بسم الله الرحمن الرحيم

Al Al-bayt University

Prince Hussein bin Abdullah College for Information Technology

Computer Science Department

Preventing Software Piracy Using Public Key Cryptography

By

Khaldoon Khawaldeh

2008

Preventing Software Piracy Using Public Key Cryptography

By

Khaldoon Khawaldeh

Supervisor: Prof. Adnan M. Al-Smadi

Co-Supervisor: Prof. Sattar J. Aboud

A Thesis Submitted to the
Scientific Research and Graduate Faculty in partial fulfillment of the
Requirements for the Degree of Master of Science
In Computer Science

Members of the Committee	**Approved**
Prof. Adnan M. Al-Smadi	
Prof. Sattar JAboud	
Dr. Mamoun Al Rababaa	
Dr. Jehad Al Nihoud	
Dr. Isam Al-Dauood	

Al Al-Bayt University
Mafraq, Jordan
2008

Dedication

To my family and friends…

For their encouragement and understanding during the period of my study.

Acknowledgements

First of all, I would like to thank God for his graces to me and for his guidance. Thanks also for Prof. Adnan M. Al-Smadi, and Prof. Sattar J. Aboud for their acceptance to be my supervisors, and for their extensive guidance, assistance and scientific hints that lightened my road during my study. Without their help and contributions, this work could not have been accomplished.

Also, I would like to thank the members of the thesis examination committee for their advice and hints that have contributed to enhancement of this research.

List of Contents

Subject	Page
Dedication	B
Acknowledgement	C
Table of contents	D
List of tables	E
List of figures	F
List of appendices	G
List of abbreviations	H
Abstract	I
Chapter one: Introduction	1
1. Introduction	1
1.1 Problem definition	2
1.2 Objectives	3
1.3 Motivations	3
1.4 Significance of study	4
1.5 Contributions	4
1.6 Thesis organization	5
2. Related work	5
Chapter two: Software piracy and software protection	10
2.1 Technical protection methods	10
2.1.1 Media-based protections	10
2.1.2 Serial-based protections	10
2.1.3 Hardware-based protections	13
2.2.4 Software as a service	13
2.1.5 Digital rights management	14
2.1.6 Obfuscation	14
2.1.7 Software diversity	15
2.2 Software cracking	15
2.2.1 Analysis	16
2.2.2 Tampering	18
2.2.3 Automation	19
2.2.4 Distribution	19
2.3 Cracks classification	19
2.3.1 Leaked serials	19
2.3.2 Key Generators	19
2.3.3 Backup-media	20

Subject	Page
2.3.4 No-CD/DVD	21
2.3.5 Patches	21
2.3.6 Fixed EXEs	21
2.3.7 Loaders	22
2.4 Specific availability	22
Chapter three: The proposed system	24
1. Used algorithms	25
2. Codes definition	50
3. Protecting EXE files	53
4. How to obtain hardware information for each client	54
5. Authentication	55
6. Customer tracking system	55
7. How the application work	62
Chapter four: Analyzing and testing of the proposed scheme	63
4.1 System Analysis	63
4.2 Experimental Test	81
Chapter five: Conclusion and future work	93
5.1 Conclusion	93
5.2 Future work	93
References	95
الملخص	98
Appendices	99

List of Tables

Table	Page
Table 3.1 TDES - PC-1	31
Table 3.2 Triple DES Shifts	32
Table 3.3 Triple DES - PC-2	34
Table 3.4 Triple DES -IP	35
Table 3.5 Triple DES- E Bit-Selection Table	37
Table 3.6 Triple DES- S_1 Determination table	38
Table 3.7 Triple DES- The definitions for $S_1,...,S_8$	39
Table 3.8 Triple DES – P definition	40
Table 3.9 Triple DES- IP^{-1} Definition	41
Table 3.10 Customers table structure	58
Table 3.11 Activations table structure	59
Table 3.12 Copy IDs table structure	60
Table 3.13 Products table structure	60
Table 3.14 Entity Relationship Model's relations	61
Table 4.1 Complexities used in the Encapsualtor	66
Table 4.2 Complexities used in ISCN	68
Table 4.3 Complexities used in the zero knowledge protocol	74
Table 4.4 Number of iterations to calculate the inverse using Baghdad method	75
Table 4.5 Complexities used in the enhanced RSA	78
Table 4.6 Number of iterations to encrypt/decrypt using enhanced RSA	79
Table 4.7 Complexities used in path-1 of the protection interface	80
Table 4.8 Complexities used in path-2 of the protection interface	81
Table 4.9 System Conditions for case 1	84
Table 4.10 System Conditions for case 2	85
Table 4.11 System Conditions for case 3	86
Table 4.12 System Conditions for case 4	87
Table 4.13 System Conditions for case 5	88
Table 4.14 System Conditions for case 6	89
Table 4.15 System Conditions for case 1 in the automatic mode	90
Table 4.16 System Conditions for case 2 in the automatic mode	90
Table 4.17 System Conditions for case 3 in the automatic mode	91
Table 4.18 System Conditions for case 4 in the automatic mode	91
Table 4.19 System Conditions for case 5 in the automatic mode	92

List of Figures

Figure	Page
Figure 2-1 Non parameter-based verification scheme	11
Figure 2-2 User parameter-based verification scheme	12
Figure 2-3 Compilation and reverse engineering flowchart	17
Figure 2-4 High-level creation of a critical decision point	18
Figure 2-5 Normal program functionality	18
Figure 2-6 A typical non parameter-based key generator	20
Figure 2-7 A patch	21
Figure 2-8 Graph depicting the availability of cracks	23
Figure 3.1 Machine ID structure	50
Figure 3.2 Installation ID generation	50
Figure 3.3 The Encapsulation Process	53
Figure 3.4 Entity Relationship Model	57
Figure 3.5 Program's Execution Flow chart	62
Figure 4.1 Execution Time Snapshot	64
Figure 4.2 Pseudo Code for files combining	65
Figure 4.3 Pseudo Code for ISCN	67
Figure 4.4 Pseudo Code for password conversion algorithm	70
Figure 4.5 Pseudo Code for Baghdad Method	72
Figure 4.6 Pseudo Code for the rest of the operations in the authentication protocol	73
Figure 4.7 Pseudo Code for the enhanced RSA	76

List of Appendices

Appendix	Page
Appendix 1 – Installation Manual	99
Appendix 2 – Major Code	101

List of Abbreviations

Abbreