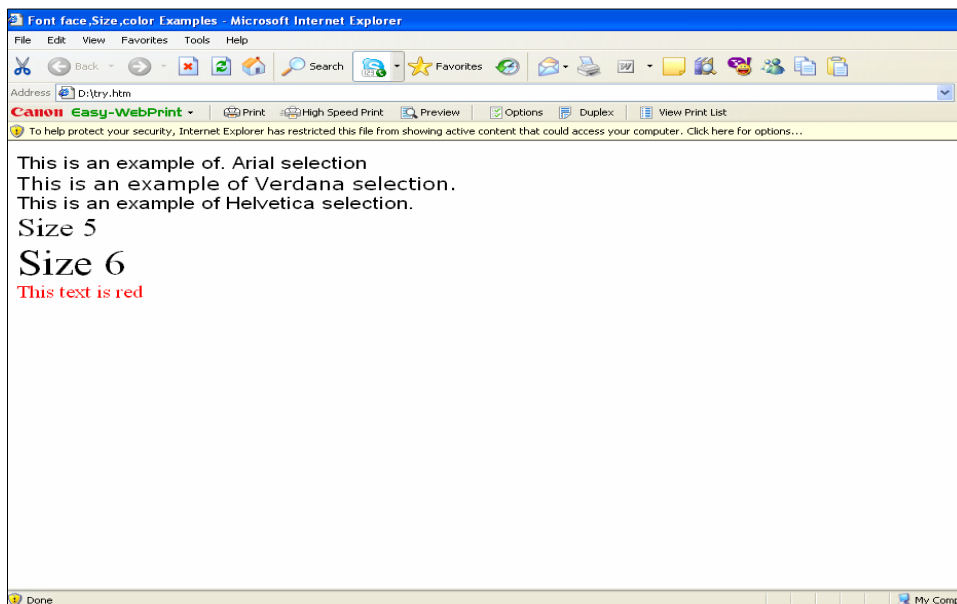


Figure (2.3) Webpage in Microsoft Internet Explorer

### **2.2.2 Browsers**

Web browsers were created specifically for the purpose of reading HTML instructions (known as markup), and displaying the resulting Web page. Web browsers interpret HTML files in their own way; the same HTML may not look exactly same from one browser to other [KeMu06]. Web browsers access HTML documents when entering a URL (Uniform Resource Locator) in the URLs field on the browser, the browser goes through the following three basic steps [NaSt00]:

1. It determines what protocol to use.
2. It looks up and contacts the server that have the specified address.
3. It requests the specific document (including its path statement) from the server computer.

The browser usually figures these previous steps by a combination of the selected protocol and the extension of the filename in question. For instance, a file called `index.html` that is accessed using a URL and started with the `http://` protocol will be displayed in the browser as an HTML file, complete with formatting and hypertext links. However, if that same file is renamed `index.txt`, even if it is loaded with an `http://` protocol URL, it will be displayed in the browser as a simple ASCII file, just as if it were being displayed in notepad, this is because the extension tells the web browser how to display the file.

Browsers and HTTP (Hyper Text Transfer Protocol) servers need not be part of the Web to function. There is no need to be connected to the Internet or to any network to write HTML documents and operate a browser. Locally stored documents and accessory files can be loaded and displayed directly on the browser. Many organizations take the advantage of this capability by distributing catalogs and product manuals, for instance, on a much less expensive, but much more interactively useful, CD-ROM,

rather than via traditional print on paper. Organizations can be connected to the Internet but also maintain private web sites and document collections for distribution to clients on their local networks. In fact, private web sites are fast becoming the technology of choice for the paperless offices during the last years. With HTML document collections, businesses can maintain personnel databases complete with employee photographs and online handbooks, collections of blueprints, parts, assembly manuals, and so on all readily and easily accessed electronically by authorized users and displayed on a local computer.

### **2.2.3 Types of Browsers**

The Web world is full of browsers of many versions and feature sets. Two of the more popular browsers are Microsoft Internet Explorer and Netscape Navigator. Web browsers are categorized by *platform* (i.e., the operating system being used on the computer in question, for example: Windows, Mac, or UNIX) and by *version number*. The features of each browser are slightly different within the same version number across platforms and are often significantly different across versions. Differences may be as minor as which font is the default or as major as whether the browser supports Java or not [TiBu05].

### **2.2.4 Hyper Text Transfer Protocol (HTTP)**

Interaction between browsers and servers are made possible by a set of computer-communication instructions called Hypertext Transfer Protocol (HTTP). This protocol defines how browsers should request WebPages and how Web servers should respond to those requests. Web browsers and Web servers do all of the HTTP work for the user, so the user only have to put his Webpages on a server and/or type the Web address into a browser.

There are not many security protocols specifically designed for individual applications. However, because the WWW has become one of the fastest growing applications in the Internet, a specific security protocols were designed to be used for secure Web transactions, like; Secure Hyper Text Transport Protocol (SHTTP) and HTTP over SSL (HTTPS) [Cis89].

### **A. Secure Hyper Text Transfer Protocol (SHTTP)**

SHTTP is a secure message-oriented communication protocol designed to be used for securing messages using the HTTP protocol. The protocol preserves the characteristics of HTTP while allowing request and reply messages to be signed, authenticated, encrypted, or any combination of these. Multiple key-management mechanisms are supported, including password-style manually shared secrets and public-key exchange. prearranged symmetric session keys are used to send confidential messages. Secure HTTP can verify message integrity and sender authenticity for a message using the message authentication code (MAC). The MAC is computed as a keyed hash over the document using a shared secret. No secure pipe is established between the parties [Cis89].

### **B. Hyper Text Transfer Protocol over SSL (HTTPS)**

Secure Sockets Layer (SSL) was developed by Netscape to provide security when transmitting information on the Internet. Netscape recognized the need to develop a process that would ensure confidentiality when entering and transmitting information on the Web. SSL utilizes both asymmetric and symmetric key encryption to setup and transfer data in a secure mode over an unsecured network. When it is used with a browser client, SSL establishes a secure connection between the client browser and the server. It sets up an encrypted tunnel between a browser and a Web



server over which data packets can travel. The common differences between SHTTP and HTTPS are:

- HTTPS is connection-oriented and operates at the transport level. It creates a secure connection over which transactions are transmitted.
- SHTTP is transaction-oriented and operates at the application level.
- HTTPS enjoys wide acceptance, while SHTTP's use is very limited (specifically designed for HTTP and not for other protocols). Not all Web browsers support SHTTP [Can01].

## **2.3 Web Security**

Web is an ocean of information where individuals and organizations are connected to each other through different networks. Therefore, whether it is an individual or an organization, security on the Web mostly implies security of information. Web security is essential in order to implement the following requirements [Bha03]:

**1. Confidentiality:** It refers to securing critical information from disclosure to unauthorized users. The extent to which confidentiality should be maintained depends upon the type of information to be secured. The type of information an organization shares with its employees, and conversely with people outside the organization, is an example that illustrates the difference in levels of confidentiality maintained. An organization would share all policies and procedures with its employees. In contrast, it would not like to share such information with outsiders. Similarly, information about the appraisal of employees would only be accessible to supervisors and the human resources (HR) department. Such information would not be accessible to all employees of the company. There were many solutions to protect (or conceal) the confidential information sent over network from eavesdroppers, but the most practical way is encryption.

**2. Integrity:** It implies ensuring that unauthorized users do not modify information. It is essential to maintain the integrity of information on the Web if users do not want that information to be misinterpreted by the intended audience. Many Methods were used to address these unauthorized alteration or modification of data; this is done with digitally signed message digest codes. Data modification includes things like insertion, deletion, and substitution [Gar01].

**3. Availability:** Ensuring the availability of data or information; this implies that the data or information is available for use whenever the need arises.

**4. Authentication:** It is a mechanism for ensuring that an individual who is trying to access a resource on the Web is “who actually claims to be”. Also, it implies ensuring that only authorized individuals are permitted to access information.

### **2.3.1 Cryptography**

*Cryptography* is the science of using mathematics to encrypt and decrypt data. Cryptography enables users to store sensitive information or transmit it across insecure networks (like the Internet) so that it cannot be read by anyone except the intended recipient. While cryptography is the science of securing data, *cryptanalysis* is the science of analyzing and breaking secure communication. Cryptanalysts are also called *attackers*. *Cryptology* embraces both cryptography and cryptanalysis [Net90].

### **2.3.2 Messages and Encryption**

The information that the sender wants to send to the receiver, which it is called “plaintext” can be English text, numerical data or stream of bits.

The sender encrypts the plaintext, using a predetermined key, and sends the resulting ciphertext over the channel. The attacker, upon seeing the ciphertext in the channel by eavesdropping, cannot determine what the plaintext was; but the receiver, who knows the encryption key, can decrypt the ciphertext and reconstruct the plaintext.

This concept is described more formally using the following mathematical notation: A cryptosystem is a five-tuple  $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$  system, where the following conditions are satisfied:

1.  $\mathcal{P}$  is a finite set of possible plaintexts.
2.  $\mathcal{C}$  is a finite set of possible ciphertexts.
3.  $\mathcal{K}$ , the key space, is a finite set of possible keys.
4. For each  $(K \in \mathcal{K})$ , there is an encryption rule  $(E_K \in \mathcal{E})$  and a corresponding decryption rule  $(D_K \in \mathcal{D})$ . Each  $(E_K: \mathcal{P} \rightarrow \mathcal{C})$  and  $(D_K: \mathcal{C} \rightarrow \mathcal{P})$  are functions such that  $(D_K(E_K(x)) = x)$  for every plaintext  $(x \in \mathcal{P})$ . It must be the case that each encryption function  $E_K$  is an injective function (i.e., one-to-one), otherwise, decryption could not be accomplished in an unambiguous manner [Sti95]. Figure (2.4) illustrates the encryption and decryption processes.

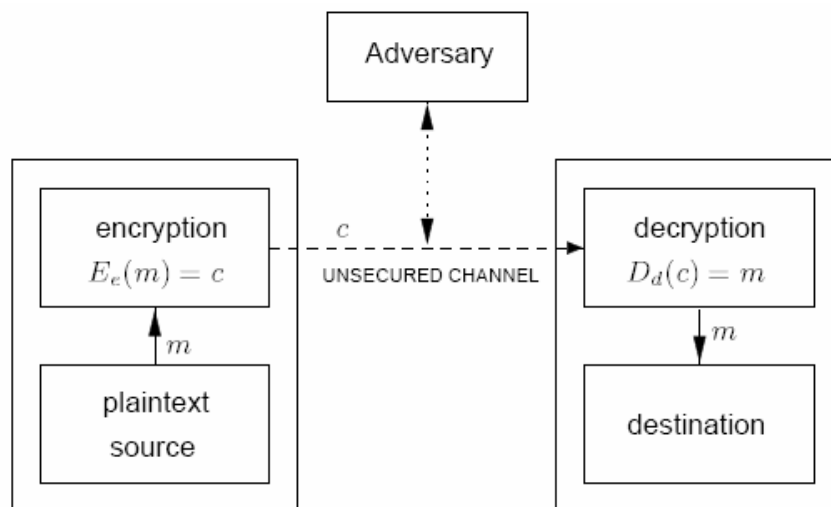


Figure (2.4) Encryption and decryption [Men96]

### 2.3.3 Cryptographic Protocols

A protocol is a series of steps, involving two or more parties designed to accomplish a task. The parties can be friends trust each other implicitly or they can be adversaries and do not trust one another. A cryptographic protocol involves some cryptographic algorithms [Sch96]. Encryption and decryption algorithms, cryptographic hash functions and pseudorandom generators are the basic building blocks for solving problems involving secrecy, authentication and data integrity. Randomness and the security of cryptographic schemes are closely related. There is no security without randomness. An encryption method provides secrecy only if the ciphertexts appear random to the adversary [Net90].

Cryptographic systems are characterized along three independent dimensions [Sta05]:

1. *The type of operations used for transforming plaintext to ciphertext.*

All encryption algorithms are based on two general principles:

(a) *Substitution*: In which each element in the plaintext (bit, letter, group of bits or letters) is mapped into another element.

(b) *Transposition*: In which elements in the plaintext are rearranged.

The fundamental requirement in both cases is that no information be lost.

2. *The number of keys used*: If both sender and receiver use the same key, the system is referred to as symmetric, single-key, secret-key, or conventional encryption. If the sender and receiver use different keys, the system is referred to as asymmetric, two-key, or public-key encryption.
3. *The way in which the plaintext is processed*: A block cipher processes the input as one block of elements at a time, producing an output block

for each input block. A stream cipher manipulates the input elements continuously, producing output one element at a time.

### 2.3.4 Symmetric Cryptosystems

Any symmetric encryption scheme has five elements as shown in figure (2.5), they are [Sta05]:

1. **Plaintext:** The original intelligible message or data that is fed into the algorithm as input.
2. **Encryption algorithm:** The encryption algorithm performs various substitutions and transformations on the plaintext.
3. **Secret key:** The secret key is also input to the encryption algorithm. The key is a value independent of the plaintext and of the algorithm. The algorithm will produce different output depending on the specific key being used at the time.
4. **Ciphertext:** The scrambled message produced as output. It depends on the plaintext and the secret key. For a given message, two different keys will produce two different ciphertexts. The ciphertext is an apparently random stream of data and is unintelligible.
5. **Decryption algorithm:** The encryption algorithm runs in reverse. It takes the ciphertext and the secret key and produces the original plaintext.

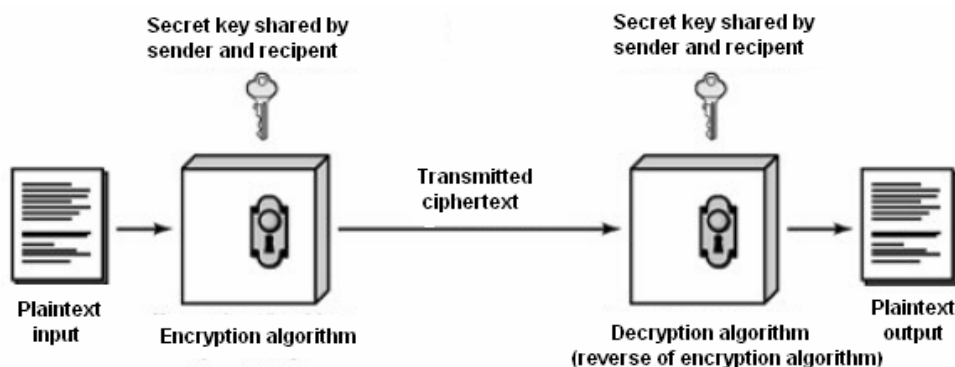


Figure (2.5) Simplified model of symmetric encryption [Sta05]

Challenges with secret key encryption include the following [Cis89]:

1. Changing the secret keys frequently avoiding the risk of compromising the keys.
2. Securely generating the secret keys.
3. Securely distributing the secret keys.

The objective of attacking an encryption system is to recover the key in use rather than recover the plaintext of a single ciphertext. There are two general approaches for attacking a conventional encryption scheme [sta05]:

1. **Cryptanalysis:** Cryptanalytic attacks rely on the nature of the algorithm plus some knowledge of the general characteristics of the plaintext to deduce a specific plaintext or to deduce the key being used.
2. **Brute-force attack:** The attacker tries every possible key on a piece of ciphertext until an intelligible translation into plaintext is obtained.

### 2.3.5 Public-Key Cryptosystems

A public-key encryption (asymmetric algorithm) scheme has triple elements, (i.e.,  $G, E, D$ ); its algorithms satisfy the following conditions [GoBe01]:

1. **Key generation algorithm:** Algorithm  $G$ , which, on input  $k$  (the security parameter) produces a pair  $(e, d)$  where  $e$  is called the public key and  $d$  is the corresponding private key (in a mathematical notation  $(e, d) \in G(k)$ ). The pair  $(e, d)$  refers to the pair of encryption/decryption keys. Asymmetric algorithms rely on one key for encryption and a different but related key for decryption [Sta05].
2. **An encryption algorithm:** Algorithm  $E$ , which takes as input a security parameter  $k$ , a public-key  $e$  from the range of  $G(k)$  and string  $m \in \{0,1\}$  (called the message), and produces as output, a string  $c \in \{0,1\}$  (called the ciphertext). The notation  $(c \in E(k, e, m))$  is used to

denote that the message  $m$  is encrypted using key  $e$  with security parameter  $k$ .

3. **A decryption algorithm:** Algorithm  $D$ , which takes as input, a security parameter  $k$ , a private-key  $d$  from the range of  $G(k)$ , and a ciphertext  $c$  from the range of  $E(k, e, m)$ . As output it produces a string  $m' \in (0, 1)$ , such that for every pair  $(e, d)$  in the range of  $G(k)$ , for every  $m$ , for every  $c \in D(k, e, m)$ , the probability  $(D(k, d, c) \neq m')$  is negligible. Public key encryption scheme is illustrated in Figure (2.6).

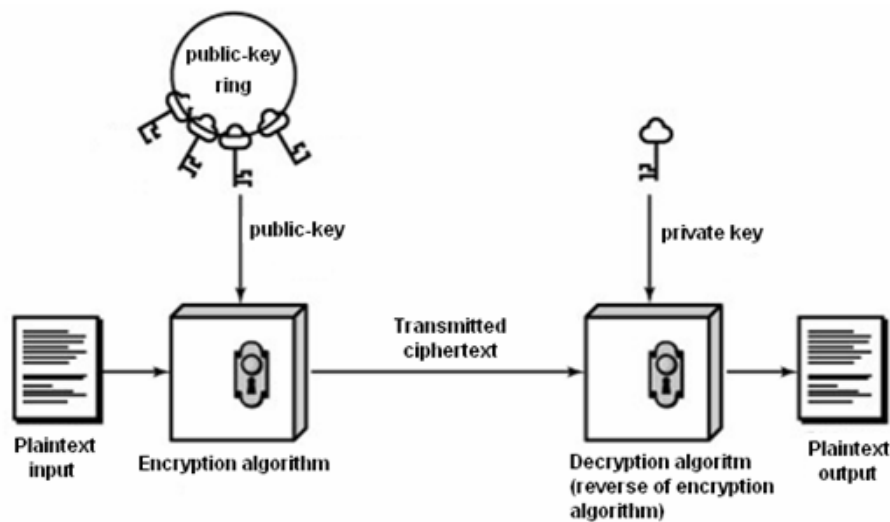


Figure (2.6) Public-key encryption [Sta05]

Among all encryption algorithms, symmetric-key encryption algorithms have the fastest implementations in hardware and software. Therefore, they are very well-suited to the encryption of large amounts of data. Symmetric-key ciphers can be employed as primitives to construct various cryptographic mechanisms including pseudorandom number generators and hash functions. Public key encryption algorithms are rarely used for data confidentiality because of their performance constraints. Instead, public key encryption algorithms are typically used in applications

involving authentication using digital signatures and key management [Cis89, Sch96].

### 2.3.6 Stream Cipher

A stream cipher is still playing an important role in cryptography. Many of the most commonly used symmetric encryption systems are stream ciphers [Sti95]. Stream cipher system converts plaintext to cipher text bit by bit. Figure (2.7) presents a simple diagram of a stream cipher. A keystream generator (sometime called a running-key generator) outputs a stream of bits:  $K_1, K_2, K_3, \dots, K_i$ . This keystream is XORed with a stream of plaintext bits,  $P_1, P_2, P_3, \dots, P_i$  to produce the stream of ciphertext bits:

$$C_i = P_i \oplus K_i \dots \dots \dots (2.1)$$

At the decryption end, the ciphertext bits are XORed with an identical and synchronized keystream to recover the plaintext bits:

$$P_i = C_i \oplus K_i \dots \dots \dots (2.2)$$

Because,

$$P_i \oplus K_i \oplus K_i = P_i \dots \dots \dots (2.3)$$

The system's security depends entirely on the keystream generator. If the keystream generator outputs an endless stream of zeros, then the produced ciphertext is equal to the plaintext and the whole operation becomes worthless. If the keystream generator spits out an endless stream of random bits, a perfect security will be obtained [Sch96].

The stream cipher is similar to the one-time pad; the difference is that a one-time pad uses a genuine random number stream whereas a stream cipher uses a pseudorandom number stream. The important design



considerations should be taken when dealing with a stream cipher system are:

1. The encryption sequence should have a large period. A pseudorandom number generator uses a function that produces a stream of bits that eventually repeated. The longer the period of repeat the more difficult it will be to do cryptanalysis.
2. The key stream should approximate the properties of a true random number stream as close as possible.
3. The output of the pseudorandom number generator is mainly depends on the value of the input key.

Stream ciphers are often breakable if the key stream repeats itself or has redundancy. To be unbreakable; a random sequence must be as long as the plaintext [Dor83]. In stream cipher, the plaintext is modulated by the random key sequence. The XOR operation often acts as a modulation operation. Some random sequence generators have been reported to generate a random key sequence such as linear feedback shift register (LFSR) [Lia09].

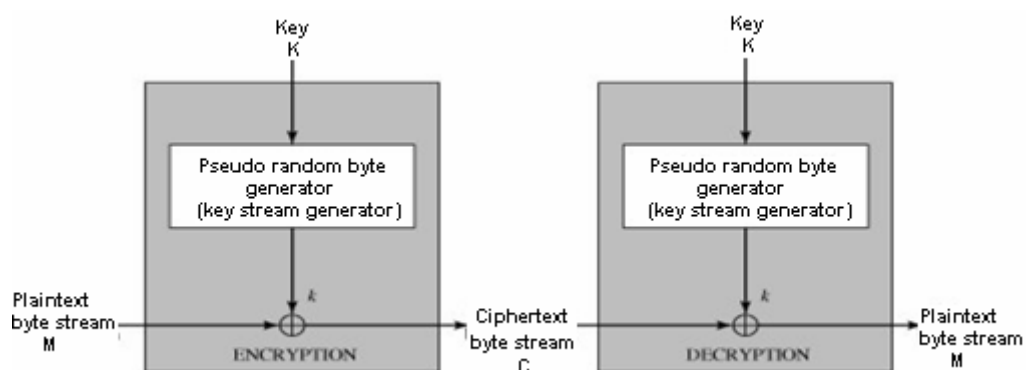


Figure (2.7) Stream cipher diagram [Sta05]

### 2.3.7 Pseudo-Random Number Generator (PRNG)

There is a close relationship between encryption and randomness. The security of encryption algorithms usually depends on the random choice of keys and bit sequences [DeKn07]. However, generating random values on a computer is a very difficult task. One possible method for such generation is to use a pseudo-random numbers generator.

A pseudo-random number generator (PRNG), as shown in Figure (2.8), is a function that once initialized with some random value (called the *seed*) it outputs a sequence that appears random, in the sense that an observer who does not know the value of the seed cannot distinguish the output from that of a (true) random bit generator. PRNG is a deterministic process: if the same state put back, the same sequence will be reproduced. This property makes PRNGs suitable for use as stream ciphers. PRNG structure is composed of a seed repository, an output function producing some random-looking bits from the seed and a feedback function that iteratively transforms the seed. It can be shown that a PRNG is necessarily periodic: the sequence it produces will repeat itself after a (possibly extremely long) period [Til05].

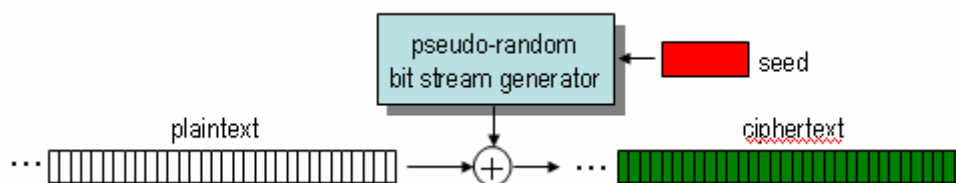


Figure (2.8) PRNG using stream cipher [Til05]

### 2.3.8 Linear Feedback Shift Register (LFSR)

LFSR is one of the appealing aspects of keystream generation in which the keystream can be produced efficiently (but insecure) in hardware [Opp05]. A LFSR, as depicted in Figure (2.9), is composed of a register

which is an array of memory cells, each capable of storing one binary value and a feedback function which consists of the XOR operator applied to a selected cells of the register (taps). For each new unit of time, the following operations are performed:

1. The content of the last memory cell is output.
2. The register is processed through the feedback function, in which the selected memory cells are XORed together to produce one bit of feedback.
3. Each element of the register advances one position, the last element being discarded, and the first one receives the result of the feedback function.

LFSR can produce sequences with large periods and good statistical distribution and theoretically well understood. However, they are not cryptographically highly secured; because efficient techniques are known to reconstruct the content of the register, and hence the full LFSR's output, by observing a short output sequence. Nevertheless, LFSRs can be used as building blocks to construct a secure Pseudo-Random Number Generator PRNG [Til05].

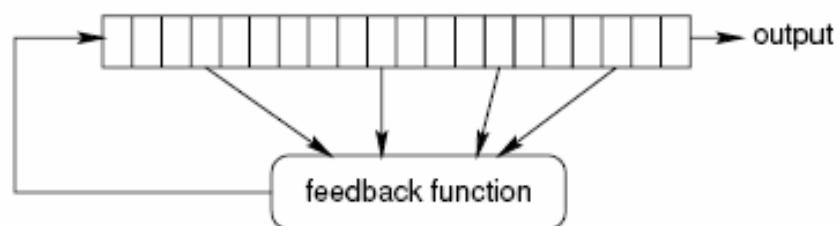


Figure (2.9) Linear feed back shift register [Til05]

### 2.3.9 Randomness Statistical Tests [Men96]

This section presents some of the well-known statistical tests designed to measure the quality of a random bit generator. These tests are

accomplished by taking a sample output sequence of the generator and subjecting it to some statistical tests. Each statistical test determines whether the sequence possesses a certain attribute that a truly random sequence would be likely to exhibit. Passing the tests merely provides probabilistic evidence that the generator produces sequences, which have to some extent certain characteristics of random sequences.

Let  $s = s_0, s_1, s_2, \dots, s_{n-1}$  be a binary sequence of length  $n$ . The statistical tests that are commonly used for determining whether the binary sequence  $\{s\}$  possesses some specific characteristics of randomness are:

### 1. Frequency Test (monobit test)

The purpose of this test is to determine whether the number of 0's and 1's in  $\{s\}$  are approximately same, as would be expected for a random sequence. Let  $n_0, n_1$  denote the number of 0's and 1's in  $s$ , respectively. The used monobit measure is:

$$X_1 = \frac{(n_0 - n_1)^2}{n} \dots\dots\dots (2.4)$$

Which approximately follows a  $\chi^2$  distribution with 1 degree of freedom ( $\nu$ ) if  $n \geq 10$ .

### 2. Serial Test (two-bit test)

The purpose of this test is to determine whether the number of occurrences of  $00$ ,  $01$ ,  $10$ , and  $11$  as subsequences of  $s$  are approximately the same, as would be expected for a random sequence.

Let  $n_0, n_1$  denote the number of 0's and 1's in  $s$ , respectively, and let  $n_{00}, n_{01}, n_{10}, n_{11}$  denote the number of occurrences of  $00$ ,  $01$ ,  $10$ ,  $11$  in  $s$ ,

respectively. It is noted that  $n_{00} + n_{01} + n_{10} + n_{11} = (n - 1)$  since the subsequences are allowed to overlap. The used statistical measure is:

$$X_2 = \frac{4}{n-1}(n_{00}^2 + n_{01}^2 + n_{10}^2 + n_{11}^2) - \frac{2}{n}(n_0^2 + n_1^2) + 1 \dots\dots\dots (2.5)$$

Which approximately follows a  $\chi^2$  distribution with 2 degrees of freedom ( $\nu$ ) if  $n \geq 21$ .

**3. Poker Test**

Let  $m$  be a positive integer such that  $\left\lfloor \frac{n}{m} \right\rfloor \geq 5 \times (2^m)$ , and let  $K = \left\lfloor \frac{n}{m} \right\rfloor$ . The sequence  $\{S\}$  is divided into  $k$  non-overlapping parts each has length  $m$ , and let  $n_i$  be the number of occurrences of one's in the  $i^{th}$  type of sequence of length  $m$ ,  $1 \leq i \leq 2^m$ . The poker test determines whether the sequences of length  $m$  appear, approximately, the same number of times in  $\{S\}$ , as would be expected for a random sequence. The used statistical measure is:

$$X_3 = \frac{2^m}{k} \left( \sum_{i=1}^{2^m} n_i^2 \right) - k \dots\dots\dots (2.6)$$

Which approximately follows a  $\chi^2$  distribution with  $2^m - 1$  degrees of freedom ( $\nu$ ). It is noted that the poker test is a generalization of the frequency test, when setting  $m = 1$  in the poker test yields the frequency test.

**4. Runs test**

The purpose of the runs test is to determine whether the number of runs (of either zeros or ones) of various lengths in the sequence  $\{s\}$  is as expected for a 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 used statistical measure is:

$$X_4 = \sum_{i=1}^k \frac{(B_i - e_i)^2}{e_i} + \sum_{i=1}^k \frac{(G_i - e_i)^2}{e_i} \dots\dots\dots (2.7)$$

Which approximately follows a  $\chi^2$  distribution with  $(2 \times k - 2)$  degrees of freedom ( $\nu$ ).

**5. Autocorrelation test**

The purpose of this test is to check the degree of correlations may found between the parts of sequence  $\{s\}$  and the (non-cyclic) shifted versions of it. Let  $d$  be a fixed integer, such that  $1 \leq d \leq \lfloor n/2 \rfloor$ . The number of bits in  $\{s\}$  not equal to their  $d$ -shifts is  $A(d) = \sum_{i=0}^{n-d-1} s_i \oplus s_{i+d}$  where  $\oplus$  denotes the XOR operator. The statistic used to conduct the autocorrelation test is:

$$X_5 = 2 \left( A(d) - \frac{n-d}{2} \right) / \sqrt{n-d} \dots\dots\dots (2.8)$$

Which approximately follows a standard normal distribution, if  $(n-d \geq 10)$ .

### 2.3.10 The $\chi^2$ Distribution

This distribution can be used to compare the goodness-of-fit of the observed frequencies of events to their expected frequencies under a hypothesized distribution. The chi-square distribution with ( $\nu$ ) degrees of freedom arises in practice when the squares of ( $\nu$ ) independent random variables having standard normal distributions are summed.

Table (C.1) in appendix C presents more details about the  $\chi^2$  distribution.

## 2.4 Cryptographic Hash Function

A hash function is an algorithm that takes a block of input data and generates a shorter fixed length piece of it. This process is done by applying some functions, like that shown in Figure (2.10). The purpose of a hash function is to produce a “*fingerprint*” of a block of data, and it attains a high level of confidence in the integrity of the block of data during transit. A special variant is “*Cryptographic Hash Function*”, which has specific features making it suitable in cryptography [Can01].

A hash function must exhibit the following properties if it is considered cryptographically suitable [Cis89]:

1. It must be consistent; that is, the same input must always create certain output.
2. It must be random ;( or give the appearance of randomness) to prevent guessing the original message.
3. It must be unique; that is, it should be nearly impossible to find two messages have the same digest (stands for digital signature).
4. It must be one way; that is, if the output is given, it must be extremely difficult, if not practically impossible, to ascertain the input message.

To achieve secure transmission, integrity protection should be ensured, and this requirement could fulfilled by a message digest or hash

algorithm, which is applied to produce a hash value that is concatenated with the transmitted message (usually before encryption). If an attacker changes the content of the message during the transmission, the calculated hash value and the transmitted hash value at the receiving end will not match. A widely used message digest or hash algorithm is message digest 5 (MD5), which is adopted in this work.

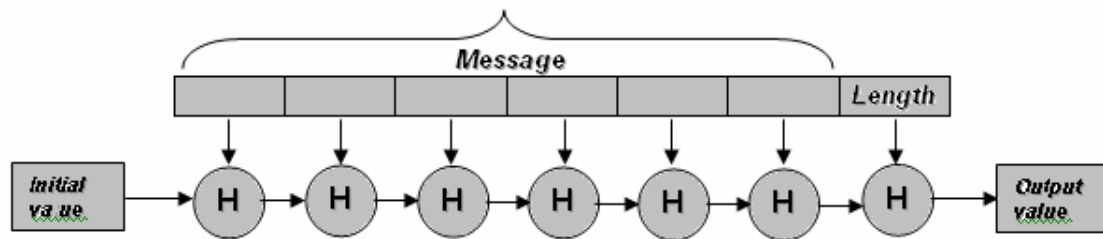


Figure (2.10) Hash function mechanism

## 2.5 Message Digest 5 (MD5) [Ber92]

It is one in a series of message digest algorithms; it was designed by professor Ronald Rivest of MIT. When the analytic work indicated that MD5's predecessor MD4 was likely to be insecure, then MD5 was designed in 1991 to be a secure replacement. MD5 digests have been widely used in the software world to provide some assurance that a transferred file has arrived intact. For example, file servers often provide a pre-computed MD5 checksum for the files, so that a user can compare the checksum of the downloaded file to it.

MD5 algorithm processes a variable-length message into a fixed-length output of 128 bits. As illustrated in Figure (2.11), the input message is broken up into chunks of 512-bit blocks (sixteen 32-bit); the message is padded so that its length becomes divisible by 512. The padding works as follows: first, a single bit, its value is 1, is appended to the end of the message. This is followed by as many zeros as are required to bring the



length of the message up to 64 bits less than a multiple of 512. The remaining bits are filled up with a 64-bit integer representing the length of the original message (in bits).

The main MD5 algorithm operates on a 128-bit state, divided into four 32-bit words, denoted A, B, C and D. These are initialized to certain fixed constants. The main algorithm then operates on each 512-bit message block in turn, each block modifying the state. The processing of a message block consists of four similar stages, termed rounds; each round is composed of 16 similar operations based on: (a non-linear function (F), modular addition, and left rotation).

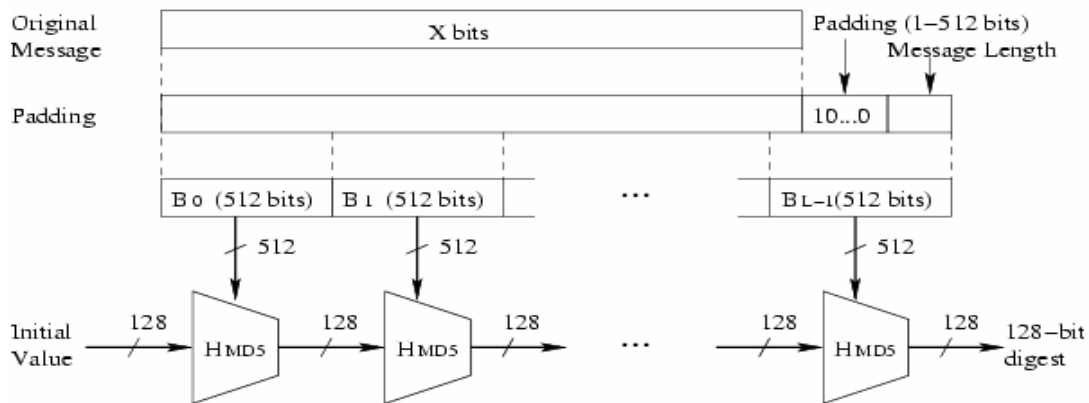


Figure (2.11) Message digest generation using MD5 [Sta05]

Figure (2.12) illustrates one operation within a round. MD5 consists of 64 of these operations (grouped in four rounds of 16 operations). As shown in the Figure, (Mi) denotes a 32-bit block of the message input, and (Ki) denotes a 32-bit constant, different for each operation. The symbol ( $\oplus$ ) denotes addition modulo  $2^{32}$ , and the symbol ( $\ll s$ ) denotes a left bit rotation by (s) places; (s) varies for each operation.

There are four possible functions; a different one is used in each round:

$$F(X, Y, Z) = (X \wedge Y) \vee (\neg X \wedge Z) \dots\dots\dots (2.9)$$

$$G(X, Y, Z) = (X \wedge Z) \vee (Y \wedge \neg Z) \dots\dots\dots (2.10)$$

$$H(X, Y, Z) = X \oplus Y \oplus Z \dots\dots\dots (2.11)$$

$$I(X, Y, Z) = Y \oplus (X \vee \neg Z) \dots\dots\dots (2.12)$$

Where the symbols ( $\oplus, \wedge, \vee, \neg$ ) denote the XOR, AND, OR and NOT operations respectively. The main operations and the four rounds are mentioned in appendix A.

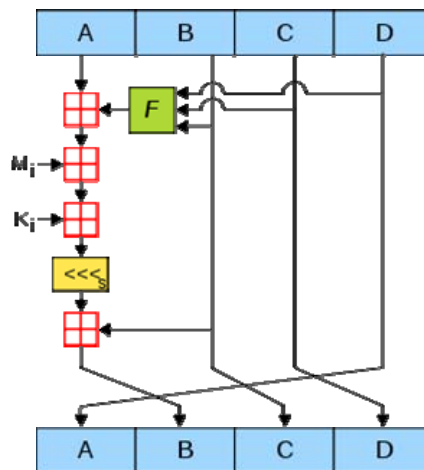


Figure (2.12) One MD5 operation within a round [Ber92]

## 2.6 Encoding

Coding means the replacement of one word with another word or symbol. In contrast, a cipher means a character-for-character or bit-for-bit transformation, without regard to the linguistic structure of the message [Tan03].

In short:

- *Encoding = to convert format, not necessarily securely.*
- *Encryption = encipherment = make secret.*

Encoding covers many different processes, including [Bar05]:

1. Storing textual data encoded in various formats: ASCII, EBCDIC and Unicode.
2. Encoding ciphertext and other binary data that cannot be printed using various formats (like base64, hexadecimal).
3. Encoding the plaintext in a specific format before encrypting it.

## **2.7 Base64 Encoding [BoFr93]**

The base64 encoding scheme was originally devised to make it possible to reliably transmit eight-bit data through transmission systems constrained to handle seven-bit data. The use of base64 encoding makes it possible to:

- Transmit image data reliably across the Internet.
- Transmit non-English characters reliably across the Internet.

The base64 (content-transfer-encoding) is designed to represent arbitrary sequences of octets in a form that need not be humanly readable. The encoding and decoding algorithms are simple, but the encoded data are consistently only about 33 percent larger than the unencoded data. A 65-character subset of US-ASCII is used, enabling 6 bits to be represented per printable character (The extra 65th character, "=", is used to signify a special processing function).

The encoding process as shown in Figure (2.13) represents 24-bit groups of input bits as output strings of 4 encoded characters. Proceeding from left to right, a 24-bit input group is formed by concatenating three 8-bit (byte) input groups. These 24 bits are then treated as 4 concatenated 6-bit groups, each of which is translated into a single digit in the base64 alphabet. When encoding a bit stream via the base64 encoding, the bit stream must be presumed to be ordered with the most significant bit first. That is, the first bit in the stream will be the high-order bit in the first 8-bit

byte, and the eighth bit will be the low-order bit in the first 8-bit byte, and so on. Each 6-bit group is used as an index into an array of 64 printable characters. The character referenced by the index is placed in the output string. These characters, identified in Table (2.6), are selected so as to be universally representable, and the set excludes characters with particular significance (e.g., ".", CR, LF). A full encoding quantum is always completed at the end of a body. When fewer than 24 input bits are available in an input group, zero bits are added (on the right) to form an integral number of 6-bit groups. Padding at the end of the data is performed using the "=" character.

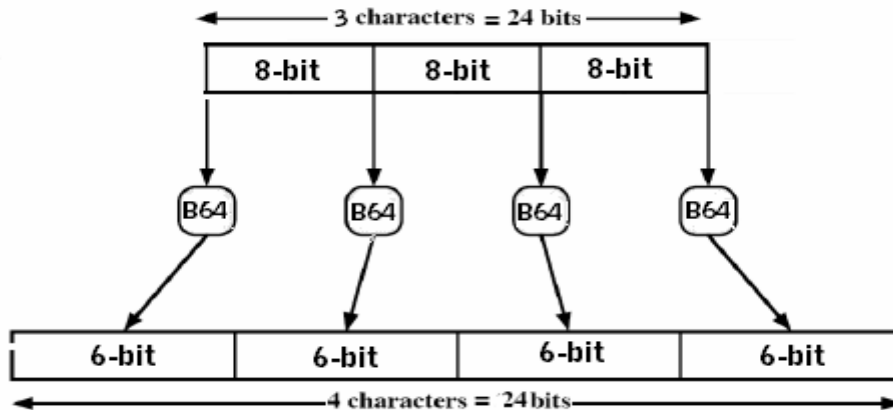


Figure (2.13) Base64 encoding diagram [Mel04]

Table (2.6) Base64 encoding [Mel04]

<b>Value:</b>	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
<b>base64:</b>	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
<b>Value:</b>	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
<b>base64:</b>	Q	R	S	T	U	V	W	X	Y	Z	a	b	c	d	e	f
<b>Value:</b>	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
<b>base64:</b>	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v
<b>Value:</b>	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
<b>base64:</b>	w	x	y	z	0	1	2	3	4	5	6	7	8	9	+	/

Since all base64, input is an integral number of octets; only the following cases can arise:

1. When the final quantum of encoding input is an integral multiple of 24 bits; the final unit of encoded output will be an integral multiple of four characters with no "=" padding.
2. When the final quantum of encoding input is exactly 8 bits; the final unit of encoded output will be two characters followed by two "=" padding characters.
3. When the final quantum of encoding input is exactly 16 bits; the final unit of encoded output will be three characters followed by one "=" padding character.

Because "=" is used only for padding at the end of the data, the occurrence of any "=" characters may be taken as evidence that the end of the data has been reached (without truncation in transit). Any characters outside of the base64 alphabet are to be ignored in base64-encoded data.

In addition to base64 encoding, many types of encoding algorithms exist, like hexadecimal encoding. Hexadecimal encoding uses two symbols (i.e., {0, 1, 2, ..., F}) to encode each byte character. In this encoding scheme, sixteen readable characters could be used to represent all variations in one nibble (i.e., half part of a byte).

Example (2.1) listed below shows the steps required to encode a small binary string into base64 encoding and hexadecimal encoding.

### **Example (2.1)**

#### **a. Base64**

Suppose the binary string is 001100110011000111111000. The 24-bit binary string can be divided into four sets of 6 bits:

001100 110011 000111 111000

If the 6-bit binary numbers are converted to base-10 notation, then:

$$0 \times 32 + 0 \times 16 + 1 \times 8 + 1 \times 4 + 0 \times 2 + 0 \times 1 = 12$$

$$1 \times 32 + 1 \times 16 + 0 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1 = 51$$

$$0 \times 32 + 0 \times 16 + 0 \times 8 + 1 \times 4 + 1 \times 2 + 1 \times 1 = 7$$

$$1 \times 32 + 1 \times 16 + 1 \times 8 + 0 \times 4 + 0 \times 2 + 0 \times 1 = 56$$

looking up 12, 51, 7 and 56 in the base64 Table (2.6), they are 'M' for 12, 'z' for 51, 'H' for 7 and '4' for 56. This binary sequence can be represented as four text characters 'MzH4'. Figure (2.14) shows the applied steps for base64 encoding operation.

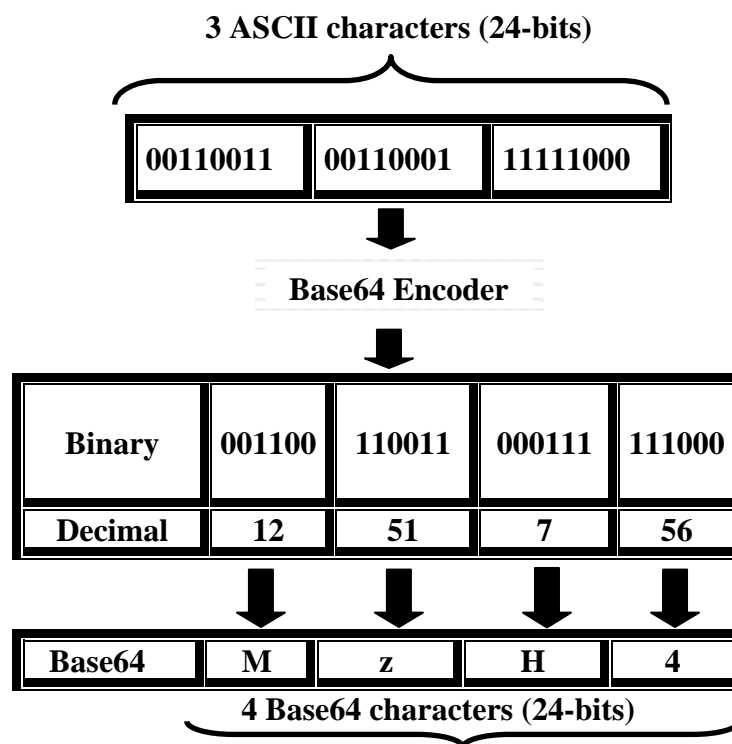


Figure (2.14) Base64 Encoding

### b. Hexadecimal

The same 24-bit binary string can be divided into six sets of 4 bits:

0011 0011 0011 0001 1111 1000

If 4-bit binary number is converted to decimal notation, then

$$0 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1 = 3$$

$$0 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1 = 3$$

$$0 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1 = 3$$

$$0 \times 8 + 0 \times 4 + 0 \times 2 + 1 \times 1 = 1$$

$$1 \times 8 + 1 \times 4 + 1 \times 2 + 1 \times 1 = 15$$

$$1 \times 8 + 0 \times 4 + 0 \times 2 + 0 \times 1 = 8$$

Then, the above binary string is represented as six hex digits '3331F8'. Figure (2.15) depicts the steps performed in hexadecimal encoding operation.

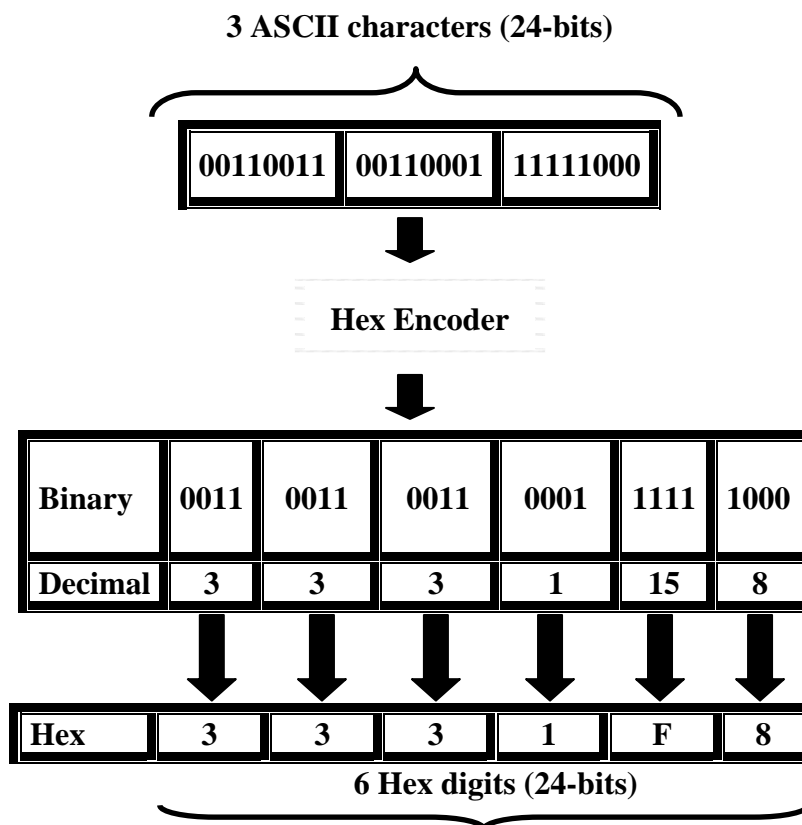


Figure (2.15) Hexadecimal encoding

The results of the above-mentioned example indicate that the 24-bit binary string (consist of 3 characters, each character is represented by 8-bit) can be coded to 4 ASCII characters using base64 encoding and to 6 hex digits using hexadecimal encoding.



# *Chapter Three*

## *The Improved HSS-Scheme*

### **3.1 Introduction**

This chapter is dedicated to present the design considerations, which were taken throughout the construction stages of the proposed HTML security system. The system deals with HTML files, and adds some security aspects on them, to make them more secured in comparison with the conventional HTML files (WebPages). The security demands relevant to the confidentiality of HTML files contents have been given the highest priority.

The system implies the steps of encoding the text messages to produce a sequence of text characters, such that this sequence is not meaningful to an unintended recipient. Beside to confidentiality, the integrity detection aspect of the contents of HTML file had been taken as a major demand, and it was fulfilled by adding digital signature to the content of the file. In addition, the HTML structure condition (i.e., the structure of the HTML file should be preserved) was taken into account.

In our proposed system, some of the theoretical concepts discussed in chapter- 2 were utilized to provide a practical way to secure HTML files. The name of the proposed system is chosen to be HSS (an acronym for **HTML Secure System**). The main target of this system is to encrypt the content of HTML file (i.e., Webpage) including text, color, and font, without changing the structure of HTML file. In addition, the system can decode the secured HTML files and display the contents by any HTML browser.

Two methods were used to generate a sequence of random numbers for encrypting and decrypting HTML file. The first method is Linear Feed Back Shift Register (LFSR). The second method is a proposed method that uses the concept of Pseudo-Random Number Generator (PRNG).

## 3.2 System Model

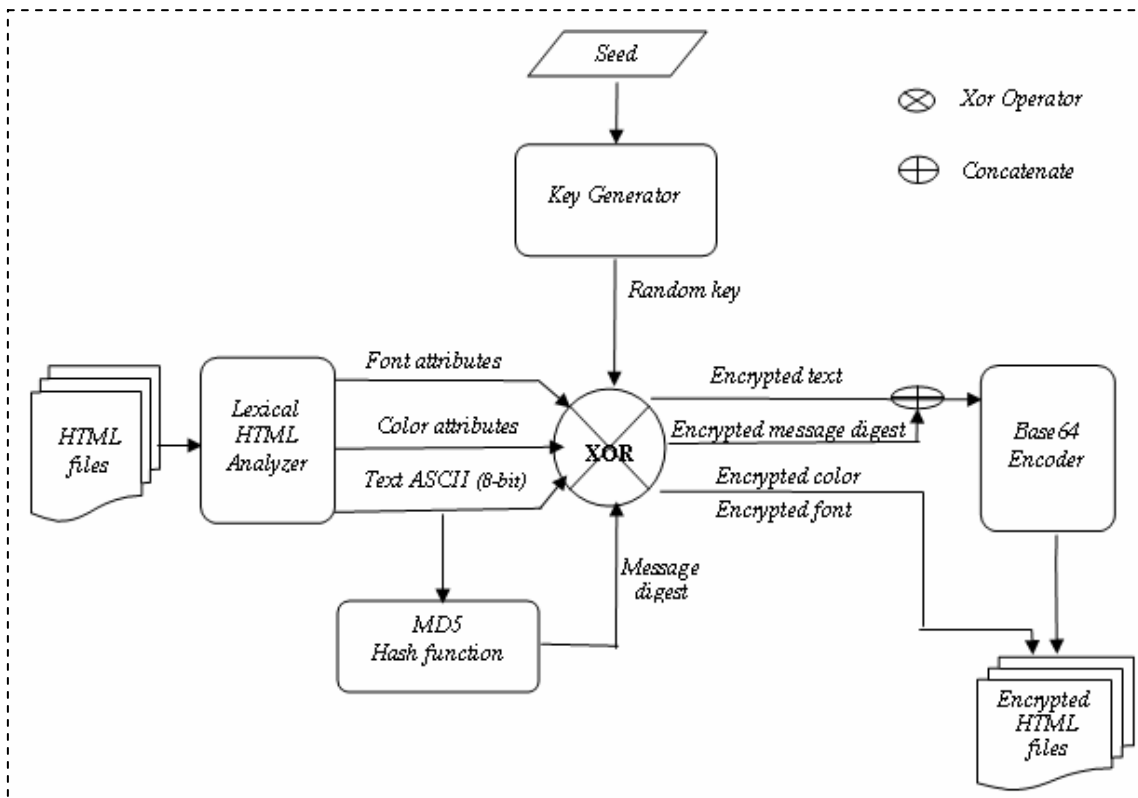
The general structure of the proposed HSS consists of two main modules as illustrated in Figure (3.1). These two modules are:

1. Encoding Module.
2. Decoding Module.

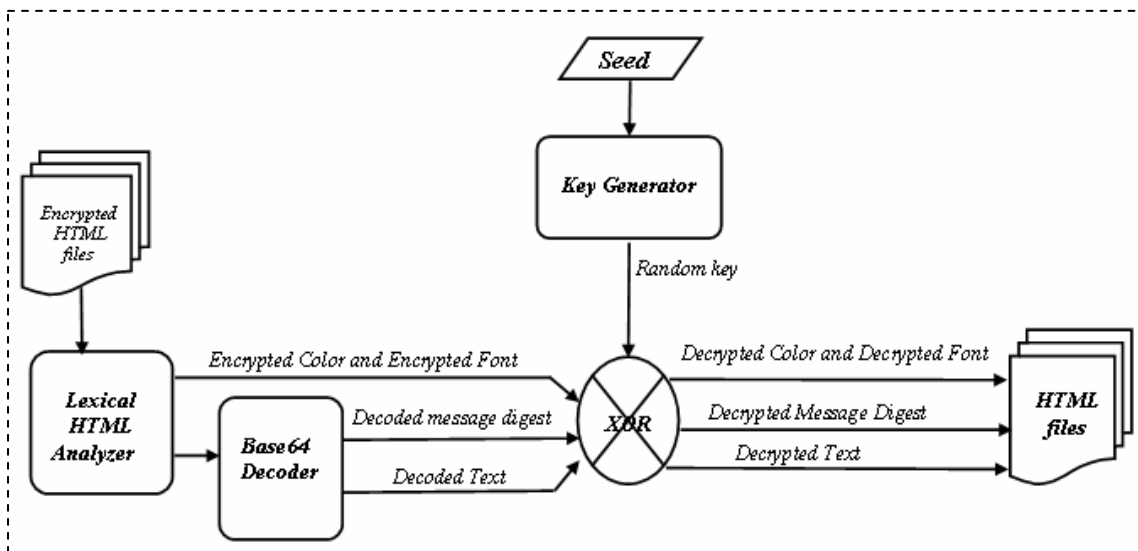
Each one of these two modules consists of sub modules. In the encoding module, HTML file content is passed through the encoding stages, and subjected to various operations to produce encrypted HTML file (encrypted Webpage) carrying a digital signature. In decoding module, the encrypted HTML file is passed through similar stages, and subjected to the same previous sequence of operations, but in reverse order. The functionality of decoding stages is reversed to retrieve the original HTML file (Webpage). The structure of the established HSS and the functionality of its modules will be discussed in details in the next sections.

## 3.3 Encoding Module

As shown in Figure (3.1a), this module consists of many sub modules, which are all together responsible for encrypting the desired HTML file content, and as a result producing the encrypted Webpage. In the following subsections, the functional and structural descriptions for each sub module are given.



a. Encoding Module



b. Decoding Module

Figure (3.1) The structure of HSS

### 3.3.1 Lexical HTML Analyzer

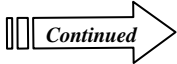
The first step in HSS encoding stage is the lexical analyzing of HTML files. The established lexical HTML analyzer reads HTML files, specifies the fixed attributes within tag contents, and extracts the implied text within body section to be ciphered. For each line found within the HTML file, the lexical analyzer is invoked to perform the intended operations until the end of body section within the HTML file. This analyzer is the major part in the implementation steps of the HSS because it detects the desired tokens to be ciphered.

The lexical HTML analyzer implies the following two main processes:

- A. Parsing process.
- B. Extraction process.

#### A. Parsing Process

Initially, HTML file may be stored on any storage device (or it is downloaded via Internet and temporarily stored on a local storage media). The established system opens the desired HTML file and scans its contents line by line until the (<BODY> or <body>) tag is met; this process is illustrated in Algorithm (3.1).

<b>Algorithm (3.1) Read HTML file</b>	
<b>Goal:</b>	<i>Find &lt;BODY&gt; or &lt;body&gt; tag position of HTML file</i>
<b>Input:</b>	<i>HTML file</i>
<b>Output:</b>	<i>&lt; BODY &gt; or &lt;body&gt; tag position, tags before &lt;body&gt; tag saved in output file(“E_HTML”)</i>
<b>Step1: Open HTML file for read</b>	
<b>Step2: Read first line of the HTML file</b>	
	 Continued

*Step3: Repeat this step until <BODY> or <body> tag is found  
Save line in "E\_HTML" file  
Read next line of "HTML" file*

*Step4: End*

## B. Extraction Process

After finding the <body> tag position, then the extraction process is applied to extract the existing text inside the <body> section. The main text data in HTML files lays in the body part. Since all of information presented by any HTML browser is contained in the body, so the extraction process is applied on the body section only.

As illustrated in Algorithm (3.2), the extraction process is performed on each line inside the body section. The process can be summarized as follows:

1. Read the contents of the line as a string. Check if this string contains color or font tags (attributes). If a color attribute is found, then algorithm (3.8) is called to cipher the color attribute value, while if a font attribute is found, then algorithm (3.9) is called to cipher the font attribute value.
2. Scan the string to find the existence of a pra symbol "<", if it is found then, allocate its position. The existence of this symbol means the beginning of HTML tag. The characters come after this pra symbol should be scanned and collected until reaching the end of the tag (i.e., till meeting the ket ">" symbol). Then the collected characters (i.e., tags content) are saved in output file called "E-HTML". The reason behind Saving HTML tags is to preserve the structure of HTML file.
3. Scan the characters coming after the ket ">" symbol, and collect these characters until reaching the pra "<" symbol. The string of

collected characters that lay between the ket ">" and the pra "<" symbols is ciphered and encoded. For each extracted character within the collected string, the random key generator will be invoked to generate a random byte to cipher this character. During the implementation of this stage, it was noticed that, when a pra-ket symbol ("<" or ">") is detected within the text, the browser will consider it a character entity and replace it with its reference value (for more details about character entities, see chapter-2). In case of ciphering text content (i.e., not color and font attributes); the produced cipher text is encoded using base64 encoding scheme.

4. The overall extracted characters will be accumulated in a single string to be used in computing the message digest of the text using MD5 function (for integrity check).

### **Algorithm (3.2) Extraction Process in HSS Encoder**

**Goal:** Extract text from the <body> container, and identify color and font attributes

**Input:**

HTML file

**Output:**

Output file "E\_HTML"

**Step1:** \\ Initialize buffers and strings

Set Str\_Char  $\leftarrow 0$       \\ Accumulates text characters for message digest process.

Set Num\_of\_char  $\leftarrow 0$

Set Str\_line  $\leftarrow$  ""

**Step2:** \\ Reading, extracting, ciphering and encoding tokens

While not end of HTML file

Read line of HTML file

Assign line from HTML file to Str\_line string

Set Str\_line  $\leftarrow$  Lcase(Str\_line)

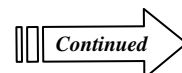
If "color" or "bgcolor" or "text" or "background" in Str\_line is found then

Call algorithm (3.8) to cipher color

If "font" in Str\_line is found then

Call algorithm (3.9) to cipher font

Read Character from Str\_line in <body> container



```

Repeat                                     || Reading characters between tag content
  Save character in "E_HTML" file || Saving tag content in output file to
                                     order the browser to display Webpage

  Read next character from file in sequence
Until character = ">"

Repeat                                     || Finding the desired text after the ending tag
  Read next character from file in sequence || Read the character to be
                                           ciphered

  Call random number generator           || To generate random_number
  Set Cipher_char ← character Xor random_number
  Set Ciphered_char(Num_of_char) ← Cipher_char
  Increment Num_of_char by 1
  If Num_of_char=3 then                 || Counter to encode 3 ciphered characters
    Call algorithm (3.10) to encode ciphered_char vector using Base 64
    encoding
    Set Num_of_char ← 0 || Reset counter for base64 encoding next 3
                        Characters

  Save encoded ciphered_char in "E_HTML" file
End If
Set Str_Char ← Str_Char + character     || Accumulate text to be
                                         processed by MD5 function

Until character = "<"
End While

Step3: || Integrity check
  Call MD5 function to compute the message digest of HTML file
  Call random number generator to cipher message digest
  Call algorithm (3.10) to encode message digest using Base64 Encoding
  Concatenate ciphered encoded text with ciphered encoded message digest
Step4: Return "E_HTML"

```

### 3.3.2 Encrypting HTML File

When confidentiality is a requirement, a high price of private dedicated network connections must be paid or data must be encrypted, so that no one who tries to intercept this data can know its content anyway. Since the encryption provides greater flexibility and lower cost, it is increasingly becoming the predominant solution. In this work, two fast encryption methods were adopted to scramble the text hold within the HTML file, in such a way it becomes safe to send it across the public





```

While Length (Sec_key) < N
  Set Sum ← 0
  For each I from 1 to length(sec_key) do
    Set Sum ← Sum + (K(I,J) × Asc(Mid(Sec_key,I,1)))
  End For
  Set Sum ← Sum mod 256
  Set Sec_key ← Sec_key + Chr(Sum)
  Set J ← j+1
End While
Step2: Return(Padded secret key)

```

After padding, the extended key needs to be converted to the binary form to be used in encryption process. Algorithm (3.4) illustrates the process of conversion to binary form.

#### **Algorithm (3.4) Binary Form Conversion**

**Goal:**

Converting secret key to binary form

**Input:**

Secret key

**Output:**

Vector Bit () of binary digits

**Step1:** *\\ Convert Sec\_key characters to vector Bit() of binary digits*

**Set** j ← -1 *\\Initialize counter*

**For each** I from 1 to N **do**

**Set** k ← Asc(Mid(Sec\_key, I, 1)) *\\ Each time convert one character of secret key to binary form*

**For each** M from 0 to 7 **do** *\\ convert to binary form (8-bits)*

**Set** j ← j + 1: **If** (k And  $2^M$ ) = 0 **Then** Set Bit (j) ← 0 **Else** Set Bit(j) ← 1

**End For**

**End For**

**Step2: Return** vector Bit()

### **3.3.3 Random Number Generators**

After the manipulation of user's key, this key is used to generate the initial values of random number generator parameters, which in turn used to generate pseudo random numbers for encryption transformations.

Two random number generators have been used in HSS, they are:

- A. The Linear Feedback Shift Register (LFSR).
- B. The Proposed Random Number Generator.

### **A. Linear Feedback Shift Register (LFSR)**

Since a long time ago, LFSRs have been used as pseudo-random number generator to perform stream ciphering. In this project, this method was adopted to perform the first step in ciphering stage. The Bit () vector resulted from algorithm (3.4) is input as bits to the shift register.

Linear Feed Back Shift Register (LFSR) implies two processes, as illustrated in algorithm (3.5), they are:

1. Selection process: When selecting bit positions (taps), two considerations need to be taken into account [Til05]:
  - a. The maximal length tap sequences should always be an even number.
  - b. The tap values in the maximal length tap sequence should be relatively prime with each other.
2. Shifting process: The bits contained in selected cells (taps) in the shift register are combined using (XOR) function, and the result is stored in Bit\_out () vector to form the key that used to cipher the text of the HTML file. Each position of the Bit\_out () vector contains the result of Xoring tap sequence for each stage. When the selected bit values are Xored together before the register is shifted, the result of the feedback function is inserted into the shift register during the shift step, to fill the position that is emptied as a result of the shift. Then, the generated random number is used for ciphering process.

### **Algorithm (3.5) Key Generation Method Using Linear Feed Back Shift Register(LFSR)**

**Goal :** Generating keys to encrypt HTML file

**Input:** Secret key

**Output:** Random\_Number

**Step 1:** *\| Construct vector Bit\_out (), each cell contains the result of Xoring tap sequence. Length of Bit\_out () is 8-bit. vector Bit() contains the binary digit of secret key*

*Choose taps from Bit() vector*

*For each k from 0 to 7 do \| Repeating 7 times to form the key used for ciphering, key length is 8-bits(byte)*

*Set Bit\_out(k) ← Bit(2) Xor Bit(7) Xor Bit(17) Xor Bit(31)*

*For each I from 31 to 1 \| Bit() vector size is 32-bit*

*Set Bit(I) ← Bit(I-1) \| Shift cells one bit to the right*

*End For*

*End For*

*Set Bit(0) ← Bit\_out(k) \|Result of Xoring will refill the first position*

**Step2:** *\| Binary to decimal conversion*

*Set Random\_Number ← 0*

*For each I from 7 to 0 do*

*Set Random\_Number ← Random\_Number + Bit\_out(I) × 2<sup>(7-i)</sup>*

*End For*

**Step3:** *Return(Random\_Number)*

## **B. Proposed Random Number Generator**

In this work, a simple and fast random number generator was suggested to extend the level of security; it has lower computational cost in comparison with LFSR. This additional level of security is obtained by using a method to generate long random sequence of numbers. Two built-in functions were used in the generator; Rnd () function and Set Seed () function.

The system uses a secret key of length 20 characters (20×8 bit=160 bit). The secret key is input to a pseudo random number generator, then as a first initial step, three vectors are constructed from the secret key input, these vectors are:

1. Vector  $R\_Seed ()$ , which is composed of six entries, each entry contains an integer value, calculated from the secret key using a convolution process. In this process, every sixth bit of vector  $Bit ()$  is collected in a buffer to form a binary number. Then this binary number is converted to a decimal form. The decimal number will then be modulated by 32768 and the result is stored in the first entry of vector  $R\_Seed ()$  as shown in Figure (3.2). The other five entries of vector  $R\_Seed ()$  will be filled in same way but using the next coming bits (after the sixth bit). The implementation steps are illustrated in step 1 of algorithm (3.6). These six values in  $R\_Seed ()$  vector will be used to reinitialize (start) the random number generator to generate six sequences of random numbers.
2. Vector  $Set\_Rnd ()$ ; which consists of six entries, each entry contains an integer value; the value of these entries are calculated using the same steps followed for calculating the entries values of  $R\_Seed ()$  vector, but the difference is that the process will start from location twelve in  $Bit$  vector  $()$ , and taking every coming seventh location. The result will be modulated by 256 and stored in the first entry of  $Set\_Rnd ()$  vector. The next entry value of  $Set\_Rnd ()$  will be calculated by applying the same previous steps with a start point at thirteenth bit. After filling  $Set\_Rnd ()$  vector, its contents will be used to make jumps in the calls of  $Rnd ()$  function. The implementation steps are illustrated in step 2 of algorithm (3.6).
3. Vectors  $Sqg1 ()$  to  $Sqg6 ()$ ; each vector is composed of pre-specified prime number of entries. These six vectors contain random sequence of numbers, which will be used to encrypt the plain text. The

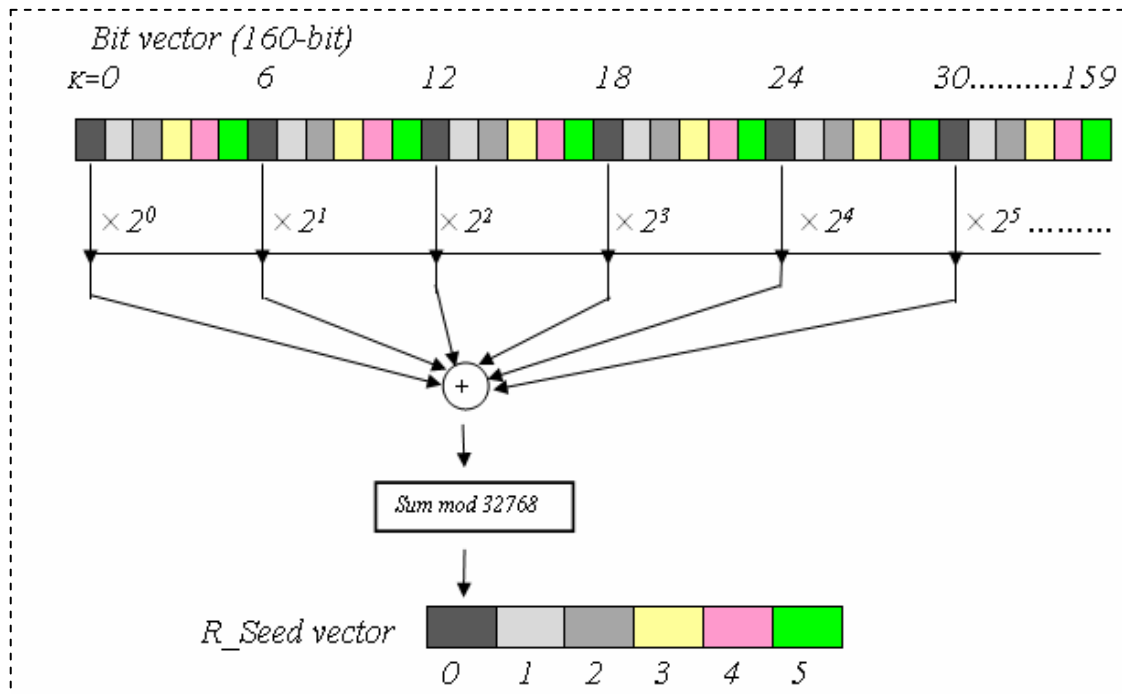


Figure (3.2) The process of filling the first location of R\_Seed vector

**Algorithm (3.6) Proposed Random Number Generator****Goal :**

Generating keys to encrypt HTML file

**Input:**

Vector Bit()

**Output:**

Random\_Number

**Step 1:** *Construct vector R\_Seed(), each location contains seed values*

```

For each I from 0 to 5 do           Number of R_seed() locations
  Set R_Seed (I) ← 0
  Set K ← I                         Begins from index I of Bit()vector
  Set J ← 1
  While K < 159 do                 Sec_key consists of 20 characters
    Set R_Seed(I) ← R_Seed(I) + J × Bit(K)
    Set J ← J × 2                   Convert binary bits to decimal
    Set K ← K + 6                   Overlapping(jump) every 6 positions in Bit() vector
  End While

```

```

Set R_Seed (I) ← R_Seed (I) mod 32768  || 32768 is the maximum
                                         positive value of Randomize
                                         statement

End For

Step2: || Construct vector Set_Rnd() ,each location contains number of times the
      Rnd() statement is repeated

For each I from 0 to 5 do                ||Number of Set_Rnd() locations
  Set Set_Rnd(I) ← 0
  Set K ← I+12                          || Begins from index 12 of Bit()vector
  Set J ← 1
  While K < 159 do
    Set Set_Rnd (I) ← Set_Rnd (I)+J×Bit(K)
    Set J ← J×2
    Set K ← K+7                          || Overlapping(jump) 7 positions in Bit() vector
  End While
  Set Set_Rnd (I) ← Set_Rnd (I) mod 256  || Restricting the limit of
                                         numbers to 256

End For

Step3: ||Construct six sequence generator vectors, Sqg1, Sqg2, Sqg3, Sqg4, Sqg5,Sqg6
      || Rnd is a function that returns random number.
      || Randomize is a function initialize random-number generator, and generate
      new sequence of random numbers

Set M1 ← 252:Redimension Sqg1 to M1
Randomize R_seed(o)                    ||Set seed value according to the first value of
                                         R_seed() vector

For each J from 0 to M1 do
  For each I from 0 to Set_Rnd (o) do
    Set X ← Rnd                          || X is dummy variable, Rnd function is repeated
                                         until the value of( Set_Rnd(0)) is reached

  End For
  Set Sqg1(J) ← 255 × Rnd  ||Filling( Sqg1) vector with the random values
End For

|| Construct Sqg2, Sqg3, Sqg4, Sqg5,Sqg6 in same way as Sqg1 Such that
M2=312,M3=230, M4=258 , M5=282 and M6=196 ,these numbers must be
relatively prime
|| initialize counters
Set K1 ← -1: K2 ← -1: K3 ← -1: K4 ← -1: K5 ← -1: K6 ← -1

Step 4: || Generate random number
Call algorithm (3.7) to generate random number

Step 5: Return(Random_Number)

```

After carrying out the steps of algorithm (3.6), then algorithm (3.7) is invoked to generate the required random number by Xoring the values

stored in vectors (Sqq1 () to Sqq6 ()). Algorithm (3.7) is invoked when there is a need to cipher each character in the plaintext.

### **Algorithm (3.7) Random Number Generator**

**Goal :**

*Generating random numbers using the proposed random generator to encrypt HTML file*

**Input:**

*( Sqq1(),Sqq2(), Sqq3(), Sqq4(), Sqq5(),Sqq6()) vectors  
(K1,K2,K3,K4,K5,K6) counters  
(M1,M2,M3,M4,M5,M6) constants*

**Output:**

*Random\_Number*

**Step1:** *\\ Generating random number from sequence generator vectors*

*If K1 < M1 then Set K1 ← K1 + 1 else Set K1 ← 0*

*If K2 < M2 then Set K2 ← K2 + 1 else Set K2 ← 0*

*If K3 < M3 then Set K3 ← K3 + 1 else Set K3 ← 0*

*If K4 < M4 then Set K4 ← K4 + 1 else Set K4 ← 0*

*If K5 < M5 then Set K5 ← K5 + 1 else Set K5 ← 0*

*If K6 < M6 then Set K6 ← K6 + 1 else Set K6 ← 0*

*Set Random\_Number ← Sqq1(K1) Xor Sqq2(K2) Xor Sqq3(K3) Xor Sqq4(K4)  
Xor Sqq5(K5) Xor Sqq6(K6)*

**Step2 :***Return(Random\_Number)*

### **3.3.4 HSS Encoder**

The HSS encoder performs three tasks to secure HTML file, they are:

#### **A. Text Contents Encryption**

HSS uses stream cipher encryption. As mentioned in chapter 2, stream cipher could be arranged to cipher one byte (or character) at each call instance. So, the sequence of random numbers (bytes) generated by each of the two random generators of HSS is used to encrypt the characters of the extracted strings from the body container. The steps of ciphering process are shown within algorithm (3.2).

## B. The Value of Color Attributes Encryption

This process is concerned with encrypting the color values found in the scanned HTML file. The reason behind changing the color is to increase the probability of not recognizing the visual appearance of the Webpage by unauthorized users. Thus, this process can add more confusion to eavesdroppers. Since, color values in HTML file can be referred either by name or by hexadecimal numbers (as mentioned in chapter-2), the color ciphering is performed in two ways according to its identified form in HTML.

Algorithm (3.8) illustrates the steps taken to cipher the values of color attribute. This algorithm is invoked when a color attribute (bgcolor, color or text) is met during the scanning stage of each line in the HTML file. The value of the designated attribute will be ciphered in two ways, according to its form, and they are:

1. If the color value is in hexadecimal form; then a “#” symbol must be detected. After that the values of the three-color components (red, green, blue) are extracted separately. The value of each color component is represented by two hexadecimal digits, and it is ciphered, by, first, converting it to a decimal form and, then, Xoring it with the generated random number. After ciphering the three-color components, the ciphered values (in their hexadecimal form) of the color components are concatenated to form the ciphered color value, which is replaced with the original color value. The entire background and foreground color values of the existing text in Webpage are encrypted.
2. If a name is used to refer to a color value in HTML file, then the list of color names used in HTML (shown in Table 2.4 in chapter-2) is used as a reference table. When one of the color names is met during the line scan process, then the reference color list will be searched to determine



the position of this color name in the color reference table. The position index of the found color name will be Xored with the generated random byte, and then the ciphered result value will be used as a position index to get the ciphered color name from the reference color table.

### **Algorithm (3.8) Encrypting Color of Webpage**

**Goal :** Changing color of the Webpage

**Input:** String (Str\_line)

**Output:** Ciphered\_color

**Step1:** *\\Construct vector table\_name() of all common color names used in HTML*

**Step2:** *\\If 'color' or 'bgcolor' or 'text' or 'background' word is found*

**While “color” or “bgcolor” or “text” or “background” found do**

**Read** next character

*\\ Find the '#' position in the line*

**If** '#' is found **then** *\\ Indicates that color is referred by hexadecimal number*

**Set** color\_component ← 0

**Read** the three color components(RGB)

**Put** first color component(Red) in color\_component

**Call** random number generator

**Compute** cipher\_color ← (Val("&h"+color\_component)) Xor random\_number

**Replace** color\_component **with** cipher\_color

*\\Manipulate other two color components in the same way as with red color*

**Write** line in E\_HTML file

**Else** *\\ If '#' not found, indicates that color is referred by name*

**Find** Position of the beginning ' ' in the line after color attribute

**Set** color\_name ← 0

**Repeat**

**Read** next character *\\ Extract color name*

**Set** color\_name ← color\_name+ character

**Until** the ending ' ' reached

*\\ Find color\_name in the table\_name*

**For** table\_index from 0 to size of table\_name **do** *\\ size of table\_name=17*

**If** color\_name = table\_name(table\_index) **then**

**Get** table\_index of table\_name

**Call** random number generator

**Set** New\_index ← ABS(random\_number-(table\_index Xor random\_number))+1

**Set** Cipher\_color ← table\_name(New\_index)

**Replace** color\_name with cipher\_color in the line

**End If**

**End For**

**End If**

**End While**

**Step3:** **Return**(ciphered\_color)

### C. The Font Size Elements Encryption

This stage is concerned with encrypting the font size values might be met when scanning the HTML file. The first step in the process of encrypting font size values is the extraction of the size value and then Xor it with a generated random number. The ciphered value is then replaced with the corresponding original size number. Algorithm (3.9) illustrates the steps implemented for encrypting the font size.

This size range limitation is due to the relative scale of font size attribute in HTML files, it is bounded to be within the range [1..7]. The size attribute value 5 can be assigned in two different ways: (1) the size can be stated absolutely, with a statement like <size=5>, or (2) can be assigned relatively with respect to the default font size “which is 3”, for example the font size becomes 5 when the statement <size=+2> is met. These two size assignment cases have been taken into consideration in the implementation steps of encrypting font size values.

#### **Algorithm (3.9) Encrypting Font of the Webpage**

**Goal:**

*Changing font of Webpage*

**Input:**

*String (Str\_line)*

**Output:**

*Ciphered\_font*

**Step1:** *\\ When “font” word is found in Str\_line*

*Set Size\_number ← 0*

*Read next characters in string after “font” word*

*Find “size” word*

**Step2: Repeat**

*Read next characters in the string after “size” word*

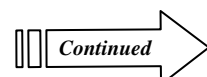
*Until beginning of ‘ ‘ is reached*

**Step3: Repeat**

*Read next character in the string*

*If character = “+” then*

*Read next character \\Skip the plus sign if found(see font table in chapter2)*



```

End If
  Set Size_number ← Size_number + character    \\Required size number is
Until ending of ‘ ‘ is reached                between quotation marks
  Set size_number ← Val(size_number)         \\Convert string to numeric
  Call random number generator
  Set Cipher_font ← ABS(random_number - (size_number Xor random_number)) + 1
  Replace size_number with cipher_font
  Write line in “E_HTML” file
Step4:Return (Cipher_font)

```

The main consideration taken when the proposed system perform text, color and font encryption is preservation of the main structure of the HTML file; so that the Web browser can interpret the HTML file correctly with no error, and the Webpage can be transmitted correctly through the internet via HTTP (Hyper Text Transfer Protocol).

### 3.3.5 MD5 Function (Message Digest 5)

In our proposed system, integrity is the second requirement that must be fulfilled after confidentiality. To attain a high level of integrity confidence to the text lay inside the body of HTML, a method must be put in place to prevent or detect alteration during transit. The method employed in this work is based on using MD5 (Message Digest) hash function. A hash function takes the text, of any length, and computes a product value of fixed length (hash value). The hash value can be used to determine if HTML file has been subjected to modification. Figure (3.3) depicts the overall encoding process for computing the digest using MD5 hash function and embedding it within the HTML file to produce a signed HTML file.

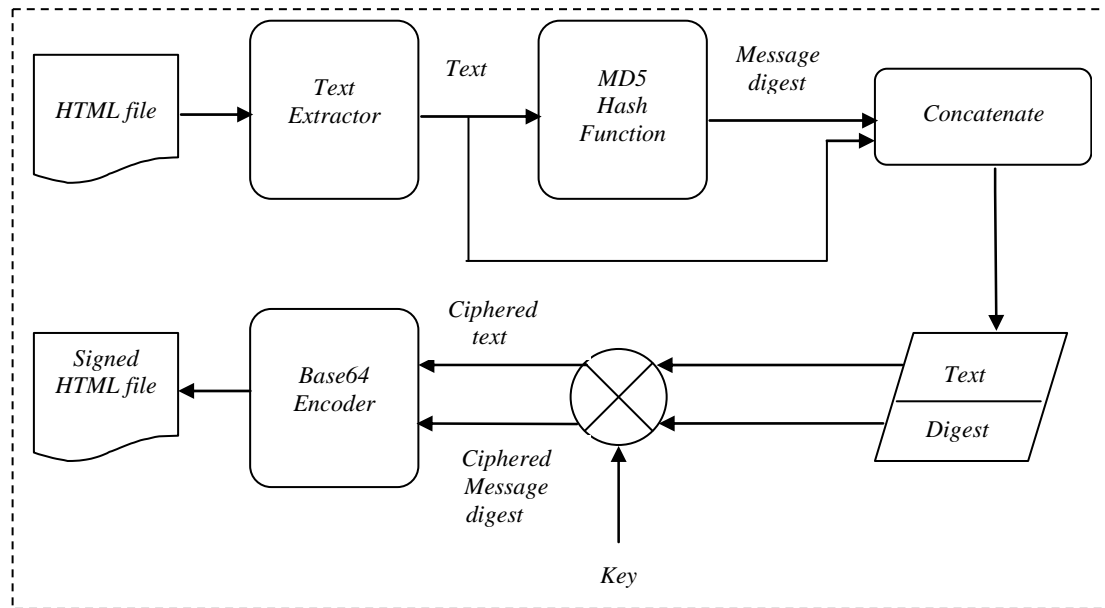


Figure (3.3) HTML digest generation using MD5 hash function

The following list clarifies the operations applied to determine the message digest of text lay within the HTML file:

1. After extracting the whole text from the body of the HTML file, as illustrated in algorithm (3.2), it is stored in as a string (*Str\_Char*). This string is entered as input to MD5 hash function.
2. The output of MD5 hash function (128 bit long) is called hash value or message digest. The probability of finding other message have same digest is of order of  $2^{-128}$ .
3. The hash value is concatenated with a synchronous pattern and put between two tags, after the `<\HTML>` tag. Only the intended receivers who have the secret key can check the message digest value. This hash value could be used as a fingerprint to check the integrity of HTML file after receiving it at the receiver side.
4. The extracted text and the calculated hash value are encrypted using one of the suggested encryption methods. The reason behind encrypting the hash value (message digest) is that the eavesdroppers cannot utilize the

message digest of the HTML file even if they can get and succeed in reading it.

5. After making encryption, to make the result of encrypted message digest readable by the receiver, the base64 encoding is used to convert the ciphered result into a screen printable form. The encoding process is shown in algorithm (3.10). The encrypted coded message digest will not be visible for the users (when using HTML browsers), only the decoder can display it using its own secret key.

### 3.3.6 Base64 Encoder

To transmit ciphertext data over a text-only medium (such as the body section of HTML file) an encoding method is needed to convert the binary ciphertext to only text form that still ambiguous to unauthorized recipient. The existence of binary data in HTML file is not acceptable. Binary data can be corrupted when it is passed through Web routes. Some systems may interpret some of the binary bytes as control characters and causes unintended results. For example, some control characters may cause certain actions by some software applications, like Control-C (also abbreviated as ^C) terminates a process.

In HSS encoder, the ciphertext data is coded using base64 encoding as shown in algorithm (3.10). The input to this algorithm is three ciphered bytes; each byte, which is 8 bit long, will be converted to a binary form. Then, all the binary digits of the three bytes are arranged in one array `Binary_digit ()`. To get the resulting base64 encoding characters, the `Binary_digit ()` array is partitioned into four groups, and each group will consist of six bits. Then, the six bits of each group are combined to form an integer number, which in turn is used as index to choose a character from the base64 table (i.e., Table 2.6 in chapter-2). The resulting sequence of base64 characters are concatenated to form the cipher text that can be

easily laid within the HTML file instead of the original text. This base64-coded text can be decoded back into binary at the destination side before decryption.

As depicted in example (2.1 in chapter-2), the reason behind not adopting hexadecimal encoding in our proposed system is that “each ASCII character (which is one byte long in storage space) is represented by two hexadecimal nibbles (digits), so any text translated into hexadecimal form will be exactly twice as big as the original data”. This might not seem like a problem for a short message, but sending a megabyte or more, the original size will be doubled. Sending this over a slow Internet connection takes twice time as long. The output of hexadecimal coding scheme is longer than that produced by base64 (which takes one third as much space), and for this reason the later coding is adopted in HSS.

### **Algorithm (3.10) Base64 Encoding**

**Goal:**

*Converting unreadable characters into readable form*

**Input:**

*Vector ( Ciphared\_char ), Num\_of\_char*

**Output:**

*Base64\_character1, Base64\_character2, Base64\_character3, Base64\_character4*

**Step1:** *\\ Filling last(1 or 2) byte of HTML file with 0's when the size of HTML file not divisible by 3*

**Initialize** *Sum\_character1, Sum\_character2, Sum\_character3, Sum\_character4 to 0*

**While** *(Num\_of\_char mod 3) <> 0 do* *\\When (Num\_of\_char) less than 3*

*Set Ciphared\_char (Num\_of\_char) ← 0* *\\ Fill last bytes with 0*

**Increment** *Num\_of\_char by 1*

**End While**

**Step2:** *\\ In base64 encoding, three ciphared characters at a time are taken.*

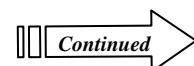
**Set** *M ← 7* *\\ Saving from LSB(least significant bit) of byte*

**For** *I from 1 to 3 do* *\\ Number of ciphared characters to be encoded*

**For** *J from 0 to 7 do*

*Set Binary\_digit(M-J) ← (Ciphared\_char(I) And (2<sup>j</sup>)) \ 2<sup>j</sup>*

**End For**



```

    Increment M by 8           \\ Each character is 8-bit(byte)
End for

Step3:\\Convert the 24 bits from three 8-bit groups to four 6-bit groups and
    Convert each of the four 6-bit groups into decimal form.
For I from 5 to 0 do           \\ Six digits
    Set Sum_character1  $\leftarrow$  Sum_character1+Binary_digit(I) $\times 2^{5-i}$ 
End For
For I from 11 to 6 do
    Set Sum_character2  $\leftarrow$  Sum_character2+Binary_digit(I) $\times 2^{11-i}$ 
End For
For I from 17 to 12 do
    Set Sum_character3  $\leftarrow$  Sum_character3+Binary_digit(I) $\times 2^{17-i}$ 
End For
For I from 23 to 18 do
    Set Sum_character4  $\leftarrow$  Sum_character4+Binary_digit(I) $\times 2^{23-i}$ 
End For

Step4:\\ Use each of the four decimals to look up the base64 character code
    Set Base64_character1  $\leftarrow$  Chr(Base64_table(Sum_character1))
    Set Base64_character2  $\leftarrow$  Chr(Base64_table(Sum_character2))
    Set Base64_character3  $\leftarrow$  Chr(Base64_table(Sum_character3))
    Set Base64_character4  $\leftarrow$  Chr(Base64_table(Sum_character4))

Step5:Return(Base64_character1, Base64_character2)
    (Base64_character3, Base64_character4)

```

To encode characters, the table of base64 code words needs to be constructed. Algorithm (3.11) shows the steps of base64 table construction process.

### **Algorithm (3.11) Generation of Base64 Table**

**Goal:**

*Set up a mapping of values (0 through 63) to base64 characters (A-Z, a-z, 0-9, '+', and '/')*

**Input:**

*Empty (Base64\_table) vector*

**Output:**

*(Base64\_table) vector filled with base64 characters*

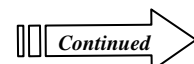
**Step1:** *\\Construct base64 table*

**Set** J  $\leftarrow$  0

**For** I from Asc("A") to Asc("Z") **do**

**Set** Base64\_table(J)  $\leftarrow$  I

**Increment** J by 1



```

End For
For I from Asc("a") to Asc("z") do
  Set Base64_table(J) ← I
  Increment J by 1
End For
For I from Asc("0") to Asc("9") do
  Set Base64_table(J) ← I
  Increment J by 1
End For
Set Base64_table(J) ← Asc("+")
Increment J by 1
Set Base64_table(J) ← Asc("\")
Step2:Return(Base64_table)

```

### 3.4 Decoding Module

The main stages of the decoding module are illustrated in Figure (3.1 b). As mentioned previously, the sequence of its stages has the inverse order in comparison with the order of the encoding module stages.

The following subsections describe the functionality of the main stages of the decoding module with an illustration to the way of implementing them.

### 3.5 HSS Decoder

In decoding process, the HSS decoder asks the user to assign the name of the encrypted HTML file. Then, it is loaded, and then the lexical HTML analyzer is invoked to parse the encrypted HTML file and extracts the desired tokens. In HSS decoder, the same parsing process (i.e., algorithm 3.1, which is previously illustrated in HSS encoder section) is invoked but the difference is reading the encrypted HTML file to be parsed, decoded and outputting the reconstructed HTML file. The extraction process in HSS decoder is same like that used in HSS encoder (i.e., algorithm 3.2 that previously mentioned in section 3.3.1); the difference is manipulating the base64 characters first then deciphering the characters to



get the original text. Also, in extraction stage the ciphered values of color and font attributes are deciphered to retrieve their original values. Algorithm (3.12) shows how the extraction process is invoked in HSS decoder and how it works to extract the encrypted tokens.

### **Algorithm (3.12) Extraction Process in HSS Decoder**

**Goal:** Extract encrypted and encoded text from the <body> tag

**Input:**

”E\_HTML” file

**Output:**

Output file “HTML” file

**Step1:** *\\ Initialize buffers and strings*

*Set Str\_Char  $\leftarrow$  0* *\\ Accumulates text characters for message digest process.*

*Set Num\_of\_char  $\leftarrow$  0, Str\_line  $\leftarrow$  “ ”*

**Step2:** *\\ Reading, extracting, ciphering and encoding tokens*

**While** not end of “E\_HTML” file

*Read* line of “E\_HTML” file

*Assign* line from E\_HTML file to Str\_line string

*Set* Str\_line  $\leftarrow$  Lcase(Str\_line)

**If** “color” or “bgcolor” or “text” or “Background” in Str\_line is found **then**

*Decrypt* color value

**If** “font” in Str\_line is found **then**

*Decrypt* font size value

*Read* Character from <body> container in E\_HTML file

**Repeat** *\\ Reading character between tag content*

*Save* character in “HTML” file *\\ Saving tag content in output file to*

*order the browser to display Webpage*

*Read* next character from file in sequence

**Until** character = “>”

**Repeat** *\\ Finding the desired text after the ending tag*

*Read* next character from file in sequence *\\ Read the character to be decoded and deciphered*

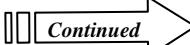
*Set* Str\_Char  $\leftarrow$  Str\_Char + character *\\ Accumulate text to be processed by MD5 function*

**Increment** Num\_of\_char by 1

**If** Num\_of\_char=4 **then** *\\ Counter to decode 4 ciphered characters*

*Call* algorithm (3.13) to decode ciphered\_char vector using base64 decoding

**For** I from 1 to 3 **do** *\\ Returned 3 characters from decoding process*

 Continued

```

    Call random number generator
    Set Deciphered_char ← Asc(Decoded_char(I)) Xor Random-number
    Save Chr(Deciphered_char) in "HTML" file
  Next I
    Set Num_of_char ← 0  ||Reset counter for decoding next 4 ciphered
                        Characters
  End If
  Read next character from file in sequence
  Until character="<"
  End While
Step3: Return "HTML" file

```

After analyzing the encrypted HTML file, HSS decoder asks the user to enter the secret key. Only the intended recipients must know this secret key, this is a major condition because HSS uses stream ciphering and, consequently, should satisfy the common conditions of symmetric cryptosystem. The processes, padding and conversion to binary form are invoked in a way similar to that mentioned in HSS encoder; see algorithms (3.3) and (3.4).

The HSS decoder performs the following tasks to retrieve the original HTML file:

- A. Base64 decoding the ciphered text found in the body section, then
- B. Decrypting the decoded text contents.
- C. Decrypting the values of found ciphered color attributes.
- D. Decrypting the values of ciphered font size.
- E. Checking the integrity of HTML file.

In the following next sections, the implementation of the above-mentioned tasks is explained.

## A. Base64 Decoder

In HSS decoder, the ciphered text is first decoded (i.e., converted from base64 character set to ASCII character set) before decrypting it. So, base64 decoder is invoked to decode the ciphered characters. During the

extraction stage, every four-base64 characters are taken from the ciphered text, and sent them to base64 decoder to convert them, as one set, into three ASCII characters. Algorithm (3.13) illustrates base64 decoding process.

### Algorithm (3.13) Base64 Decoding

**Goal:**

Converting base64 coding characters into ASCII 8-bit characters

**Input:**

Vector Ciphered\_char(), Num\_of\_char

**Output:**

ASCII\_character1, ASCII\_character2, ASCII\_character3

**Step1:** || Filling last(1 or 2) byte of HTML file with A's when the size of HTML file not divisible by 4

**Initialize** Sum\_character1, Sum\_character2, Sum\_character3 to 0

**While** (Num\_of\_char mod 4) <> 0 do ||When Num\_of\_char less than 4

**Set** Ciphered\_char (Num\_of\_char) ← "A" || Fill last bytes with A's

**Increment** Num\_of\_char by 1

**End While**

**Step2:** || In base64 decoding, four ciphered characters at a time are taken.

**For** I from 0 to 3 do || Number of ciphered characters to be decoded

**For** J from 0 to 63 do

**If** ASC(Ciphered\_char(I))= Base64\_table(J) **then**

**Set** Ciphered\_ch (I) ← J

**Exit For** J

**End If**

**End For** J

**End For** I

**Set** M ← 5 || Saving from LSB(least significant bit)

**For** I from 0 to 3 do || Number of ciphered characters to be decoded

**For** J from 0 to 5 do

**Set** Binary\_digit(M-J) ← (Ciphered\_ch (I) And (2<sup>J</sup>)) \ 2<sup>J</sup>

**End For**

**Increment** M by 6 || Each character is 6-bit

**End For**

**Step3:** || Convert the 24 bits from four 6-bit groups to three 8-bit groups and Convert each of the three 8-bit groups into decimal form.

**For** I from 7 to 0 do || eight bits

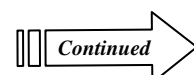
**Set** Sum\_character1 ← Sum\_character1 + Binary\_digit(I) × 2<sup>7-i</sup>

**End For**

**For** I from 15 to 8 do

**Set** Sum\_character2 ← Sum\_character2 + Binary\_digit(I) × 2<sup>15-i</sup>

**End For**



```

For I from 23 to 16 do
  Set Sum_character3 ← Sum_character3+Binary_digit(I)×  $2^{23-i}$ 
End For

Step4:\\ the 8-bit character is returned by taking the Chr() function of the sum
  Set ASCII_character1 ← Chr( (Sum_character1))
  Set ASCII_character2 ← Chr( (Sum_character2))
  Set ASCII_character3 ← Chr( (Sum_character3))

Step5:Return (ASCII_character1, ASCII_character2
  ASCII_character3)

```

As illustrated in the above algorithm, each set of four-base64 characters is entered as an input, then the table index of each base64 character is assigned. The index value of each base64 character is converted to its binary form to construct the vector Binary\_digit () that has 24-bit length. Then, this vector is partitioned into three portions, each consist of 8-bits. The bits of each portion are combined to get the index value of the ASCII character, which is in turn, is established using the Char () function.

## B. Decrypting Text

After the retrieval of ciphered ASCII characters using base64 decoder, these characters need to be deciphered. So, the same two random number generators, used in encoding module, are invoked to generate the same sequence of random bytes in order to decrypt the ciphered text. The major condition that must be hold when using HSS decoder is that “the recipient must choose the same secret key that was chosen at HSS encoding stage”, this condition is necessary to get the same sequence of random numbers (bytes). Algorithm (3.5) illustrates the process of generating a random number using linear feed back shift register (LFSR), and algorithm (3.6) illustrates the steps of the proposed random number generator. The random byte generated by the random number generator is used to decrypt

the ciphered text characters inside the body section by performing the XOR operation. Algorithm (3.12) illustrates the applied steps to perform the XOR operation.

### **C. Decrypting the Value of Color Attributes**

In HSS decoder, the value of color attributes of the encrypted HTML file is decrypted. During the extraction process, the lexical HTML analyzer searches the lines of the encrypted HTML file to find the color attributes that had been encrypted in encoding stage. Then, the process of decrypting the color value is performed using steps like those followed in encryption process (see algorithm 3.8) but in reverse order. In brief words, the random number generator is invoked to decrypt the three-color components (RGB) of the ciphered color value and the ciphered color name to obtain the original color of the text found in the decrypted Webpage.

### **D. Decrypting the Value of Font Size**

In a way similar to that followed for decrypting the value of color attributes, the value of font size is decrypted. In HSS decoder, the lexical HTML analyzer scans the encrypted HTML file to detect and extracts the font elements. When it detects a font size attribute, its value is decrypted by Xoring its extracted value with the generated random number. The steps of the decryption process are like those applied in encryption process (i.e., algorithm 3.9) but in reverse order. The ciphered value of font size is Xored with the generated random number to obtain the original value of the font size.

## E. Checking the Integrity of HTML File

In HSS decoding phase, the integrity of HTML file must be checked. As shown in Figure (3.4), the integrity check process of HTML file can be done by applying the following steps:

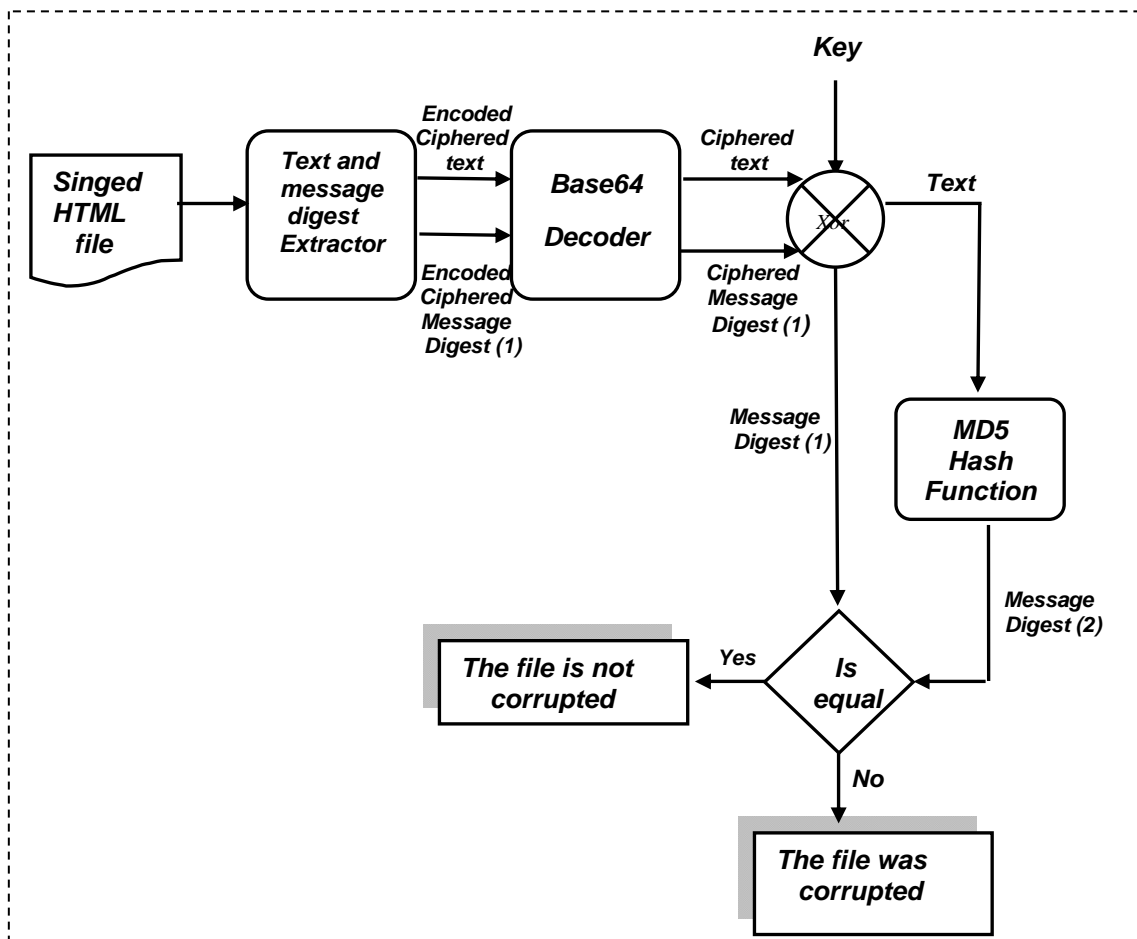


Figure (3.4) The integrity check process

1. At the receiver, the signed HTML file (produced by HSS encoder) is input to the text and message digest handler, in order to extract the encoded ciphared text and the encoded ciphared message digest that was created in the encoding phase. This message digest is numbered by (1) in Figure (3.4). The reason behind extracting the encoded ciphared text is to redetermine the digest (i.e., digest (2) in the Figure) by applying

MD5 hash function on the decoded and decrypted text which was extracted from the encrypted HTML file.

2. The encoded ciphered message digest (1) is input to base64 decoder to be decoded (i.e., converted from base64 characters to ASCII characters).
3. Then the ASCII message digest is decrypted using the same secret key that was chosen at the HSS encoding stage.
4. As a final step in the integrity checking stage, a comparison process is done between message digest (1) and message digest (2). If they are equal, it indicates that the transmitted HTML file was not modified or altered during its transition via internet channel, and a message “*The HTML file is not corrupted*” is prompted to user. Otherwise, if the two digests are not equal, then this indicates that a modification was done on the transmitted HTML file, in such case the message “*The HTML file was corrupted*” is issued to acknowledge the user.

Algorithm (3.14) illustrates the applied steps for checking the integrity of the signed HTML file.

#### **Algorithm (3.14) Integrity Check**

**Goal:**

*Checking the integrity of sent HTML file*

**Input:**

*Singed HTML file*

**Output:**

*A prompted message*

**Step1:** *Integrity check*

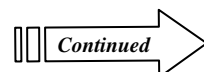
*Extract message digest from “E\_HTML” file*

*Call algorithm (3.13) to decode message digest using Base64 decoding*

*Call random number generator*

*Decrypt message digest(1)*

*Call MD5 function to compute the message digest(2) of decrypted “E-HTML” File*



*Compare message digest (1) and (2)*  
*If message digest(1)=message digest(2) then*  
*A message“ The HTML file is not corrupted”*  
*Else*  
*A message” The HTML file is corrupted”*  
*End if*  
**Step2: Return** (prompted message)



# *Chapter Four*

## *Implementation and Testing*

### **4.1 Introduction**

This chapter is devoted to present the implementation part of our research work, and to present the established user interfaces of the system. Also, in this chapter the results of the conducted randomness tests are presented in order to evaluate the secrecy level of the system, taking into consideration that two types of random generators have been used to perform ciphering, the first one is Linear Feedback Shift Register (LFSR) and the second one is our proposed random generator.

The results of conducted statistical tests on the ciphertext produced by using each of the random generators are displayed for comparison purpose. Beside to random test results, the elapsed encoding time is computed to evaluate the system performance.

The proposed system was established using Visual Basic (version 6.0) programming language. The developed programs were tested under the environment of windows XP service pack2 operating system, the hardware environment was a laptop computer (Processor Intel ® core ™ Duo CPU 1.83 GHz, RAM 504 M-byte).

## 4.2 System Implementation

Figure (4.1) shows the start form of HSS system.

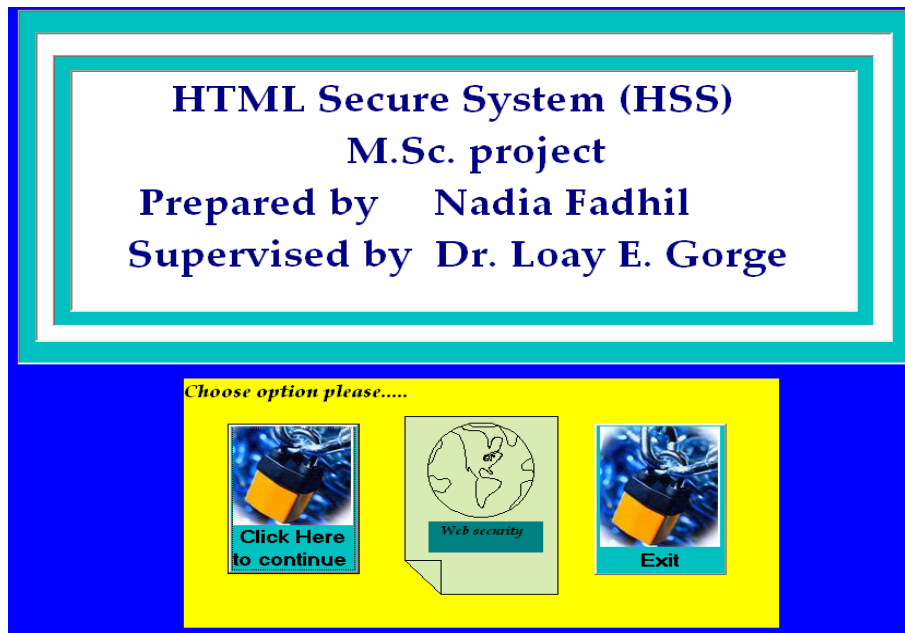


Figure (4.1) HSS interface

While Figure (4.2) shows the main form, its menu consists of the following options:

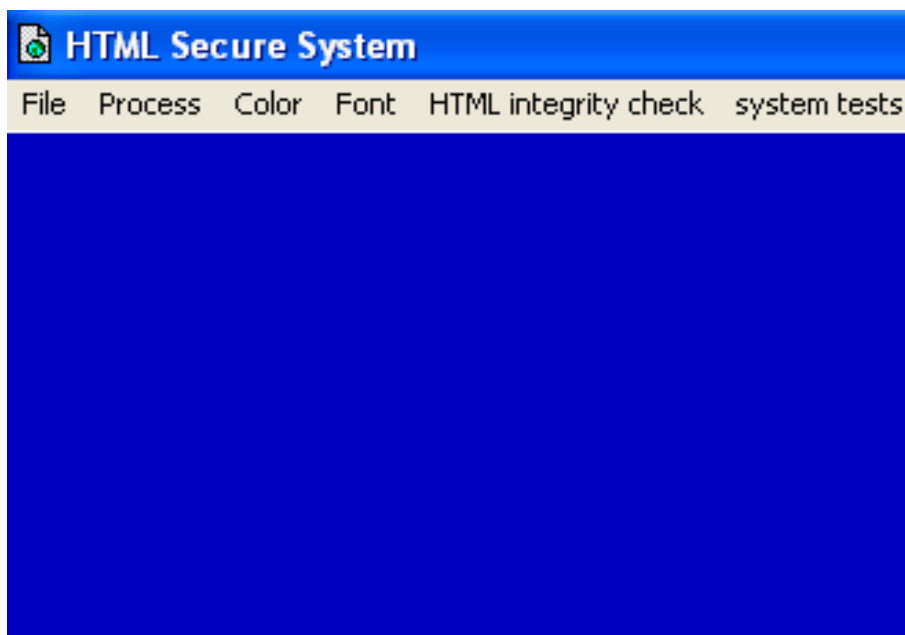
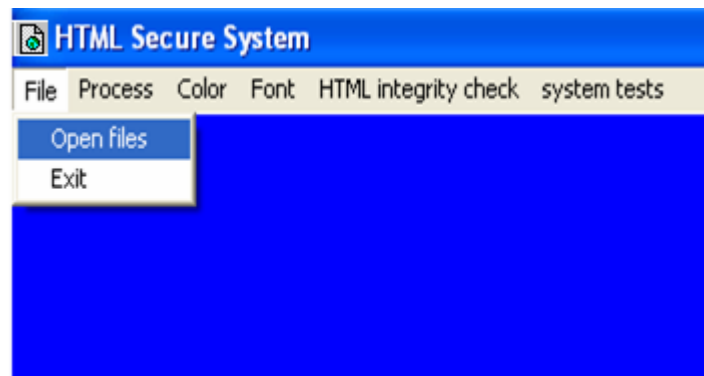


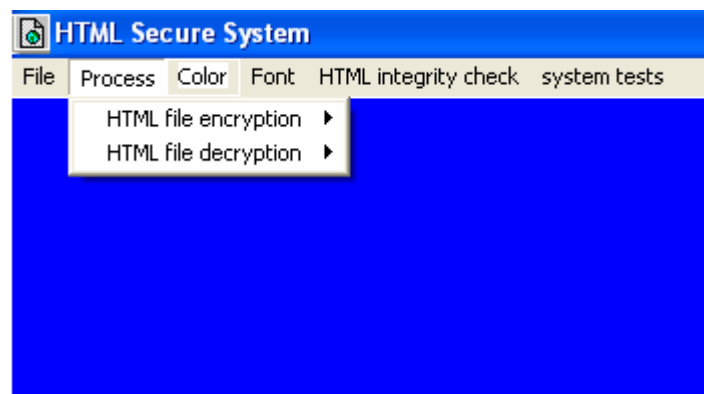
Figure (4.2) HSS main menu

1. **File:** through this option, the user can select one of the existing HTML files stored in the local storage media and display its contents by making a call to the Internet explorer. Also, HSS system program could be terminated by selecting the *Exit* option. Figure (4.3) presents the File sub menu.



**Figure (4.3) File sub menu**

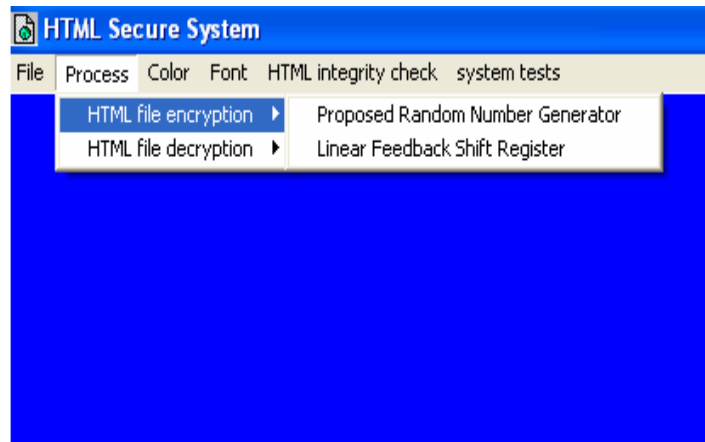
2. **Process:** this menu option consists of the following two sub-menu lists as shown in Figure (4.4):



**Figure (4.4) Process sub menu**

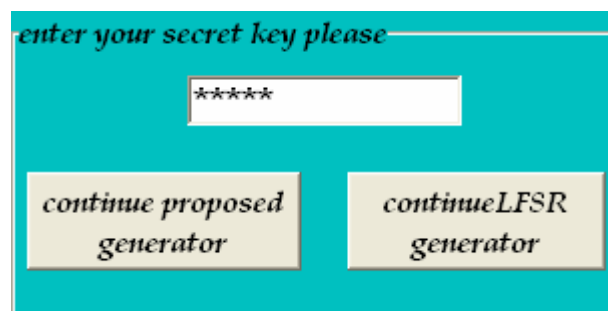
- A. **HTML File Encryption:** When a user make a mouse click on this menu option, another sub menu will be displayed to let the user select the type of used random generator. Through HSS, the user can select one of the two random number generators (i.e., either

*Linear Feedback Shift Register or the proposed random number generator*) as shown in Figure (4.5).



**Figure (4.5) HTML file encryption sub menu**

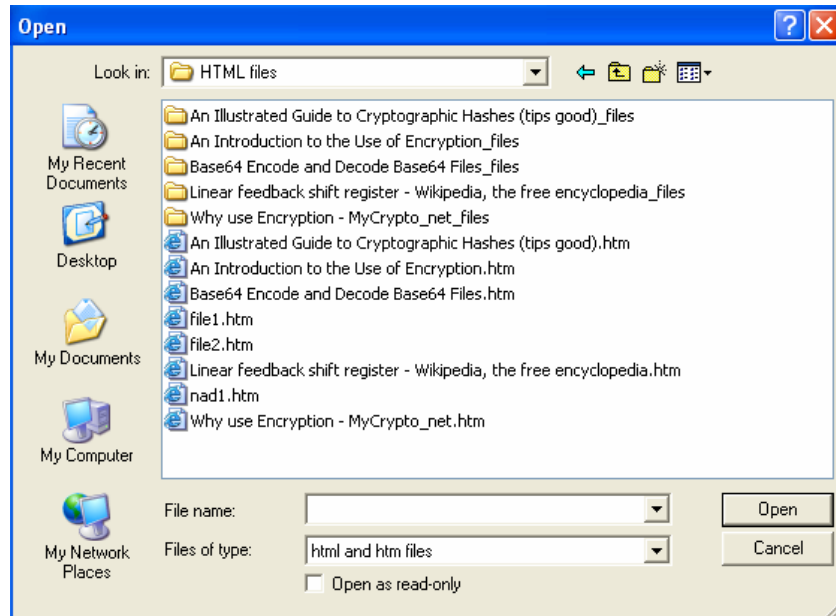
Once the type of the random generator is selected, then a text box, shown in Figure (4.6), will displayed asking the user to enter the secret key. If the user's entry is less than 20 characters (i.e., length of the secret key), then HSS generates the required padded characters to keep the length of secret key equal to 20.



**Figure (4.6) Input text box for secret key assignment**

After entering the secret key, the process of converting it from a character string form to a binary form is performed. Then, the selected generator is initialized using the binary contents of secret key. After the initialization stage, HSS system becomes ready to

encrypt any selected HTML file. The encryption of any HTML file is started directly after choosing the HTML file name and its path by the common dialog control, which is shown in Figure (4.7).



**Figure (4.7) Menu of HTML files**

Figures (4.8) and (4.9) show a sample of an HTML file and its source code before encryption (i.e., the plain text). Figures (4.10) and (4.11) show the contents of the corresponding encrypted HTML file (i.e., ciphertext) and its source code after encryption by the proposed random number generator. While, Figure (4.12) and Figure (4.13) present the contents of encrypted HTML file by Linear Feedback Shift Register generator. In all above mentioned figures, the ciphered text lay within the ciphertext version of the HTML file, is encoded using base64 encoding scheme.

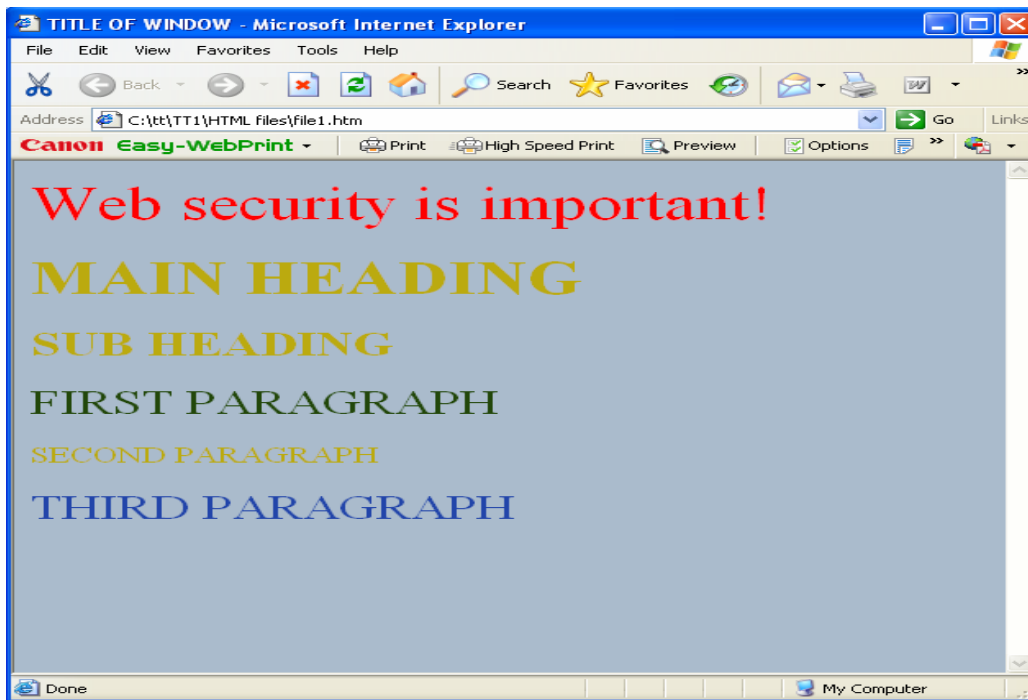


Figure (4.8) The contents of an HTML file sample (i.e., plaintext)

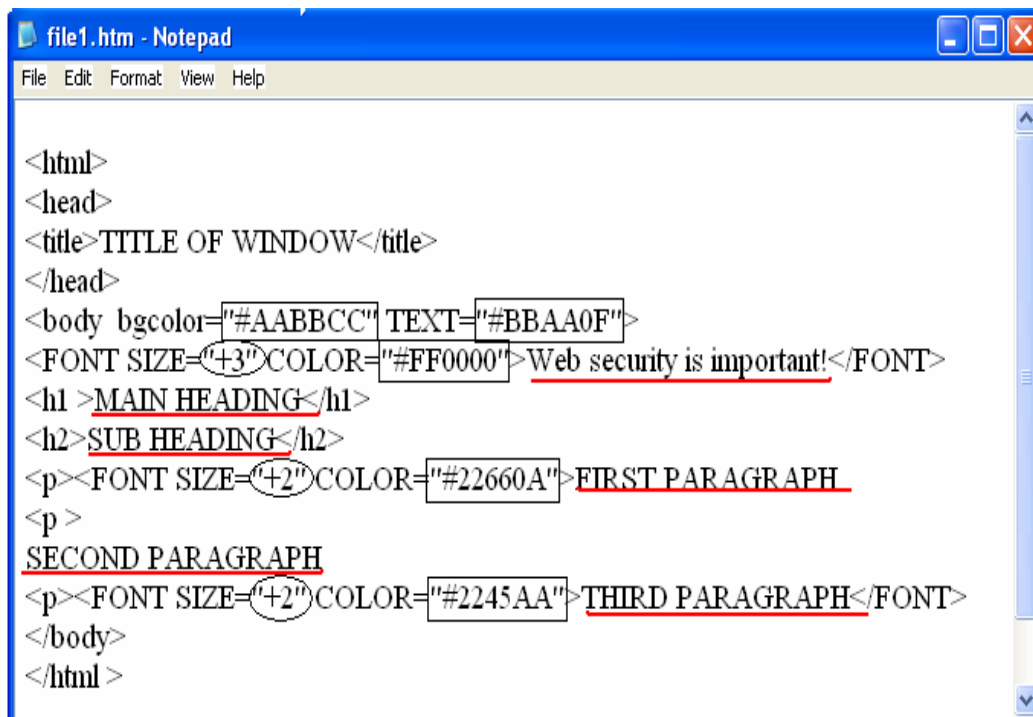


Figure (4.9) The source code of HTML file sample (i.e., plaintext)

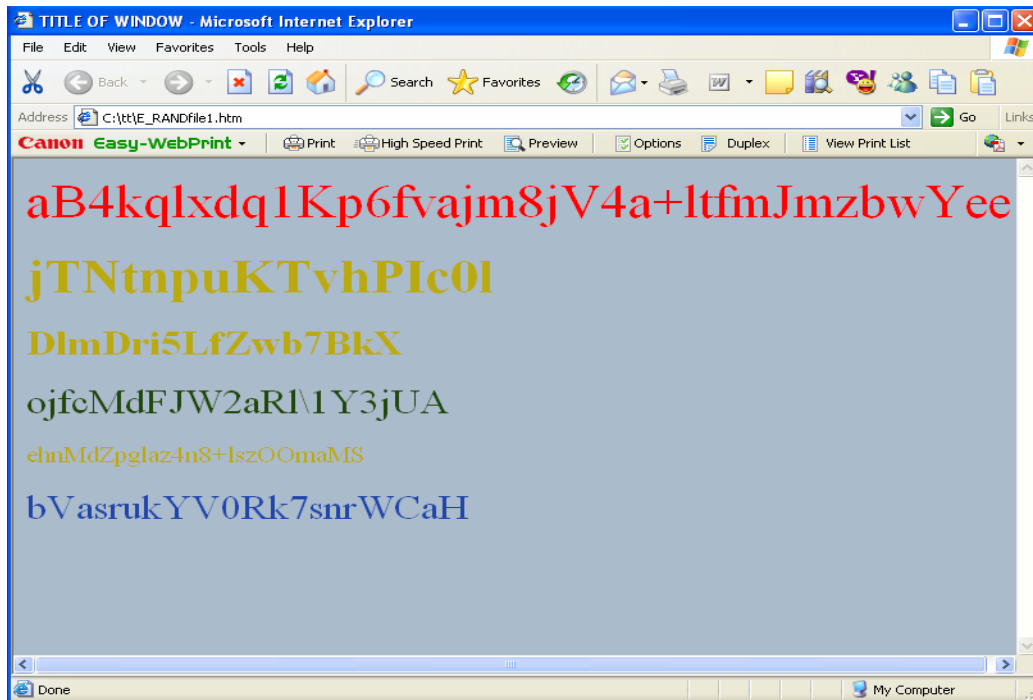


Figure (4.10) The contents of encrypted HTML file sample using the proposed random generator (i.e., ciphertext)

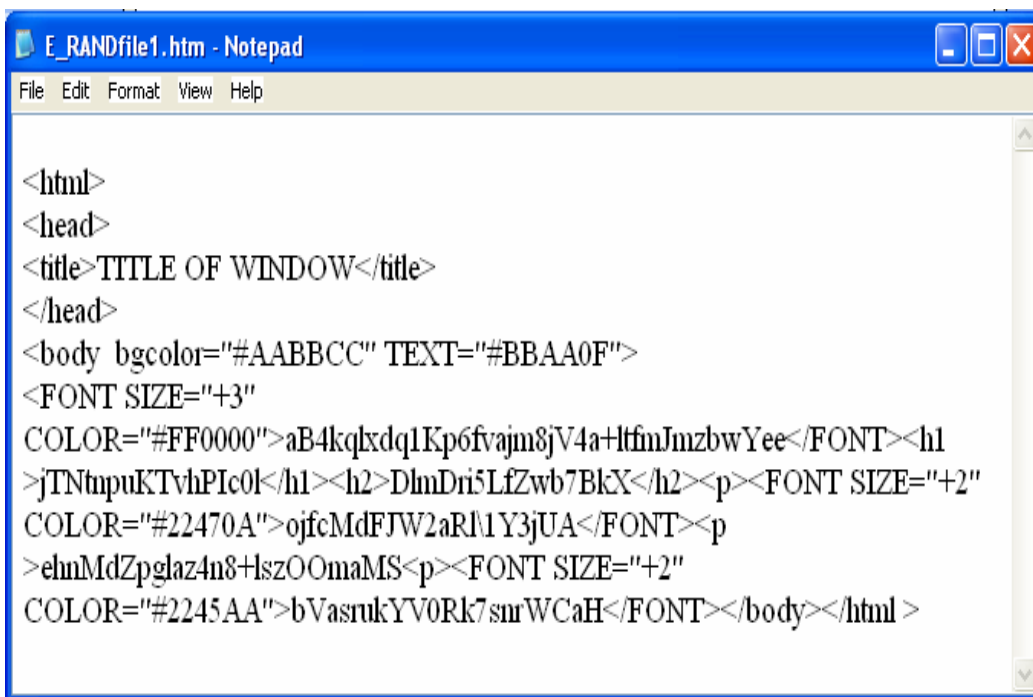
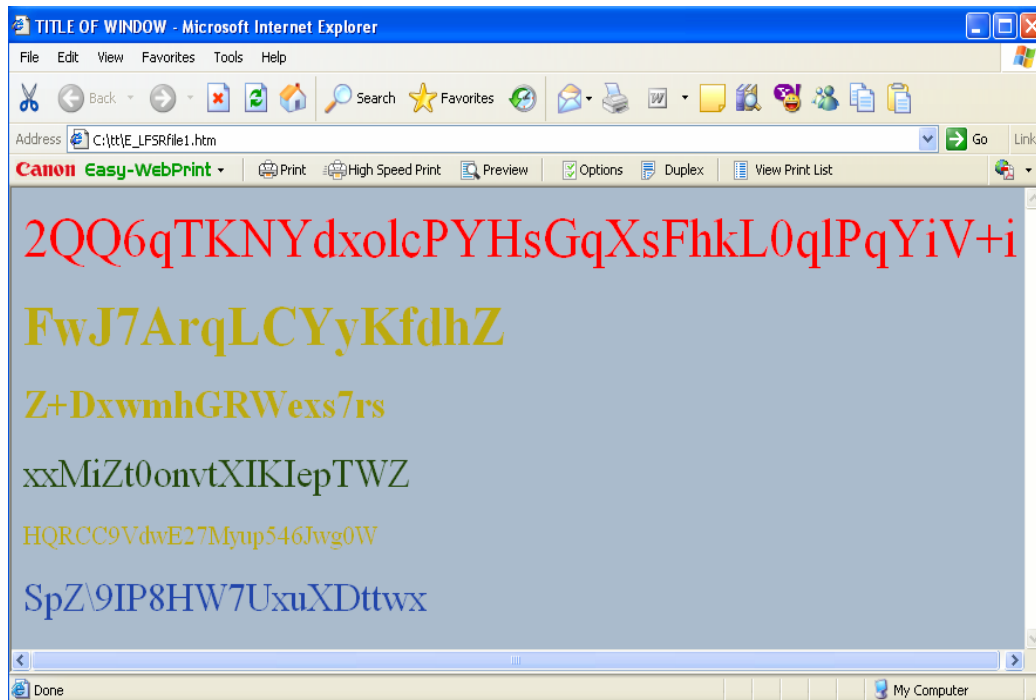
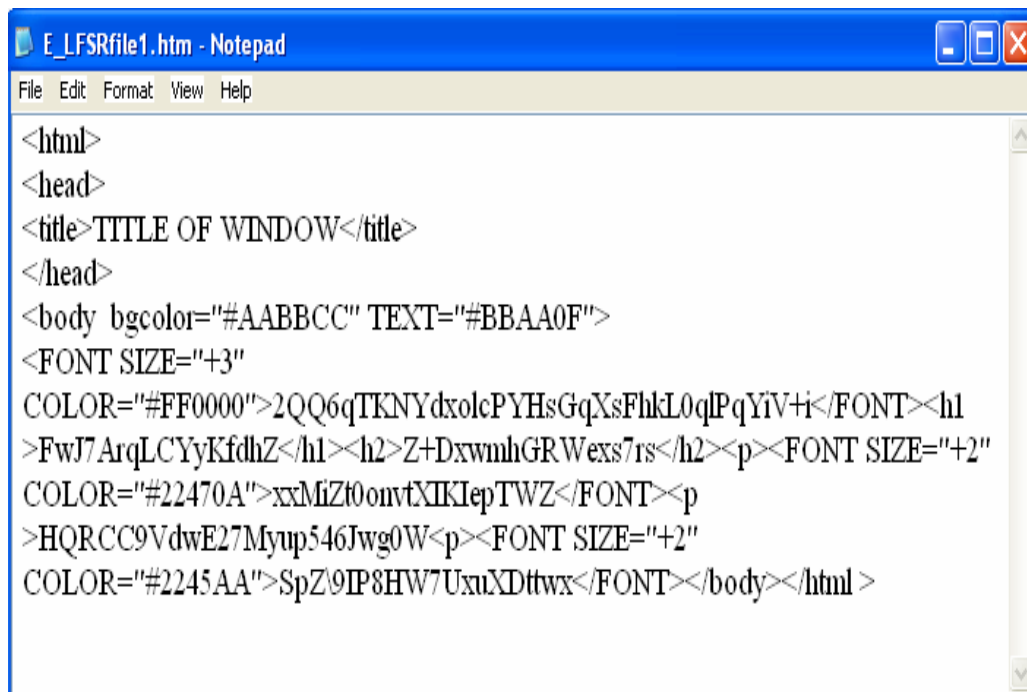


Figure (4.11) The source code of encrypted HTML file sample using the proposed random generator (i.e., ciphertext)



**Figure (4.12) The contents of encrypted HTML file sample using LFSR generator (i.e., ciphertext)**

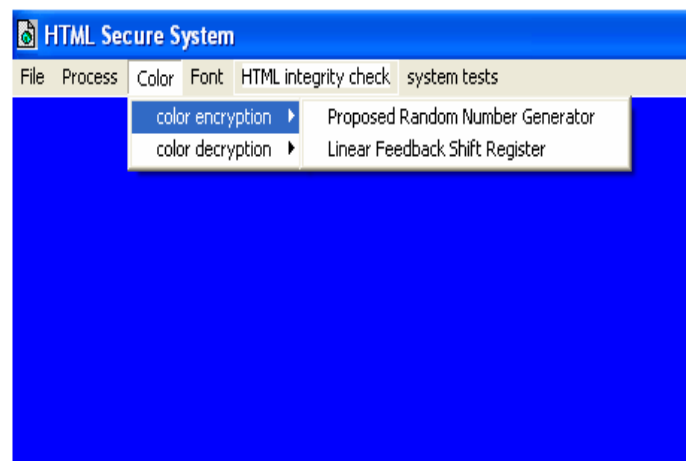


**Figure (4.13) The source code of encrypted HTML file sample using LFSR generator (i.e., ciphertext)**



**B. HTML File Decryption:** With this option, the user can follow same sequence of the steps performed with HTML file encryption, but with reversed order. The decryption process is performed using one of the two random number generators mentioned previously.

**3. Color:** this option is used to encrypt and decrypt the color attributes may found in the body section of HTML file. So, when choosing this option, a sub-menu appears on screen asking user to choose either **Color Encryption** or **Color Decryption**, as depicted in Figure (4.14). After choosing color encryption process and assigning the type of random generator, a text box appears asking the user to enter the secret key, then a common dialog box control is displayed to let the user choose the HTML file whose color is going to be encrypted. Figures (4.15) and (4.16) show the encrypted HTML file sample and its source code after encrypting its colors. Due to color decryption option, the original colors of the Webpage could be retrieved.



**Figure (4.14) The color sub menu**

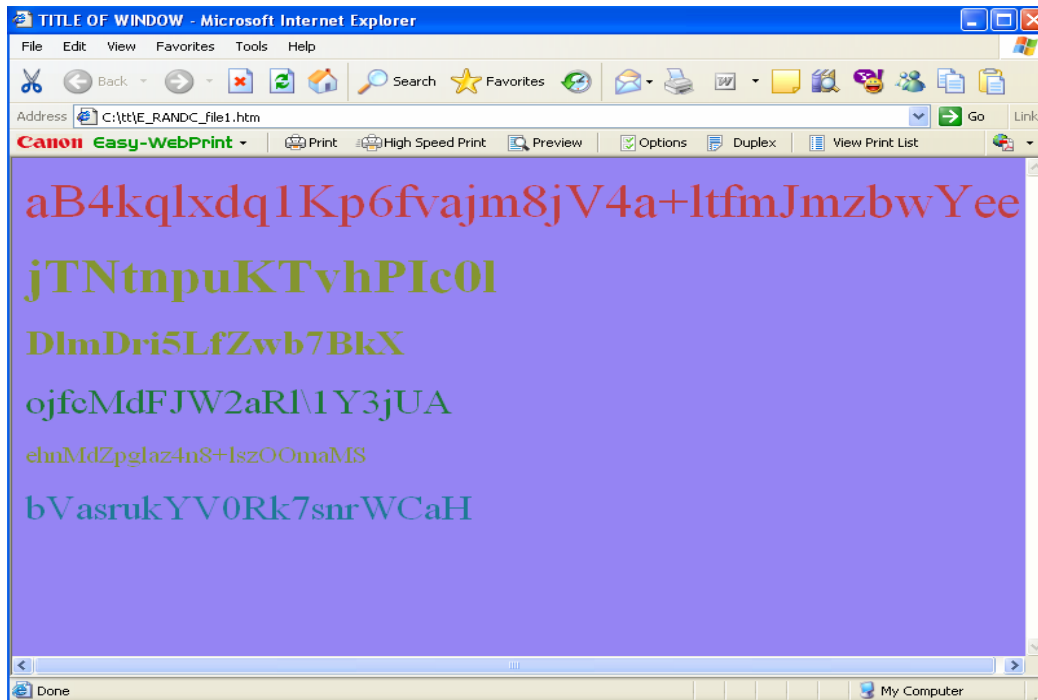


Figure (4.15) The encrypted HTML file sample after color encryption

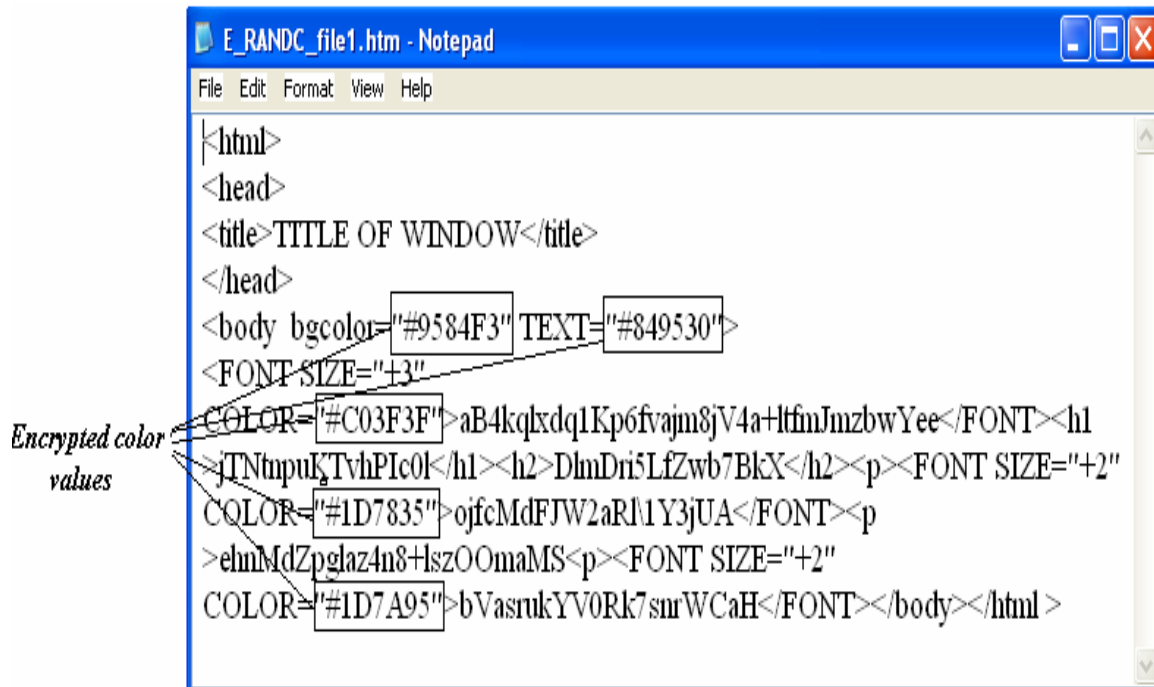


Figure (4.16) The source code of encrypted HTML file sample after color encryption

A comparison between the original content of the HTML file sample, shown in Figure (4.9), and its encrypted version shown in Figure (4.16), the original background color (bgcolor) value #AABBCC, shown in the first Figure, is encrypted to #9584F3. While the color of the whole text value #BBAA0F is encrypted to #849530. The colors used to define specific paragraphs like #FF0000, #22470A and #2245AA are encrypted to #C03F3F, #1D7835 and #1D7A95, respectively.

Another HTML file sample was taken such that it contains color attributes referred by names. As depicted in Figure (4.17), the background color of the HTML file is red, the color of the whole text is blue and the color of a specific paragraph is green. After color encryption, the ciphered color names become cyan, silver and white, respectively, as shown in Figure (4.18).

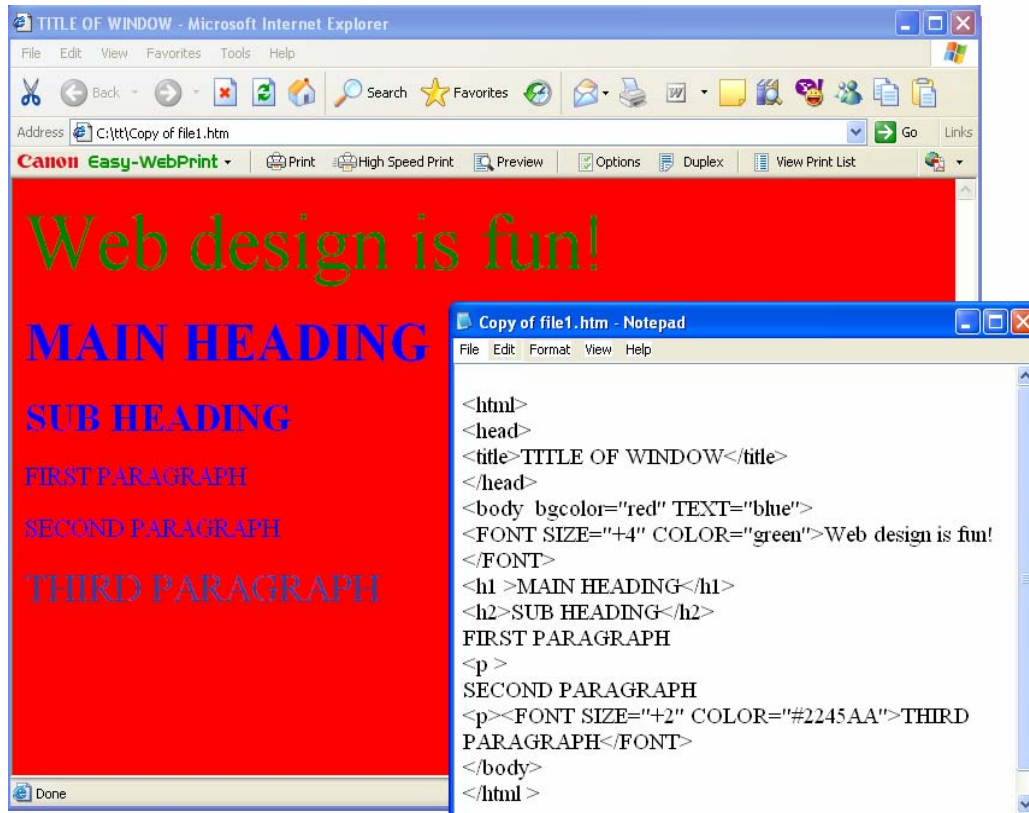
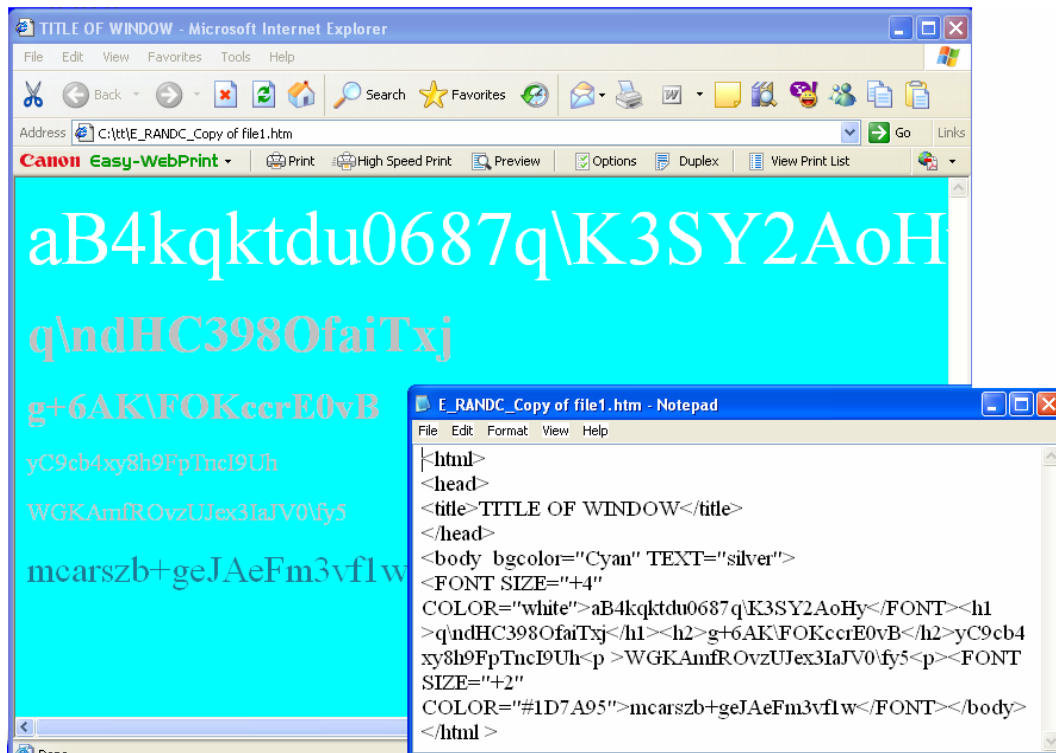


Figure (4.17) HTML file sample1



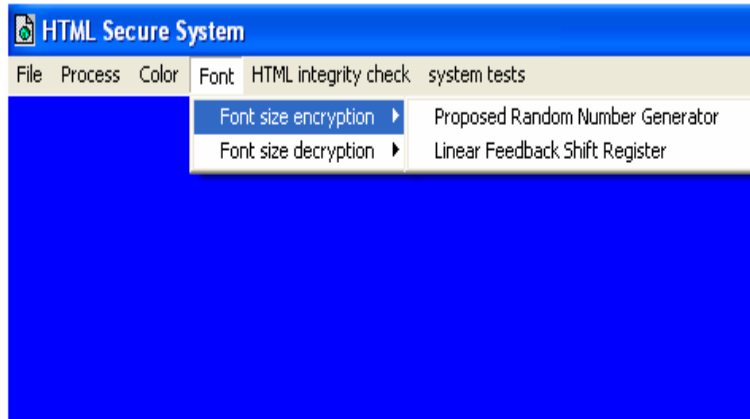


Figure (4.19) The font sub menu

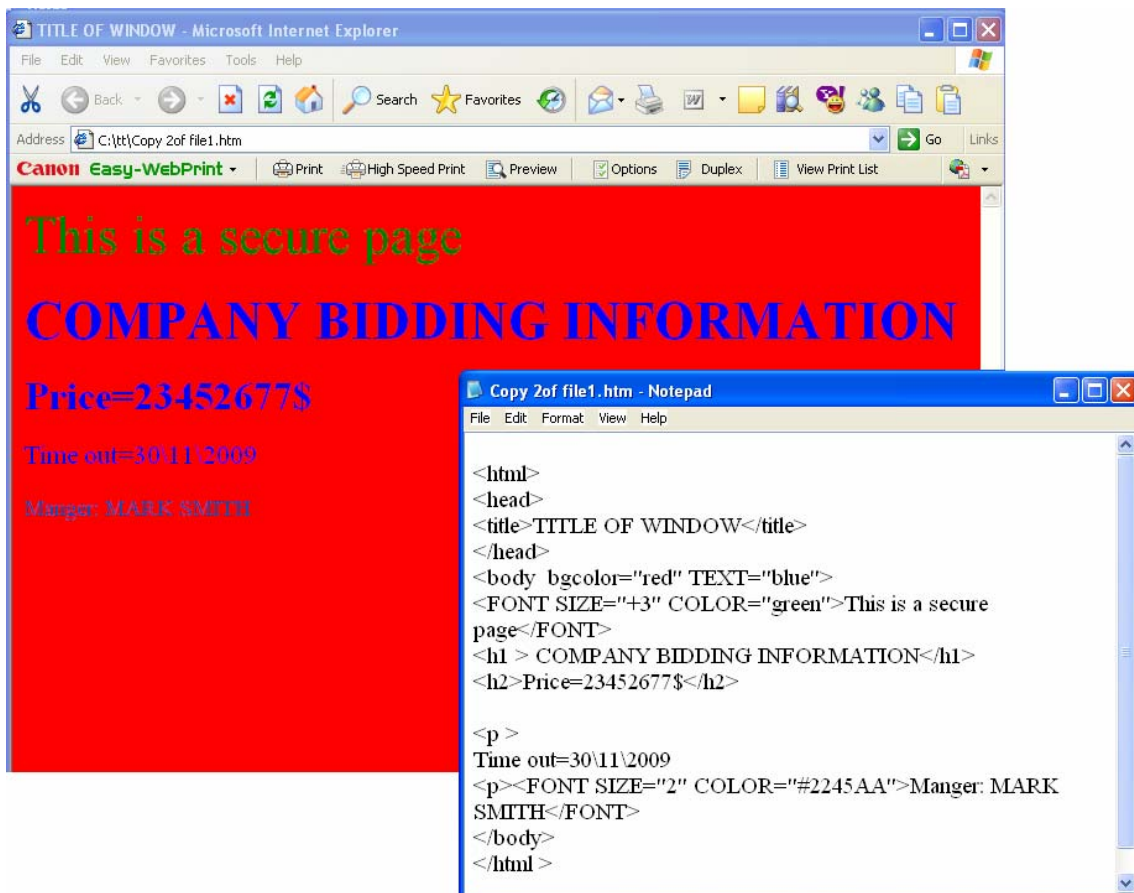


Figure (4.20) HTML file sample 2

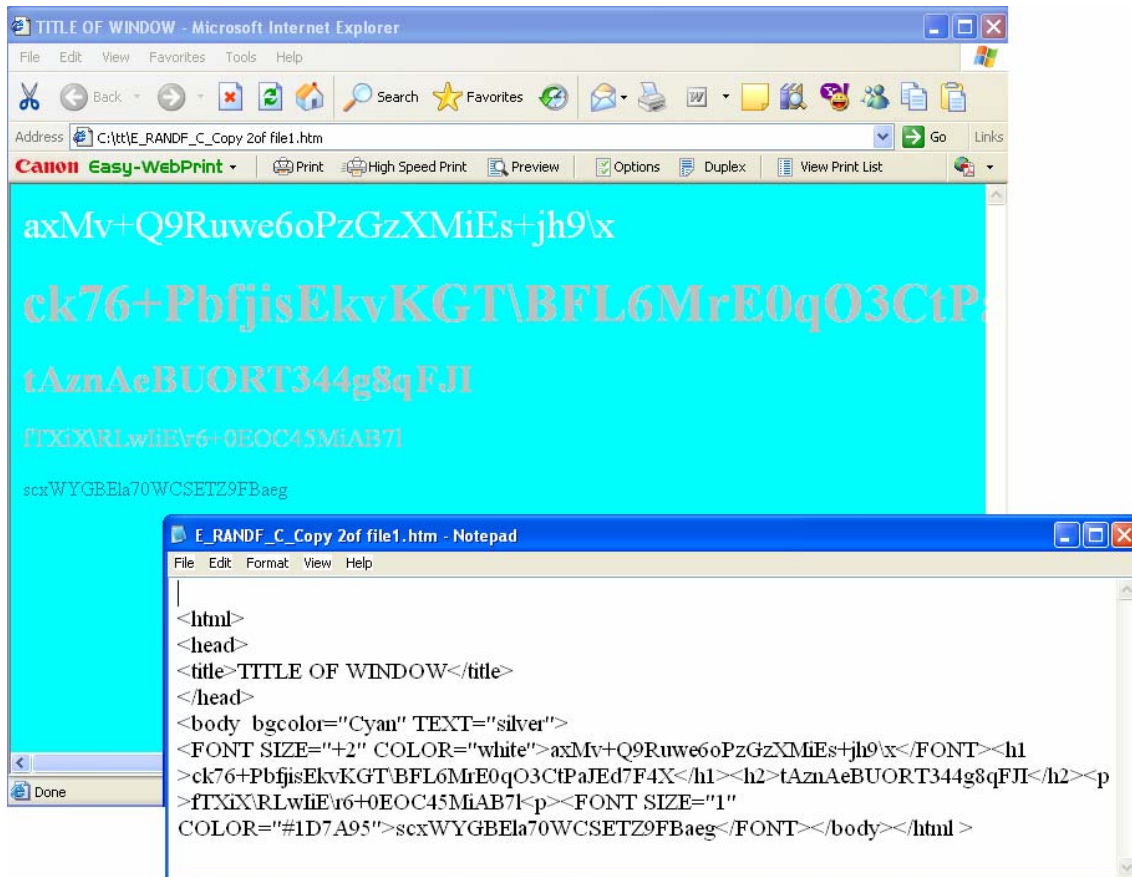


Figure (4.21) The encrypted HTML file (text, color, font size) encryption

5. **HTML Integrity Check:** this option is used by recipient (i.e., receiver) only, because the message digest of HTML file is calculated during the file encryption stage, and the value of the message digest is attached to the encrypted HTML file. When, a recipient tries to decrypt the HTML file, the integrity of the HTML file is checked by enabling the **HTML Integrity Check** option. The user should first enter the secret key, if the contents of encrypted HTML file was not modified and consequently its digest is correct, then a message box appears to inform the user that the tested HTML file is not corrupted during its transfer from server side to client side, otherwise a message box appears to acknowledge that the tested HTML file was corrupted during its transfer.

6. *System Test*: this option is used for evaluation purpose. The encrypted text could be evaluated using the five traditional statistical randomness tests.

### 4.3 System Evaluation

Many measures could be used to assess the performance of any developed security system. In this section, the evaluation results of HSS are presented, the evaluation is based on two kinds of metrics; they are:

- The randomness test measures applied on the produced ciphertext.
- The elapsed encoding time to perform the encryption and decryption processes.

#### 4.3.1 Randomness Tests

The randomness degree of a stream cipher system was investigated using the following statistical tests:

- a. Frequency test
- b. Serial test
- c. Poker test
- d. Run test
- e. Autocorrelation test

These five traditional randomness tests have been applied on the cipher text contained in the body section of the encrypted HTML file. In this kind of tests, different ciphertext subjects with different sizes were selected. The chosen ciphertext samples are produced using one of the applied random generators: (i.e., Linear Feedback Shift Register (LFSR) and the proposed random number generator).

The randomness tests program can assess the degree of randomness of the produced encrypted text found in each tested HTML file. The test

results could be shown as charts and tables for each HTML file. The table specifies whether each encrypted HTML file has passed or failed in each test of the five traditional randomness tests. In the remaining part of this chapter, the symbols in the incoming charts have the following meaning:

N represents the number of bits in the ciphered text.

N0 represents the number of zero's in the ciphered text.

N1 represents the number of one's in the ciphered text.

N00 represent every successive two zeros in the ciphered text.

N11 represent every successive two ones in the ciphered text.

N01 represent every successive pair of zero-one in the ciphered text.

N10 represent every successive pair of one-zero in the ciphered text.

The following six samples of encrypted HTML files are experimented:

### **HTML File1:**

1. Testing the ciphertext produced from file1.html using the proposed random number generator: Figure (4.22) shows the results of five randomness tests applied on the ciphertext version of HTML file1. Table (4.1) illustrates the corresponding evaluation result.

**Table (4.1) The randomness tests result for the ciphered HTML file1 using the (proposed random generator)**

	<i>Random tests</i>	<i>Value</i>	<i>Chi-square value</i>
<b>HTML File1 Size= (1KB)</b>	<i>Frequency test</i>	1.47 P	<=3.841
	<i>Serial test</i>	1.07 P	<=5.991
	<i>Poker test</i>	3 P	<=43.77
	<i>Run test</i>	20.9 P	<=26.3
	<i>Autocorrelation test</i>	1.6 P	<=3.841



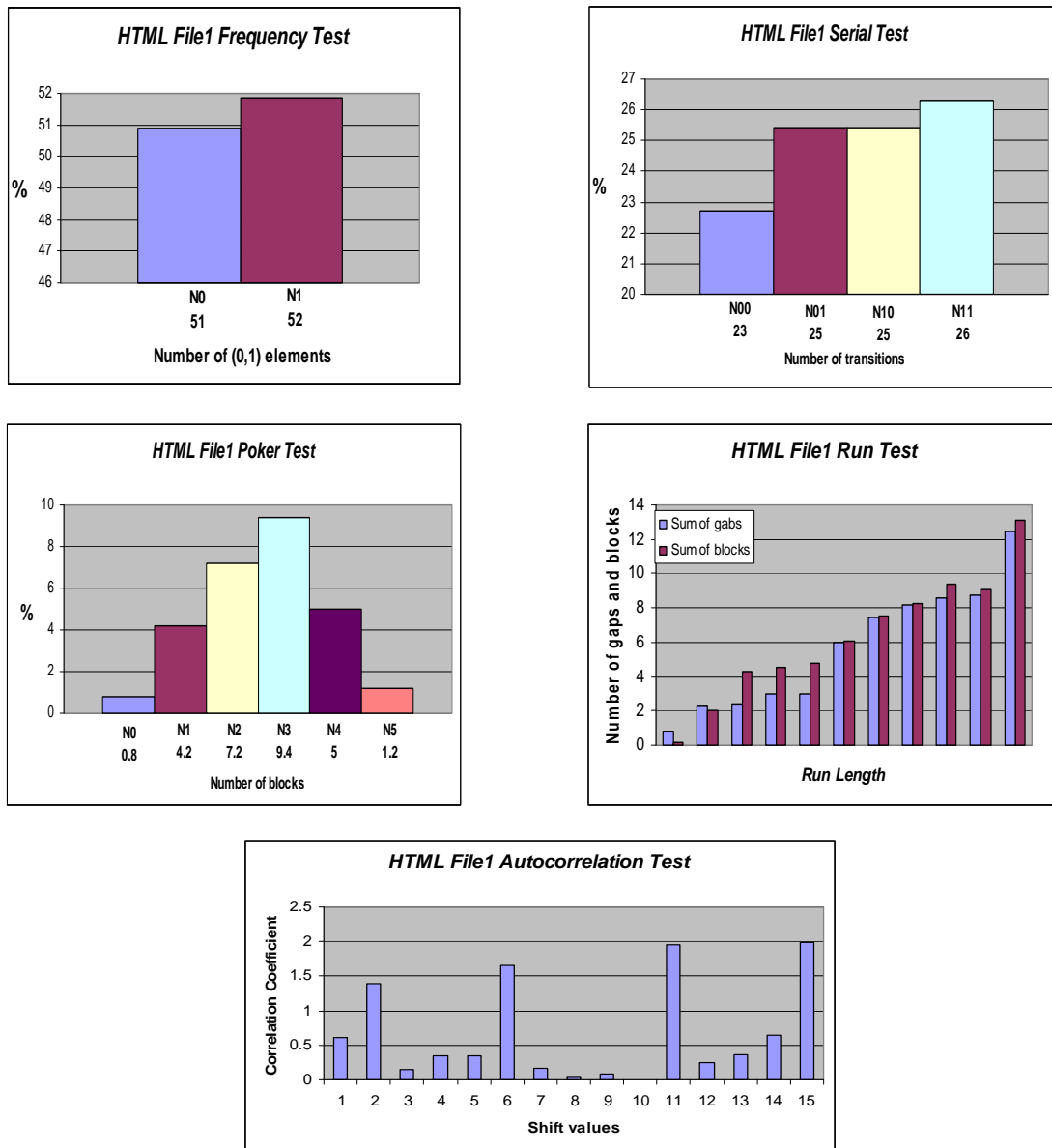
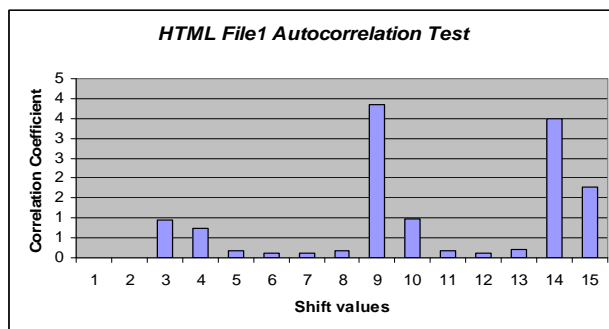
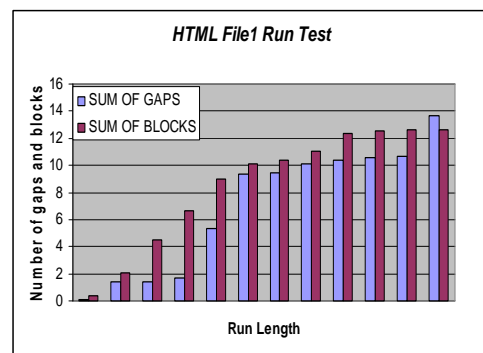
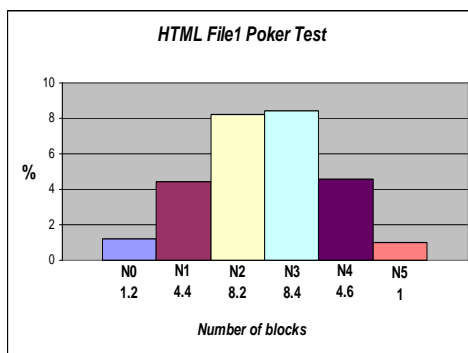
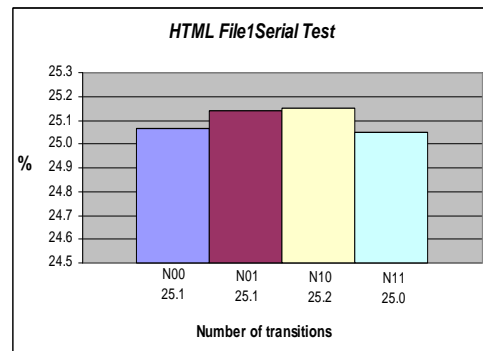
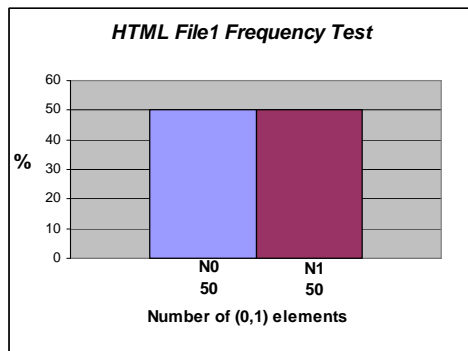


Figure (4.22) The results of five randomness tests for the ciphered HTML file1 using the (proposed random generator)

2. Testing the ciphertext produced from file1.html using Linear Feedback Shift Register generator: Figure (4.23) shows the results of five randomness tests applied on the ciphertext version of HTML file1, and Table (4.2) summarizes the evaluation results.

**Table (4.2) The randomness tests result for the ciphered HTML file1 using (LFSR random generator)**

<b>HTML File1</b> Size= (1KB)	<b>Random tests</b>	<b>Value</b>	<b>Chi-square value</b>
	<b>Frequency test</b>	0 P	$\leq 3.841$
	<b>Serial test</b>	0.04 P	$\leq 5.991$
	<b>Poker test</b>	1 P	$\leq 43.77$
	<b>Run test</b>	45.24 F	$\leq 33.29$
	<b>Autocorrelation test</b>	1.76 p	$\leq 3.841$



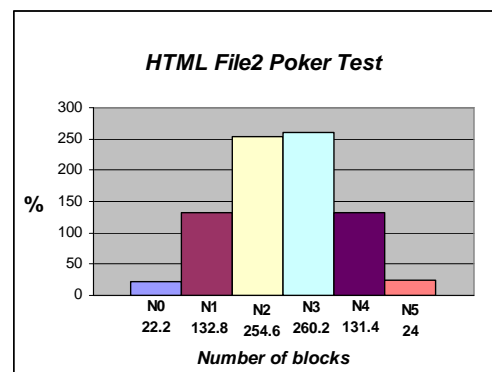
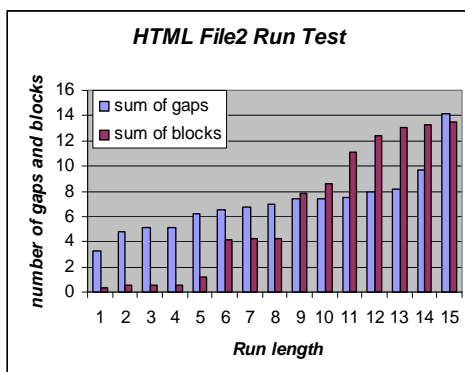
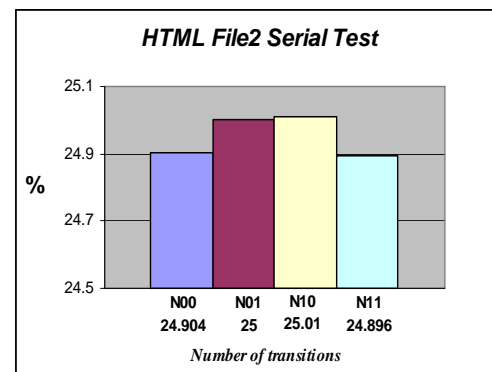
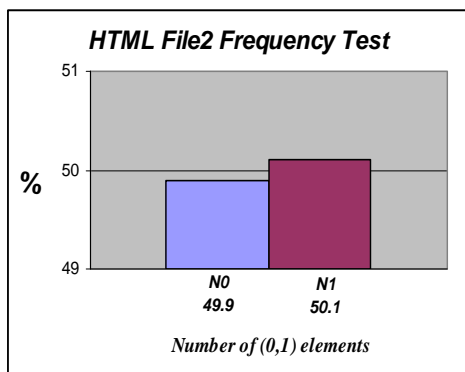
**Figure (4.23) The results of five randomness tests for the ciphered HTML file1 using (LFSR random generator)**

**HTML File 2:**

1. Testing the ciphertext produced from file2.html using the proposed random number generator: Figure (4.24) shows the results of the five randomness tests applied on the ciphertext version of HTML file2, and Table (4.3) summarizes the evaluation results.

**Table (4.3) The randomness tests for the ciphered HTML file2 using the (proposed random generator)**

	<i>Random tests</i>	<i>Value</i>	<i>Chi-square value</i>
<b>HTML File2 Size= (7.99KB)</b>	<i>Frequency test</i>	0.12 P	$\leq 3.841$
	<i>Serial test</i>	1.8 P	$\leq 5.991$
	<i>Poker test</i>	4 P	$\leq 43.77$
	<i>Run test</i>	27.6 P	$\leq 41.34$
	<i>Autocorrelation test</i>	0.2 P	$\leq 3.841$



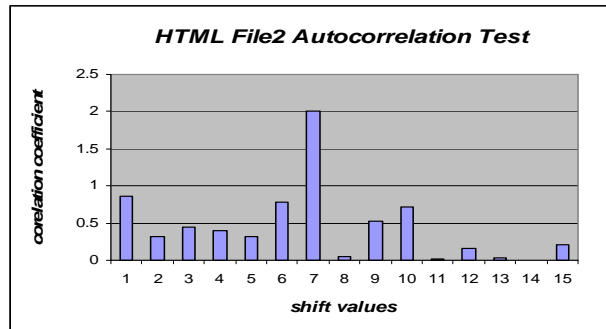
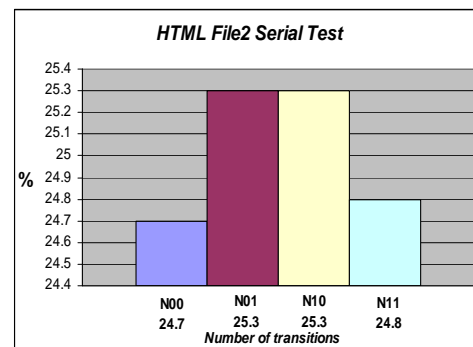
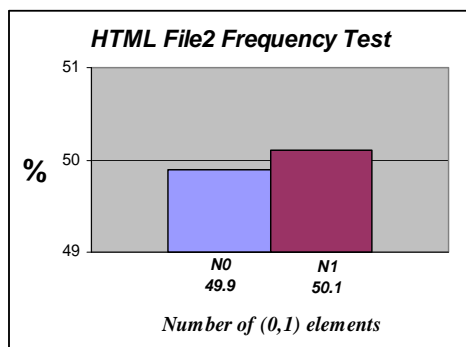


Figure (4.24) The results of the five randomness tests for the ciphered HTML file2 using the (proposed random generator)

2. Testing the ciphertext produced from file2.html using linear feedback shift register generator: Figure (4.25) shows the results of the five randomness tests applied on the ciphertext version of HTML file2, and Table (4.4) summarizes the evaluation results.

Table (4.4) The randomness tests for the ciphered HTML file2 using (LFSR random generator)

	<i>Random tests</i>	<i>Value</i>	<i>Chi-square value</i>
<b>HTML File2</b> Size= (7.99KB)	<i>Frequency test</i>	0.03 P	$\leq 3.841$
	<i>Serial test</i>	2.4 P	$\leq 5.991$
	<i>Poker test</i>	5 P	$\leq 43.77$
	<i>Run test</i>	63.9 F	$\leq 43.77$
	<i>Autocorrelation test</i>	0.26 P	$\leq 3.841$



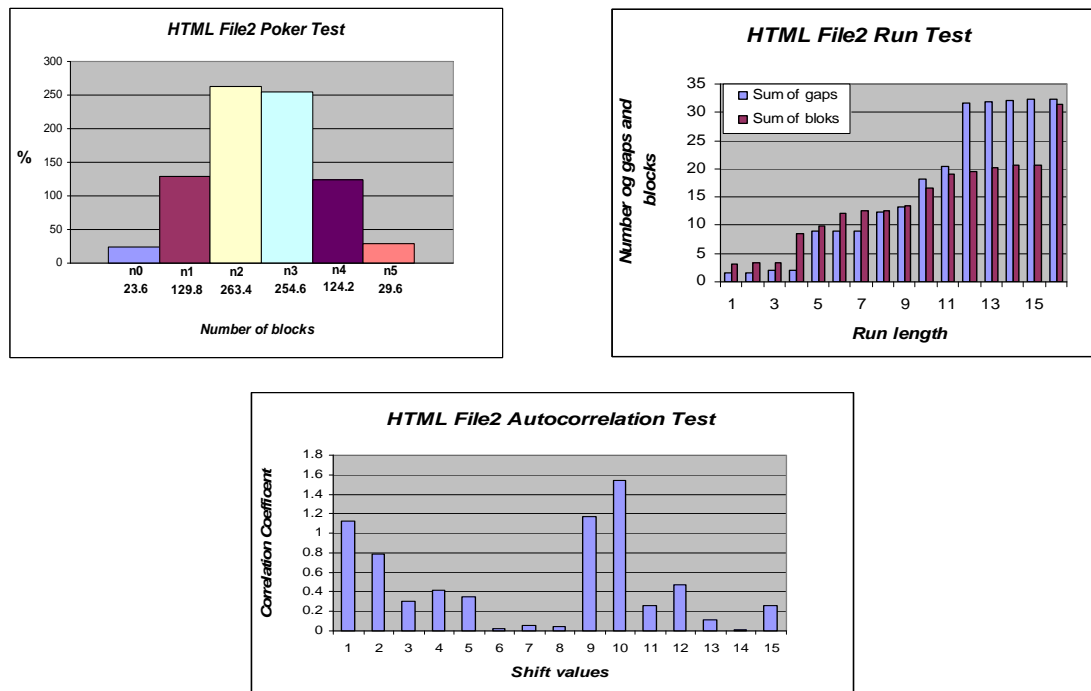


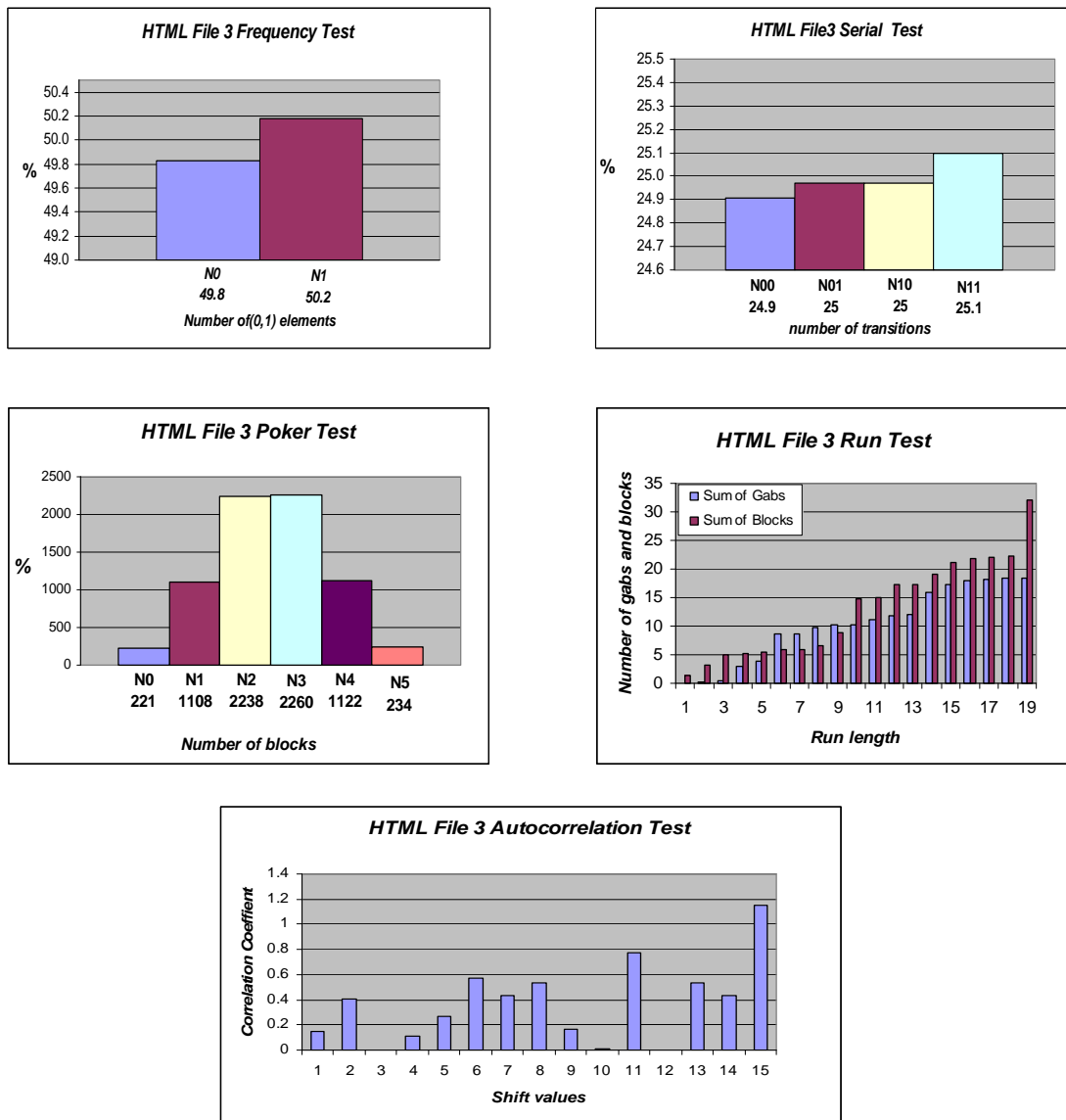
Figure (4.25) The results of the five randomness tests for the ciphered HTML file2 using (LFSR random generator)

**HTML File 3:**

1. Testing the ciphertext produced from file3.html using the proposed random number generator: Figure (4.26) shows the results of five randomness tests applied on the ciphertext version of HTML file3, and Table (4.5) summarizes the evaluation results.

Table (4.5) The randomness tests result for the ciphered HTML file3 using the (proposed random generator)

	<i>Random Tests</i>	<i>Value</i>	<i>Chi-square value</i>
<b>HTML File3</b> Size= (27KB)	<i>Frequency test</i>	2.2 P	<=3.841
	<i>Serial test</i>	2.5 P	<=5.991
	<i>Poker test</i>	4 P	<=43.77
	<i>Run test</i>	50.52 F	<=43.77
	<i>Autocorrelation test</i>	1.15 P	<=3.841

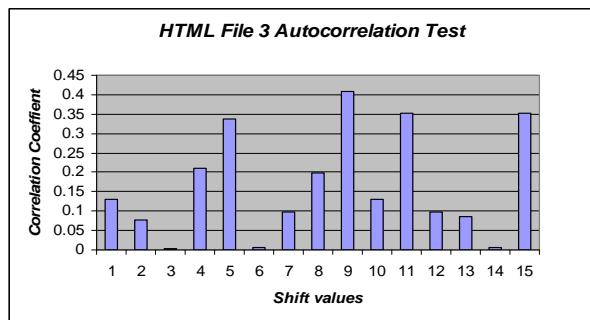
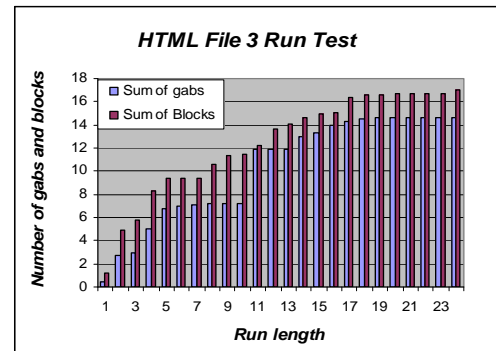
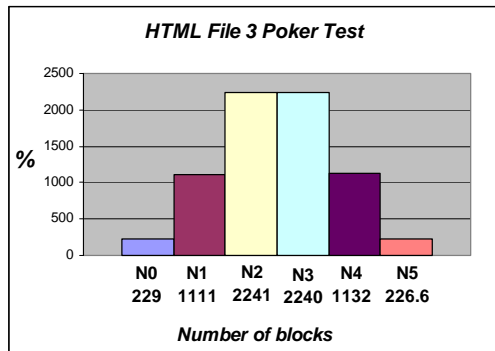
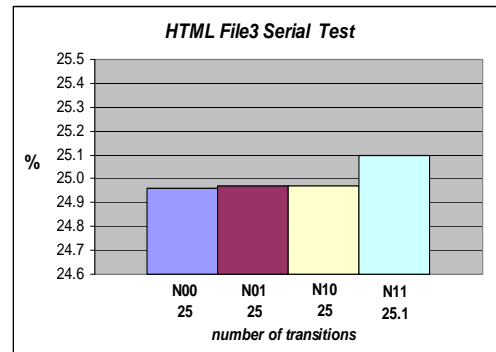
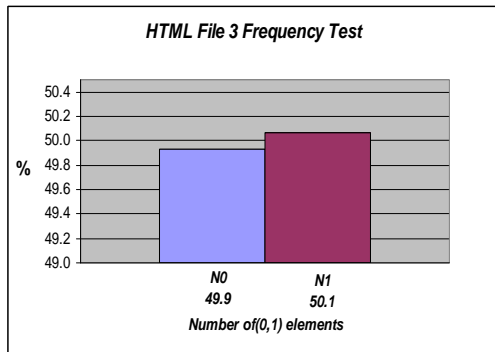


**Figure (4.26) The results of five randomness tests for the ciphered HTML file3 using the (proposed random generator)**

- Testing the ciphertext produced from file3.html using linear feedback shift register generator: Figure (4.27) shows the results of five randomness tests applied on the ciphertext version of HTML file3, and Table (4.6) summarizes the evaluation results.

**Table (4.6) The randomness tests result for the ciphered HTML file3 using (LFSR random generator)**

	<i>Random tests</i>	<i>Value</i>	<i>Chi-square value</i>
<b>HTML File3</b> Size= (27KB)	<i>Frequency test</i>	0.4 P	$\leq 3.841$
	<i>Serial test</i>	0.6 P	$\leq 5.991$
	<i>Poker test</i>	2 P	$\leq 43.77$
	<i>Run test</i>	403.15 F	$\leq 55.76$
	<i>Autocorrelation test</i>	0.35 P	$\leq 3.841$



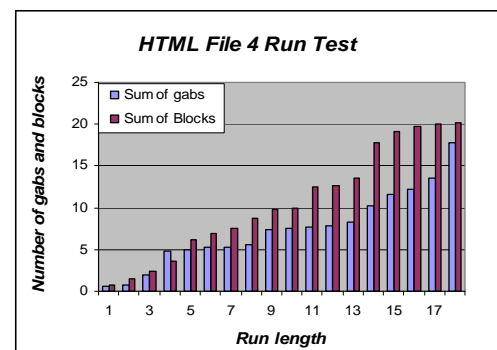
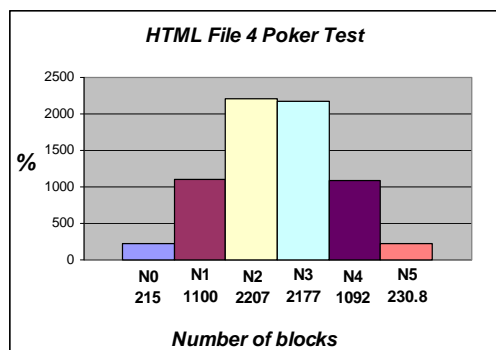
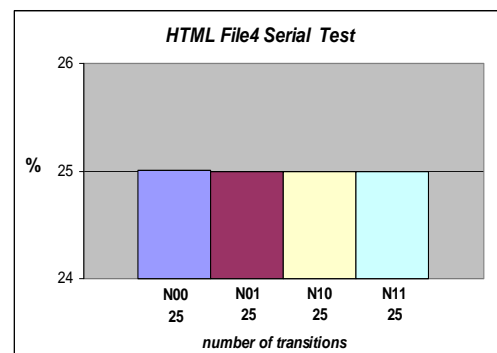
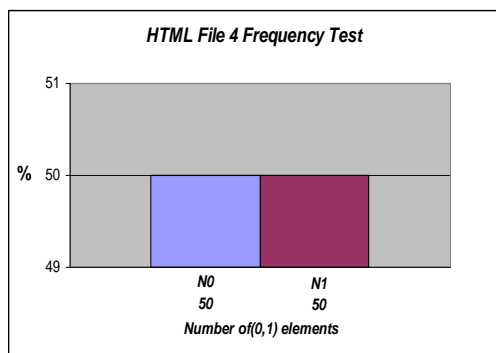
**Figure (4.27) The results of five randomness tests for the ciphered HTML file3 (using LFSR random generator)**

**HTML File 4:**

1. Testing the ciphertext produced from file4.html using the proposed random number generator: Figure (4.28) shows the results of the five randomness tests on the ciphertext version of HTML file4, and Table (4.7) summarizes the evaluation results.

**Table (4.7) The randomness tests for the ciphered HTML file4 using the (proposed random generator)**

<b>HTML File4 Size= (31.4KB)</b>	<i>Random tests</i>	<i>Value</i>	<i>Chi-square value</i>
	<i>Frequency test</i>	0.09 P	$\leq 3.841$
	<i>Serial test</i>	0.5 P	$\leq 5.991$
	<i>Poker test</i>	5 P	$\leq 43.77$
	<i>Run test</i>	37.9 P	$\leq 43.77$
	<i>Autocorrelation test</i>	0.36 P	$\leq 3.841$





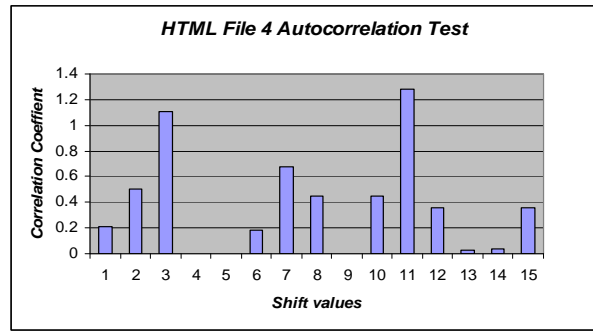
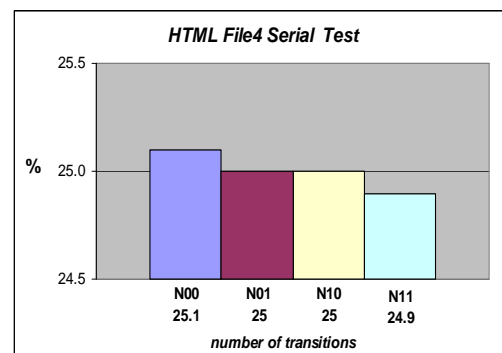
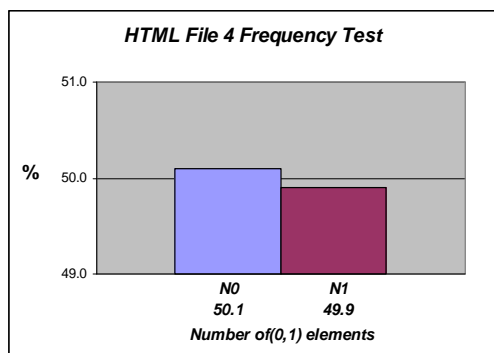


Figure (4.28) The result of five randomness tests for the ciphered HTML file4 using the (proposed random generator)

2. Testing the ciphertext produced from file4.html using linear feedback shift register generator: Figure (4.29) shows the results of the five randomness tests applied on the ciphertext version of HTML file4, and Table (4.8) summarizes the evaluation results.

Table (4.8) The randomness tests for the ciphertext of HTML file4 using (LFSR random generator)

	Random tests	Value	Chi-square value
<b>HTML File4</b> Size= (31.4KB)	Frequency test	0.71 P	$\leq 3.841$
	Serial test	0.7 P	$\leq 5.991$
	Poker test	3 P	$\leq 43.77$
	Run test	14.91 P	$\leq 43.77$
	Autocorrelation test	0.05 P	$\leq 3.841$



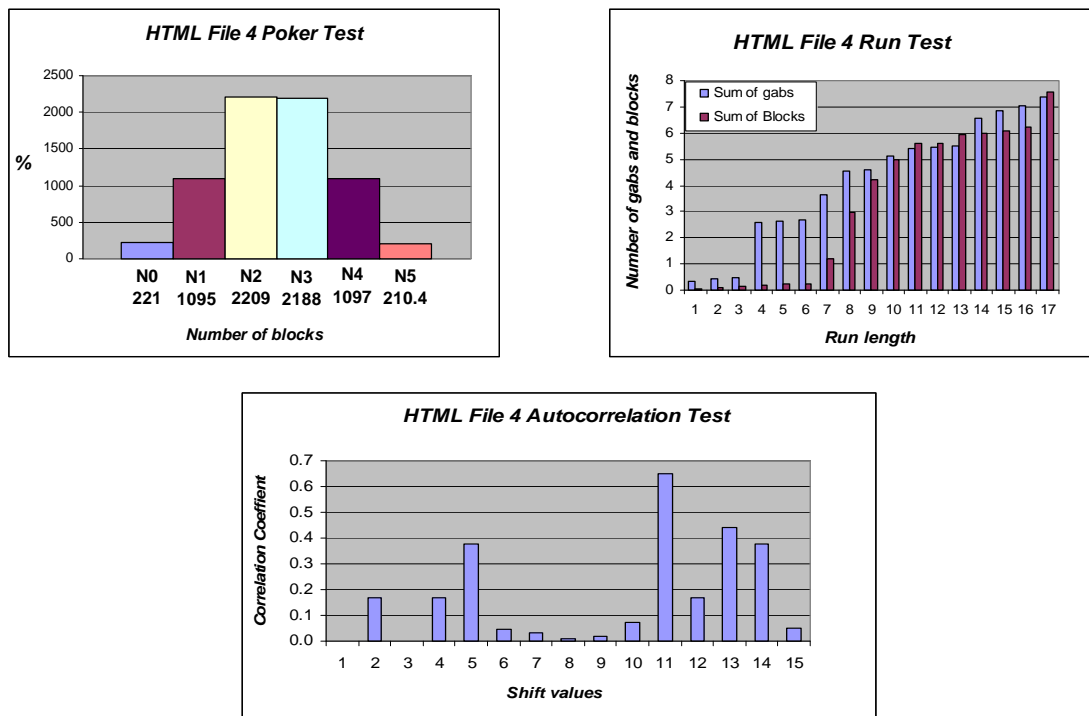


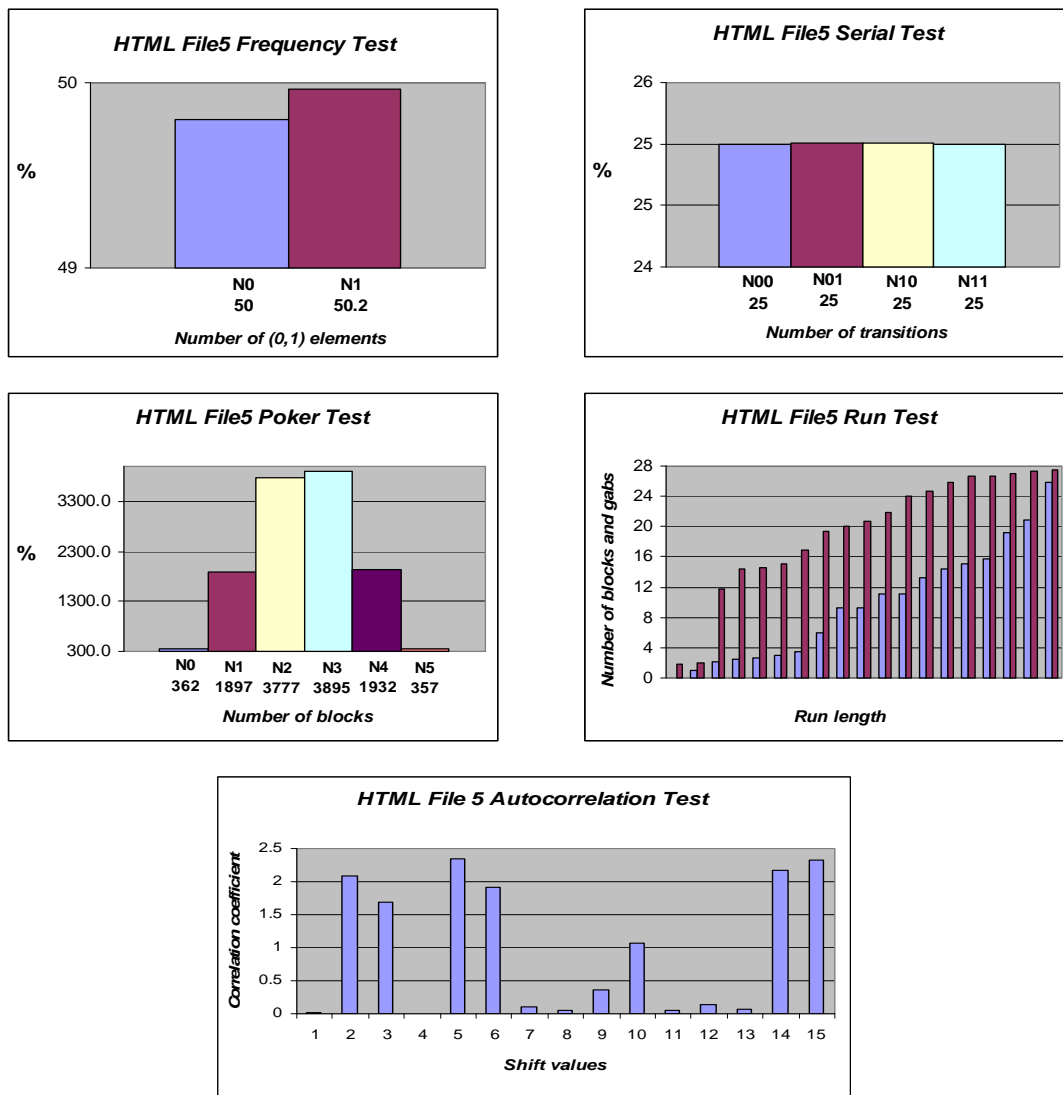
Figure (4.29) The results of five randomness tests for the ciphered HTML file4 using (LFSR random generator)

### HTML File 5:

1. Testing the ciphertext produced from file5.html using the proposed random number generator: Figure (4.30) shows the results of five randomness tests applied on the ciphertext version of HTML file5, and Table (4.9) summarizes the evaluation results.

Table (4.9) The randomness tests for the ciphertext of HTML file5 using the (proposed random generator)

	<i>Random tests</i>	<i>Value</i>	<i>Chi-square value</i>
<b>HTML File5</b> Size= (42.9KB)	<i>Frequency test</i>	3.3 P	$\leq 3.841$
	<i>Serial test</i>	3.4 P	$\leq 5.991$
	<i>Poker test</i>	25 P	$\leq 43.77$
	<i>Run test</i>	35.28 P	$\leq 43.77$
	<i>Autocorrelation test</i>	2.34 P	$\leq 3.841$

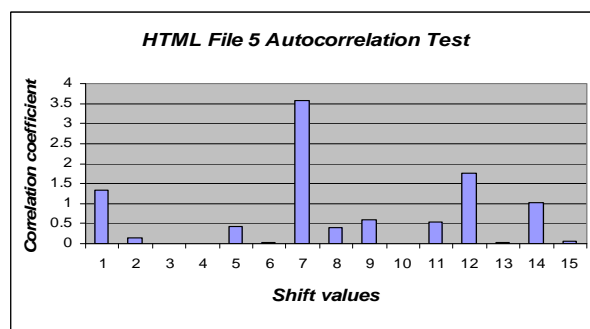
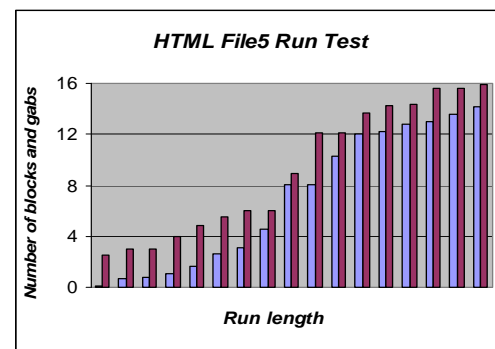
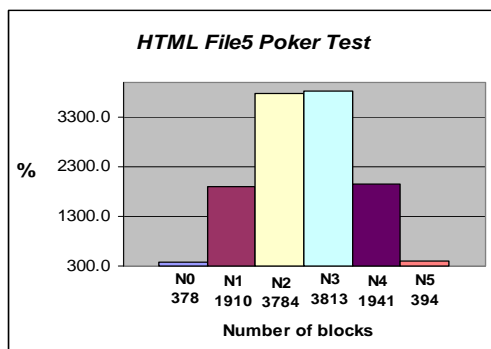
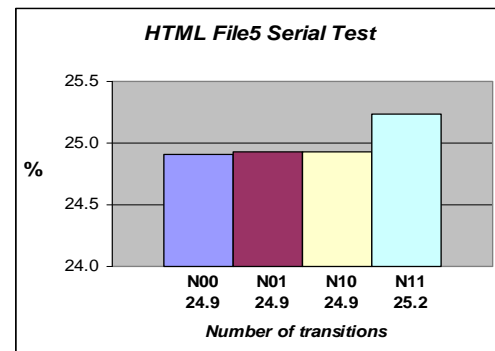
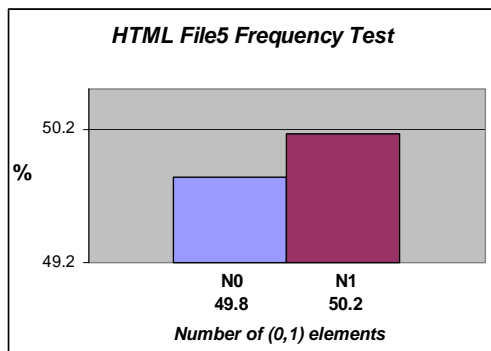


**Figure (4.30) The results of five randomness tests for the ciphertext of HTML file5 using the (proposed random generator)**

- Testing the ciphertext produced from file5.html using linear feedback shift register generator: Figure (4.31) shows the results of the five randomness tests applied on the ciphertext version of HTML file5, and Table (4.10) summarizes the evaluation results.

**Table (4.10) The randomness tests result for the ciphered HTML file5 using (LFSR random generator)**

<b>HTML File5 Size= (42.9KB)</b>	<b>Random tests</b>	<b>Value</b>	<b>Chi-square value</b>
	<b>Frequency test</b>	3.3 P	$\leq 3.841$
	<b>Serial test</b>	5.8 P	$\leq 5.991$
	<b>Poker test</b>	6 P	$\leq 43.77$
	<b>Run test</b>	30.1 P	$\leq 43.77$
	<b>Autocorrelation test</b>	0.05 P	$\leq 3.841$



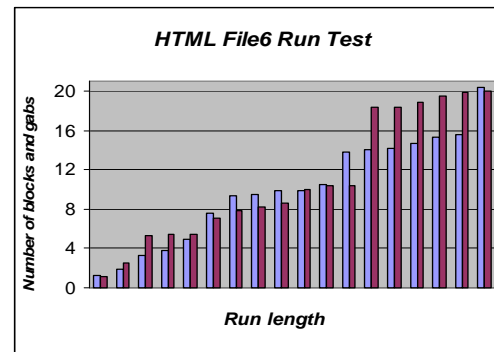
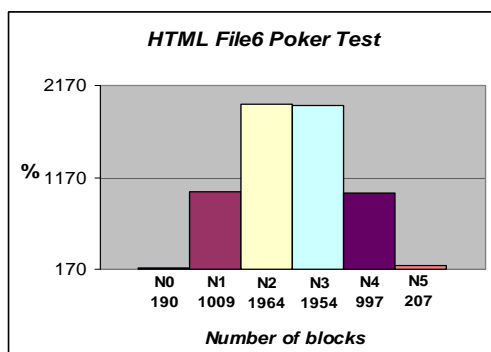
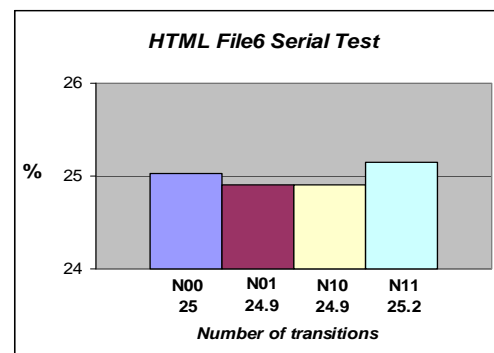
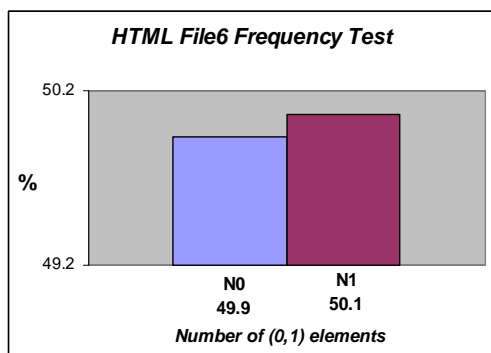
**Figure (4.31) The results of five randomness tests for the ciphered HTML file5 using (LFSR random generator)**

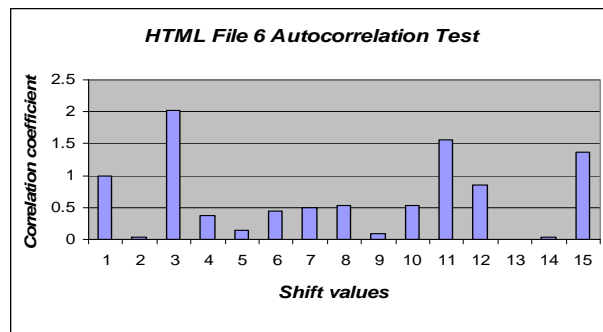
**HTML File 6:**

1. Testing the ciphertext produced from file6.html using the proposed random number generator: Figure (4.32) shows the results of the five randomness tests applied on the ciphertext version of HTML file6, and Table (4.11) summarizes the evaluation results.

**Table (4.11) The randomness tests for the ciphered HTML file6 using the (proposed random generator)**

<b>HTML File6 Size= (66.5 KB)</b>	<b>Random tests</b>	<b>Value</b>	<b>Chi-square value</b>
	<b>Frequency test</b>	0.3 P	$\leq 3.841$
	<b>Serial test</b>	2.2 P	$\leq 5.991$
	<b>Poker test</b>	8 P	$\leq 43.77$
	<b>Run test</b>	40.29 P	$\leq 43.77$
	<b>Autocorrelation test</b>	1.3 P	$\leq 3.841$



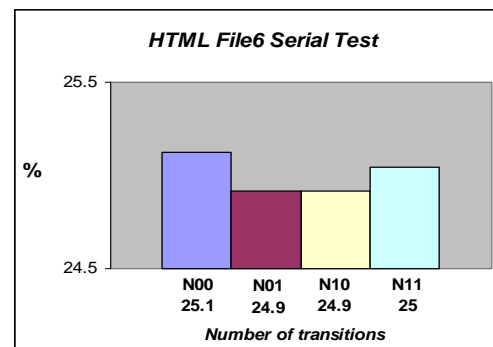
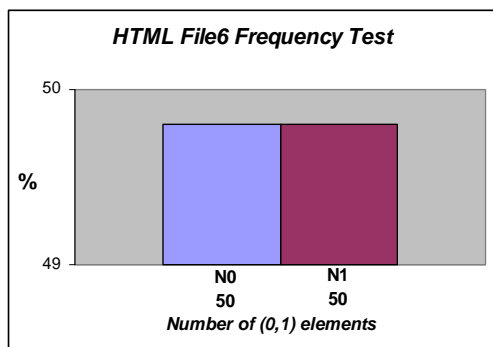


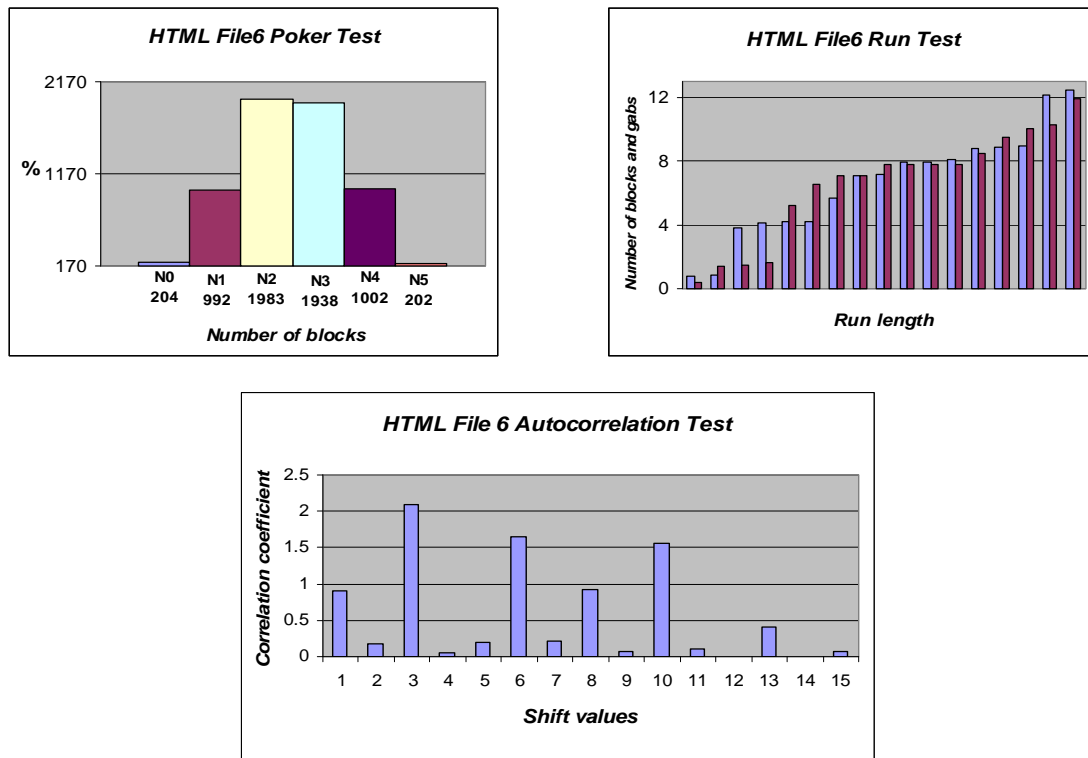
**Figure (4.32) The results of the five randomness tests for the ciphered HTML file6 using the (proposed random generator)**

2. Testing the ciphertext produced from file6.html using linear feedback shift register generator. Figure (4.33) shows the results of five randomness tests applied on the ciphertext version of HTML file6, and Table (4.12) summarizes the evaluation results.

**Table (4.12) The randomness tests for the ciphered HTML file6 using (LFSR random generator)**

	<i>Random tests</i>	<i>Value</i>	<i>Chi-square value</i>
<b>HTML File6</b> Size= (66.5 KB)	<i>Frequency test</i>	0.11 P	$\leq 3.841$
	<i>Serial test</i>	1.98 P	$\leq 5.991$
	<i>Poker test</i>	7 P	$\leq 43.77$
	<i>Run test</i>	24.4 P	$\leq 43.77$
	<i>Autocorrelation test</i>	0.08 P	$\leq 3.841$





**Figure (4.33) The results of five randomness tests for the ciphered HTML file6 using (LFSR random generator)**

The results of the experimental tests of randomness can be summarized that, after applying the five random tests, the results show that, the proposed random number generator and the linear feedback shift register have sufficient degree of randomness.

### 4.3.2 The Elapsed Encoding Time

In this section, the elapsed encryption time by HSS is investigated. The elapsed time was measured by applying ciphering and deciphering processes on different HTML files having different sizes.

In HSS encoding stage, the total time is computed as the summation of the (1) the time taken for key generation, (2) the time taken for encryption process, (3) the time taken for bas64 encoding, (4) the time taken for color encryption and (5) the time taken for font size encryption.

The time for key generation is measured for each one of the two applied random number generators (i.e., LFSR and the proposed random number generator). Figure (4.34) shows the variation of measured encoding time for five HTML files.

In HSS decoding stage, the total time is computed by summing the elapsed times of all stages of HSS decoder. Figure (4.35) shows the elapsed decoding time for five HTML files using two random number generators. Table (4.13) illustrates the encoding and decoding time with different HTML file sizes.

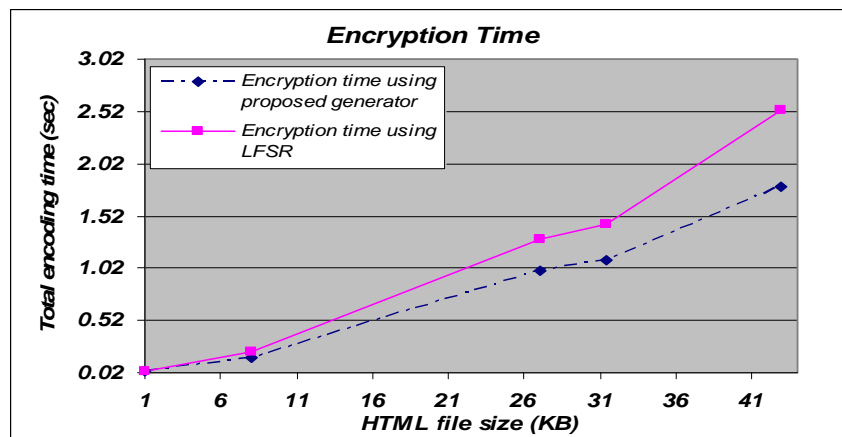


Figure (4.34) The variation of elapsed encoding time versus HTML file size

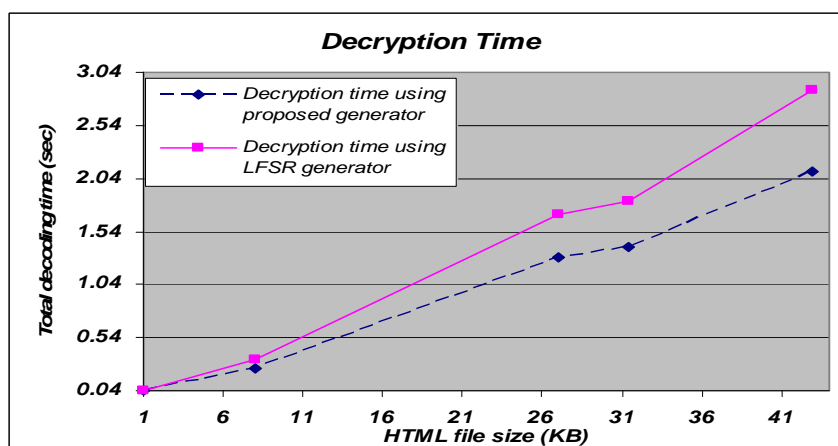


Figure (4.35) The variation of elapsed decoding time versus HTML file size



**Table (4.13) The elapsed encoding and decoding time values when using one of the two generators**

<i>HTML File Size (KB)</i>	<i>Encryption Time (MSec)</i>			<i>Decryption Time (MSec)</i>		
	<i>Proposed Generator</i>	<i>LFSR Generator</i>	<i>The difference in Time (MSec)</i>	<i>Proposed Generator</i>	<i>LFSR Generator</i>	<i>The difference in Time (MSec)</i>
<i>1</i>	<i>0.03</i>	<i>0.03</i>	<i>Zero(MSec)</i>	<i>0.04</i>	<i>0.04</i>	<i>Zero(MSec)</i>
<i>7.99</i>	<i>0.17</i>	<i>0.22</i>	<i>30%</i>	<i>0.25</i>	<i>0.33</i>	<i>30%</i>
<i>27</i>	<i>1</i>	<i>1.3</i>	<i>30%</i>	<i>1.3</i>	<i>1.7</i>	<i>30%</i>
<i>31.4</i>	<i>1.1</i>	<i>1.44</i>	<i>30%</i>	<i>1.4</i>	<i>1.83</i>	<i>30%</i>
<i>42.9</i>	<i>1.8</i>	<i>2.53</i>	<i>40%</i>	<i>2.1</i>	<i>2.94</i>	<i>40%</i>
<i>66.5</i>	<i>1.2</i>	<i>1.69</i>	<i>40%</i>	<i>1.8</i>	<i>2.54</i>	<i>40%</i>
<i>274</i>	<i>6.7</i>	<i>9.4</i>	<i>40%</i>	<i>9.5</i>	<i>13.3</i>	<i>40%</i>

The test results of the elapsed encoding and decoding time can be summarized as follows:

1. From the results in Figure (4.34) and Figure (4.35), the elapsed encoding and decoding time using the proposed random number generator is less than the required time using linear feedback shift register generator. Therefore, the proposed generator is faster.
2. From the results in Table (4.13), the total difference between encoding and decoding time when using proposed random number generator or linear feedback shift register is increased when the size of HTML file increases.
3. The elapsed time taken for color and font encryption shown in Figures (4.16) and (4.21) is about (1 milliseconds to 1.5 milliseconds) of the total encryption time, and the elapsed time for color and font decryption has nearly the same percentage.

4. The difference in encoding time values of the two HSS variants (i.e., using LFSR or proposed random generators) increases when the size of the HTML files is increased. As illustrated in Figure (4.34), the difference between the two encryption times increases when choosing large HTML files. In some cases, it was found that when the size of HTML files increases, the encoding time decreases. This is because some HTML files contain images, videos more than text data. See sixth raw in Table (4.13).

# Chapter Five

## Conclusions and Future Works

### 5.1 Introduction

Through the discussion presented in previous chapter, some remarks related to the performance behavior of the introduced HSS scheme have given. In this chapter a list of conclusions are presented and some recommendations for further work are given, they could contribute in increasing the level of HTML security.

### 5.2 Conclusions

A summary of some derived conclusions are given in the following:

1. The results of the conducted tests indicated that the proposed scheme (HSS) can satisfy the two security requirements (i.e., confidentiality and integrity). Also, HSS can secure any HTML file without changing its structure, and any Internet Explorer can interpret and display the contents of secured HTML file as an encrypted Webpage and the hosting servers can transmit the Webpage via HTTP (Hyper Text Transfer Protocol) media.
2. Two optional random number generators have been used in HSS. The traditional linear feedback shift register and the proposed random number generator. The results of the conducted comparisons between the two generators indicated that:
  - A. The output sequence of the proposed random generator has longer period than that for LFSR. The periodicity of its output is about  $2^N$ , where  $N$  is the product of multiplying the

length value of the six vector generators used in the proposed random number generator. The six values representing the length of the vectors must be relatively prime numbers. The output sequence length is long enough to encrypt the huge HTML files. Since the period length of the key stream depends on the length of the individual generators used in the scheme, the number of generators could be increased when there is a need to elongate the period.

- B.** In linear feedback shift register generator, the periodicity of its key stream is about  $2^{32}$  bit long (i.e., 4 GBytes), and this relatively short periodicity is fair enough to encrypt HTML files.
  - C.** The output sequence of the two generators has shown acceptances level of randomness according to the statistical randomness tests.
  - D.** The proposed random generator is faster than LFSR generator. This was indicated from the results of the conducted time tests. It was found that the proposed random generator is (30% to 40%) faster than the traditional generator. This percentage range varies according to the size and contents of HTML file.
- 3.** From Figure (4.34) and Figure (4.35), the relation between the HTML file size and the total encoding time shows a sort of approximately linear dependency. This linear relationship is obvious when the contents of encrypted HTML file are of text type. While, when HTML file has significant ratio of non-text data, then the encoding time become less than the time expected by the linear relationship.

4. The decoding time takes more than the encoding time, because the size of the encrypted HTML file is larger than the size of its original form (i.e., plaintext).
5. The time taken for color and font encryption and decryption is about (1% to 2%) of the total time of whole encryption and decryption operations.

### **5.3 Future Works**

1. Improve the workflow of the proposed HSS to be online HTML file security system.
2. Using public key encryption methodology to extend the application domain of the proposed HSS (i.e., become applicable by more than one corporation).
3. Extend the work of the proposed HSS to be capable to secure other types of Web-files (like, Extensible Markup Language XML).
4. Improve the proposed HSS to encrypt a Website not only a Webpage, by improving the system capability to trace hyperlinks.

# References

- [Aba06] Abass, A.; “An Encryption Based Security System for E-mail “; M.Sc. thesis; Informatics Institute for Postgraduate Studies; Iraq; 2006.
- [Alm05] Al Moosaway, E.; “Encryption and Compression of HTML Files”; M.Sc. thesis; University of Technology; Iraq; 2005.
- [Alw03] Alwash, W.; “Design and Implementation of an E-mail Security System”; M.Sc. thesis; College of Science; Al-Nahrain University; Iraq; 2003.
- [Ant00] Ants Soft; “HTML Protector”; Web Master Software Tools; 2000.
- [Bar05] Baker, E.; “Recommendation for Key Management”; NIST Special Publication 800-57; 2005.
- [Ber92] Berson, A.;”Differential Cryptanalysis Mod  $2^{32}$  with Applications to MD5”; EUROCRYPT: 71-80; ISBN 3-540-56413-6.
- [Bha03] Bhasin, S.; “Web Security Basics”; Premier Press; 2003.
- [BoFr93] Borenstein, N. and Freed, N.; ”MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and

Describing the Format of Internet Message Bodies”; Bell core; Inn soft; [RFC 1521](#); September; 1993.

- [Bos93]** Bosselaers, A; ”Collisions for the Compression Function of MD5”; Springer; pp. 239-304; London; ISBN 3-540-57600-2.
  
- [Bro98]** Brown, M. R.; ”Special Edition Using HTML 4”; Macmillan Computer Publishing; Indianapolis; USA; 1998.
  
- [Can01]** Canavan, E.;” Fundamentals of Network Security”; Artech House Inc.; London; 2001; ISBN 1-58053-176-8.
  
- [Cin99]** Cintron, D.; “Fast Track Web Programming: A Programmer’s Guide to Mastering Web Technologies”; Wiley Computer Publishing; 1999.
  
- [Cis89]** Cisco; “Designing Network Security”; Cisco Press Publications; 1989.
  
- [Dar98]** Darnell, R.; ”HTML 4 Un-Leashed”; Macmillan Computer Publishing; 2<sup>nd</sup> Edition; 1998.
  
- [DeKn07]** Delfs, H., Knebl, H.; ”Introduction to Cryptography Principles and Applications”; Springer; 2007.
  
- [Dor83]** Dorothy, E.; ”Cryptography and Data Security”; Addison-Wesley Publication Inc.; 1983.

- [**Gar01**] Garfinkel, S.; “Web Security: Privacy & Commerce”; 2<sup>nd</sup> Edition; O’ Reilly; 2001.
- [**GoBe01**] Goldwasser, S. and Bellare, M.; “Lecture Notes on Cryptography”; MIT Laboratory of Computer Science; USA; 2001.
- [**KeMu06**] Kennedy, B. and Musciano, C.; “HTML & XHTML: The Definitive Guide”; O’Reilly Publishing; 6<sup>th</sup> Edition; 2006.
- [**Lia09**] Lian, S.; “Multimedia Content Encryption Techniques and Applications”; CRC Press Inc.; 2009.
- [**Mel04**] Melnick; C.; ”What is Base64”; Copyright 2004; Website:  
<http://www.aardwulf.com>
- [**Men96**] Menezes, J.; ”Handbook of Applied Cryptography”; CRC Press Inc.; 1996.
- [**NaSt00**] Navarro, A. and Stauffer, T.; “HTML by Example”; Que Printing; Indiana; 2000.
- [**NaTa98**] Navarro, A. and Tabinda, K.; ”Effective Web Design: Master the Essentials”; Sybex Inc.; 1998.
- [**Net90**] Network Associates Inc.; ”Introduction to Cryptography”; 1990.
- [**Opp05**] Oppliger, R.;”Contemporary Cryptography”; Aretch House Inc.; 2005.



- [Ozc03] Ozcan, M.; “Design and Development of Practical and Secure E-mail System”; M.Sc. thesis; Graduate School of Engineering and Natural Sciences”; Sabanci University; Istanbul; 2003.
- [Pfa00] Pfaffenberger, B.; “HTML Bible”; 2<sup>nd</sup> Edition; IDG Books; Worldwide Inc.; 2000.
- [Pro03] ProtoWare; “HTML Guardian”; Protoware Inc.; Shareware Connection; 2003.
- [Rad07] Radhamani, G.; “Web Service Security and E-Business”; IDEA Group publishing; Multi Media University; Malaysia; 2007.
- [Rhe03] Rhee, M.; “Internet Security Cryptographic Principles, Algorithms and Protocols”; John Wiley & Sons Ltd.; Seoul National University; 2003.
- [Sch96] Schneier, B.; ”Applied Cryptography: Protocols, Algorithm, and Source Code in C”; Wiley Computer Publishing; John Wiley & Sons Inc.; 2<sup>nd</sup> Edition; ISBN: 0471128457; 1996.
- [Sta96] Staffer, T.;” HTML by Example”; Que printing; 1996.
- [Sta03] Stallings, W.; “Cryptography and Network Security Principles and Practices”; Prentice Hall; 2<sup>nd</sup> Edition; New Jersey; 2003.
- [Sta05] Stallings, W.; “Cryptography and Network Security Principles and Practices”; Prentice Hall; 4<sup>th</sup> Edition; New Jersey; 2005.

- [Sti95]** Stinson, D.; "Cryptography: Theory and Practice"; CRC Press Inc; 1995.
- [Tan03]** Tanenbaum; A.; "Computer Network"; 4<sup>th</sup> Edition; Prentice Hall Publishing; March 2003.
- [TiBu05]** Tittel, E. and Burmeister, M.; "HTML 4 for Dummies"; Wiley Publishing Inc.; 5<sup>th</sup> Edition; Indiana; 2005.
- [Til05]** Tilorg, V.; "Encyclopedia of Cryptography and Security"; Springer; 2005.
- [WiBa98]** Williams, A. and Barber, K.; "Active Server Pages Black Book"; Coriolis Group; 1998.

# Appendix A

This appendix illustrates the MD5 operations and its rounds, which are used in the implementation of the proposed system.

## A. The MD5 Message Digest [Sch96]

1. The four 32-bit variables (A, B, C, D) are initialized to have the following values:

$$A = 0x01234567$$

$$B = 0x89abcdef$$

$$C = 0xfedcba98$$

$$D = 0x76543210$$

These are called chaining variables.

2. The four variables are copied to different set of variables (i.e., a, b, c, d): *a* gets *A*, *b* gets *B*, *c* gets *C*, and *d* gets *D*.
3. The main loop has four rounds. Each round uses different operation 16 times. Each operation performs a nonlinear function on three elements of the set  $\{a, b, c, d\}$ . then, the result is added to the fourth variable, a sub-block of the text and a constant. Then, rotates that result to the left for a variable number of bits and adds the result to one of  $\{a, b, c, d\}$ . Finally, the result replaces one of  $\{a, b, c, d\}$ .
4. If  $M_j$  represents the  $j^{\text{th}}$  sub-block of the message (from 0 to 15), and  $\lll s$  represents a left circular shift of  $s$  bits, then, the four following operations are applied:

$$\text{FF}(a,b,c,d,M_j,s,t_i) \text{ denotes } a = b + ((a + F(b,c,d) + M_j + t_i) \lll s)$$

$$\text{GG}(a,b,c,d,M_j,s,t_i) \text{ denotes } a = b + ((a + G(b,c,d) + M_j + t_i) \lll s)$$

HH(a,b,c,d,Mj,s,ti) denotes  $a = b + ((a + H(b,c,d) + M_j + t_i) \lll s)$

II(a,b,c,d,Mj,s,ti) denotes  $a = b + ((a + I(b,c,d) + M_j + t_i) \lll s)$

Such that, the following four rounds (64 steps) are applied:

**Round 1:**

FF (a, b, c, d, M0, 7, 0xd76aa478)  
 FF (d, a, b, c, M1, 12, 0xe8c7b756)  
 FF (c, d, a, b, M2, 17, 0x242070db)  
 FF (b, c, d, a, M3, 22, 0xc1bdcee)  
 FF (a, b, c, d, M4, 7, 0xf57c0faf)  
 FF (d, a, b, c, M5, 12, 0x4787c62a)  
 FF (c, d, a, b, M6, 17, 0xa8304613)  
 FF (b, c, d, a, M7, 22, 0xfd469501)  
 FF (a, b, c, d, M8, 7, 0x698098d8)  
 FF (d, a, b, c, M9, 12, 0x8b44f7af)  
 FF (c, d, a, b, M10, 17, 0xffff5bb1)  
 FF (b, c, d, a, M11, 22, 0x895cd7be)  
 FF (a, b, c, d, M12, 7, 0x6b901122)  
 FF (d, a, b, c, M13, 12, 0xfd987193)  
 FF (c, d, a, b, M14, 17, 0xa679438e)  
 FF (b, c, d, a, M15, 22, 0x49b40821)

**Round 2:**

GG (a, b, c, d, M1, 5, 0xf61e2562)  
 GG (d, a, b, c, M6, 9, 0xc040b340)  
 GG (c, d, a, b, M11, 14, 0x265e5a51)  
 GG (b, c, d, a, M0, 20, 0xe9b6c7aa)  
 GG (a, b, c, d, M5, 5, 0xd62f105d)  
 GG (d, a, b, c, M10, 9, 0x02441453)  
 GG (c, d, a, b, M15, 14, 0xd8a1e681)  
 GG (b, c, d, a, M4, 20, 0xe7d3fbc8)

GG (a, b, c, d, M9, 5, 0x21e1cde6)  
GG (d, a, b, c, M14, 9, 0xc33707d6)  
GG (c, d, a, b, M3, 14, 0xf4d50d87)  
GG (b, c, d, a, M8, 20, 0x455a14ed)  
GG (a, b, c, d, M13, 5, 0xa9e3e905)  
GG (d, a, b, c, M2, 9, 0xfcefa3f8)  
GG (c, d, a, b, M7, 14, 0x676f02d9)  
GG (b, c, d, a, M12, 20, 0x8d2a4c8a)

**Round 3:**

HH (a, b, c, d, M5, 4, 0xfffa3942)  
HH (d, a, b, c, M8, 11, 0x8771f681)  
HH (c, d, a, b, M11, 16, 0x6d9d6122)  
HH (b, c, d, a, M14, 23, 0xfde5380c)  
HH (a, b, c, d, M1, 4, 0xa4beea44)  
HH (d, a, b, c, M4, 11, 0x4bdecfa9)  
HH (c, d, a, b, M7, 16, 0xf6bb4b60)  
HH (b, c, d, a, M10, 23, 0xbebfbcb70)  
HH (a, b, c, d, M13, 4, 0x289b7ec6)  
HH (d, a, b, c, M0, 11, 0xeeaa127fa)  
HH (c, d, a, b, M3, 16, 0xd4ef3085)  
HH (b, c, d, a, M6, 23, 0x04881d05)  
HH (a, b, c, d, M9, 4, 0xd9d4d039)  
HH (d, a, b, c, M12, 11, 0xe6db99e5)  
HH (c, d, a, b, M15, 16, 0x1fa27cf8)  
HH (b, c, d, a, M2, 23, 0xc4ac5665)

**Round 4:**

II (a, b, c, d, M0, 6, 0xf4292244)  
II (d, a, b, c, M7, 10, 0x432aff97)  
II (c, d, a, b, M14, 15, 0xab9423a7)

II (b, c, d, a, M5, 21, 0xfc93a039)  
II (a, b, c, d, M12, 6, 0x655b59c3)  
II (d, a, b, c, M3, 10, 0x8f0ccc92)  
II (c, d, a, b, M10, 15, 0xffeff47d)  
II (b, c, d, a, M1, 21, 0x85845dd1)  
II (a, b, c, d, M8, 6, 0x6fa87e4f)  
II (d, a, b, c, M15, 10, 0xfe2ce6e0)  
II (c, d, a, b, M6, 15, 0xa3014314)  
II (b, c, d, a, M13, 21, 0x4e0811a1)  
II (a, b, c, d, M4, 6, 0xf7537e82)  
II (d, a, b, c, M11, 10, 0xbd3af235)  
II (c, d, a, b, M2, 15, 0x2ad7d2bb)  
II (b, c, d, a, M9, 21, 0xeb86d391)

The constants,  $t_i$ , were chosen as follows:

In step  $i$ ,  $t_i$  is the integer part of  $2^{32} * \text{ABS}(\sin(i))$ , where  $i$  is in radians.

5. After all of this,  $a$ ,  $b$ ,  $c$ , and  $d$  are added to  $A$ ,  $B$ ,  $C$ ,  $D$ , respectively, and the algorithm continues with the next block of data. The final output is the concatenation of  $A$ ,  $B$ ,  $C$ , and  $D$ .

# Appendix B

**Table (B.1) Common character entities**

<i>Name</i>	<i>Symbol</i>	<i>Decimal</i>	<i>Description</i>
<i>quota</i>	“	&#34;	<i>Quotation mark</i>
	&	&	<i>ampersand</i>
<i>lt</i>	<	&#60;	<i>Less than</i>
<i>gt</i>	>	&#62;	<i>Greater than</i>
<i>nbsp</i>		&#160;	<i>Non breaking space</i>
<i>ixcl</i>	¡	&#161;	<i>Inverted exclamation mark</i>
<i>cent</i>	¢	&#162;	<i>Cent sign</i>
<i>pound</i>	£	&#163;	<i>Pound sign</i>
<i>curren</i>		&#164;	<i>Currency sign</i>
<i>yen</i>	¥	&#165;	<i>Yen sign = yuan sign</i>
<i>brybar</i>	/	&#166;	<i>Broken bar = broken vertical bar</i>
<i>sect</i>	§	&#167;	<i>Section sign</i>
<i>uml</i>	¨	&#168;	<i>Diaeresis = spacing diaeresis</i>
<i>copy</i>	©	&#169;	<i>Copyright sign</i>
<i>ordf</i>	ª	&#170;	<i>Feminine ordinal indicator</i>
<i>laquo</i>	«	&#171;	<i>Left-pointing double angle quotation mark</i>
<i>not</i>	¬	&#172;	<i>Not sign</i>
<i>shy</i>		&#173;	<i>Soft hyphen = discretionary hyphen</i>
<i>reg</i>	®	&#174;	<i>Registered sign = registered trade mark sign</i>
<i>macr</i>	–	&#175;	<i>Macron = spacing macron = overline</i>
<i>deg</i>	°	&#176;	<i>Degree sign</i>
<i>plusmn</i>	±	&#177;	<i>Plus-minus sign = plus-or-minus sign</i>
<i>sup2</i>	²	&#178;	<i>Superscript two = superscript digit two</i>
<i>sup3</i>	³	&#179;	<i>Superscript three = superscript digit three</i>
<i>acute</i>	´	&#180;	<i>Acute accent = spacing acute</i>
<i>micro</i>	μ	&#181;	<i>Micro sign</i>

Name	Symbol	Decimal	Description
<i>para</i>	¶	&#182;	<i>Pilcrow sign = paragraph sign</i>
<i>middot</i>	·	&#183;	<i>Middle dot = Georgian comma</i>
<i>cedil</i>	,	&#184;	<i>Cedilla = spacing cedilla</i>
<i>sup1</i>	<sup>1</sup>	&#185;	<i>Superscript one = superscript digit one</i>
<i>ordm</i>	º	&#186;	<i>Masculine ordinal indicator</i>
<i>raquo</i>	»	&#187;	<i>Right-pointing double angle quotation mark</i>
<i>frac14</i>	¼	&#188;	<i>Fraction one quarter</i>
<i>frac12</i>	½	&#189;	<i>Fraction one half</i>
<i>frac34</i>	¾	&#190;	<i>Fraction three quarters</i>
<i>iquest</i>	¿	&#191;	<i>Inverted question mark</i>
<i>Agrave</i>	À	&#192;	<i>Latin capital letter A with grave</i>
<i>Aacute</i>	Á	&#193;	<i>Latin capital letter A with acute</i>
<i>Acirc</i>	Â	&#194;	<i>Latin capital letter A with circumflex</i>
<i>Atilde</i>	Ã	&#195;	<i>Latin capital letter A with tilde</i>
<i>Auml</i>	Ä	&#196;	<i>Latin capital letter A with diaeresis</i>
<i>Aring</i>	Å	&#197;	<i>Latin capital letter A with ring above</i>
<i>AElig</i>	Æ	&#198;	<i>Latin capital letter AE</i>
<i>Ccedil</i>	Ç	&#199;	<i>Latin capital letter C with cedilla</i>
<i>Egrave</i>	È	&#200;	<i>Latin capital letter E with grave</i>
<i>Eacute</i>	É	&#201;	<i>Latin capital letter E with acute</i>
<i>Ecirc</i>	Ê	&#202;	<i>Latin capital letter E with circumflex</i>
<i>Euml</i>	Ë	&#203;	<i>Latin capital letter E with diaeresis</i>
<i>Igrave</i>	Ì	&#204;	<i>Latin capital letter I with grave</i>
<i>Iacute</i>	Í	&#205;	<i>Latin capital letter I with acute</i>
<i>Icirc</i>	Î	&#206;	<i>Latin capital letter I with circumflex</i>
<i>Iuml</i>	Ï	&#207;	<i>Latin capital letter I with diaeresis</i>
<i>ETH</i>	Ð	&#208;	<i>Latin capital letter ETH</i>
<i>Ntilde</i>	Ñ	&#209;	<i>Latin capital letter N with tilde</i>
<i>Ograve</i>	Ò	&#210;	<i>Latin capital letter O with grave</i>
<i>Oacute</i>	Ó	&#211;	<i>Latin capital letter O with acute</i>
<i>Ocirc</i>	Ô	&#212;	<i>Latin capital letter O with circumflex</i>



Name	Symbol	Decimal	Description
Otilde	Õ	&#213;	Latin capital letter O with tilde
Ouml	Ö	&#214;	Latin capital letter O with diaeresis
times	x	×	Multiplication sign
Oslash	Ø	&#216;	Latin capital letter O with stroke
Ugrave	Ù	&#217;	Latin capital letter U with grave
Uacute	Ú	&#218;	Latin capital letter U with acute
Ucirc	Û	&#219;	Latin capital letter U with circumflex
Uuml	Ü	&#220;	Latin capital letter U with diaeresis
Yacute	Ý	&#221;	Latin capital letter Y with acute
THORN	Þ	&#222;	Latin capital letter THORN
szlig	ß	&#223;	Latin small letter sharp s = ess-zed
agrave	à	&#224;	Latin small letter a with grave
aacute	á	&#225;	Latin small letter a with acute
acirc	â	&#226;	Latin small letter a with circumflex
atilde	ã	&#227;	Latin small letter a with tilde
auml	ä	&#228;	Latin small letter a with diaeresis
aring	å	&#229;	Latin small letter a with ring above
aelig	æ	&#230;	Latin small letter ae
ccedil	ç	&#231;	Latin small letter c with cedilla
egrave	è	&#232;	Latin small letter e with grave
eacute	é	&#233;	Latin small letter e with acute
ecirc	ê	&#234;	Latin small letter e with circumflex
euml	ë	&#235;	Latin small letter e with diaeresis
igrave	ì	&#236;	Latin small letter i with grave
iacute	í	&#237;	Latin small letter i with acute
icirc	î	&#238;	Latin small letter i with circumflex
iuml	ï	&#239;	Latin small letter i with diaeresis
eth	ð	&#240;	Latin small letter eth
ntilde	ñ	&#241;	Latin small letter n with tilde
ograve	ò	&#242;	Latin small letter o with grave
oacute	ó	&#243;	Latin small letter o with acute

<i>Name</i>	<i>Symbol</i>	<i>Decimal</i>	<i>Description</i>
<i>ocirc</i>	ô	&#244;	<i>Latin small letter o with circumflex</i>
<i>otilde</i>	õ	&#245;	<i>Latin small letter o with tilde</i>
<i>ouml</i>	ö	&#246;	<i>Latin small letter o with diaeresis</i>
<i>divide</i>	÷	&#247;	<i>Division sign</i>
<i>oslash</i>	ø	&#248;	<i>Latin small letter o with stroke</i>
<i>ugrave</i>	ù	&#249;	<i>Latin small letter u with grave</i>
<i>uacute</i>	ú	&#250;	<i>Latin small letter u with acute</i>
<i>ucirc</i>	û	&#251;	<i>Latin small letter u with circumflex</i>
<i>uuml</i>	ü	&#252;	<i>Latin small letter u with diaeresis</i>
<i>yacute</i>	ý	&#253;	<i>Latin small letter y with acute</i>
<i>thorn</i>	þ	&#254;	<i>Latin small letter thorn</i>
<i>yuml</i>	ÿ	&#255;	<i>Latin small letter y with diaeresis</i>

# Appendix C

**Table (C.1) The Selected percentiles of the  $\chi^2$  (chi-square) distribution**

$\nu$	$\alpha$					
	0.100	0.050	0.025	0.010	0.005	0.001
1	2.7055	3.8415	5.0239	6.6349	7.8794	10.8276
2	4.6052	5.9915	7.3778	9.2103	10.5966	13.8155
3	6.2514	7.8147	9.3484	11.3449	12.8382	16.2662
4	7.7794	9.4877	11.1433	13.2767	14.8603	18.4668
5	9.2364	11.0705	12.8325	15.0863	16.7496	20.5150
6	10.6446	12.5916	14.4494	16.8119	18.5476	22.4577
7	12.0170	14.0671	16.0128	18.4753	20.2777	24.3219
8	13.3616	15.5073	17.5345	20.0902	21.9550	26.1245
9	14.6837	16.9190	19.0228	21.6660	23.5894	27.8772
10	15.9872	18.3070	20.4832	23.2093	25.1882	29.5883
11	17.2750	19.6751	21.9200	24.7250	26.7568	31.2641
12	18.5493	21.0261	23.3367	26.2170	28.2995	32.9095
13	19.8119	22.3620	24.7356	27.6882	29.8195	34.5282
14	21.0641	23.6848	26.1189	29.1412	31.3193	36.1233
15	22.3071	24.9958	27.4884	30.5779	32.8013	37.6973
16	23.5418	26.2962	28.8454	31.9999	34.2672	39.2524
17	24.7690	27.5871	30.1910	33.4087	35.7185	40.7902
18	25.9894	28.8693	31.5264	34.8053	37.1565	42.3124
19	27.2036	30.1435	32.8523	36.1909	38.5823	43.8202
20	28.4120	31.4104	34.1696	37.5662	39.9968	45.3147
21	29.6151	32.6706	35.4789	38.9322	41.4011	46.7970
22	30.8133	33.9244	36.7807	40.2894	42.7957	48.2679
23	32.0069	35.1725	38.0756	41.6384	44.1813	49.7282
24	33.1962	36.4150	39.3641	42.9798	45.5585	51.1786
25	34.3816	37.6525	40.6465	44.3141	46.9279	52.6197
26	35.5632	38.8851	41.9232	45.6417	48.2899	54.0520
27	36.7412	40.1133	43.1945	46.9629	49.6449	55.4760
28	37.9159	41.3371	44.4608	48.2782	50.9934	56.8923
29	39.0875	42.5570	45.7223	49.5879	52.3356	58.3012
30	40.2560	43.7730	46.9792	50.8922	53.6720	59.7031
31	41.4217	44.9853	48.2319	52.1914	55.0027	61.0983
63	77.7454	82.5287	86.8296	92.0100	95.6493	103.4424
127	147.8048	154.3015	160.0858	166.9874	171.7961	181.9930
255	284.3359	293.2478	301.1250	310.4574	316.9194	330.5197
511	552.3739	564.6961	575.5298	588.2978	597.0978	615.5149
1023	1081.3794	1098.5208	1113.5334	1131.1587	1143.2653	1168.4972

# الخلاصة

يعتبر الانترنت من الأوساط المهمة والشائعة الاستخدام في نقل المعلومات وذلك لسهولة وقلة كلفة استخدامه. وقد أدت الزيادة في عدد المنظمات والشركات والأفراد المنتفعة من استخدام الانترنت إلى زيادة في حجم الحاجة إلى تأمين طرق لنقل البيانات بشكل أكثر مؤمن عبر الانترنت. من اللغات الشائعة لكتابة صفحات الويب هي اللغة المعلمة للنص المتشعب (HTML) وان صفحات الويب المكتوبة بهذه اللغة يمكن تفعيلها بواسطة أي متصفح ويب. وعلى الرغم من شيوع استخدام هذه اللغة في أعداد الصفحات إلا أنها تتسم بضعف إمكانية حماية البيانات وتحقيق مستوى مقبول من الأمانية وذلك من خلال استخدام الوسائل والأدوات القياسية التي تتبعها اللغة.

إن الهدف الرئيسي لهذا البحث هو حماية محتويات ملف ال(HTML) من القارئ غير المخولين. إن المتطلبات الامنيه المأخوذة بنظر الاعتبار في حماية محتويات ملف ال(HTML) هي تحقيق السرية (الخصوصية) وتثبيت سلامة المعلومات. فالخصوصية يمكن تحقيقها بواسطة طرق التشفير المعروفة والمتداولة أما التحقق من سلامة المعلومات فيمكن تحقيقه من خلال استخدام طريقة دالة العلامات (MD5) لإنتاج توقيع رقمي (message digest) لمحتويات الملف المرمز.

إن النظام المقترح في هذا البحث لحماية محتويات الملف (HTML) يتكون من وحدتين، هما وحدة التشفير ووحدة فك التشفير، وتتضمن كل وحدة ثلاث مراحل: توليد الأرقام العشوائية، دالة علامات الرقم والترميز باستخدام النظام الترميز-64. بالنسبة إلى مولدات الأرقام العشوائية، تم في هذا البحث استخدام مولدين للأرقام العشوائية. الأول هو مسجل الزحف الخطي ذو التغذية الرجعية (LFSR). والثاني هو مولد الأرقام العشوائية المقترح. بالنسبة إلى مسجل الزحف الخطي ذو

التغذية الرجعية فهو يولد سلسلة مكونة من  $2^{32}$  من الأرقام العشوائية . أما المولد المقترح فهو يولد تقريبا  $2^N$  من الأرقام العشوائية حيث  $N$  متغير تعتمد قيمته على أطوال المولدات المستخدمة في المولد المقترح.

أن نتائج الاختبارات الاحصائية التي أجريت لاختبار كفاءة كلا المولدين أشارت إلى أن الأرقام العشوائية المتولدة منها ذات مستوى مقبول من العشوائية . كما أن نتائج اختبار الوقت المستغرق لكلا المولدين في إنجاز عملية التشفير وفك الشفرة أشارت إلى كون المولد المقترح أسرع من المولد التقليدي (مسجل الزحف الخطي ذو التغذية الرجعية) بنسبة تتراوح بين (٣٠% إلى ٤٠%) وذلك اعتمادا على حجم ملف (HTML).

وقد تم دعم مناعة النظام المقترح باستخدام نظام الترميز - ٦٤ (encoding)

(Base64) لغرض ترميز النص المشفر الناتج من عملية التشفير.



جمهورية العراق  
وزارة التعليم العالي والبحث العلمي  
جامعة النهرين  
كلية العلوم

# تحسين أمنية ملفات اللغة المعلمة للنص المتشعب

رسالة

مقدمة الى كلية العلوم في جامعة النهرين كجزء من  
متطلبات نيل درجة الماجستير في علوم الحاسبات

من قبل

ناديه فاضل أبراهيم

(بكالوريوس الجامعة التكنولوجية ١٩٨٨)

إشراف

د. لؤي أدور جورج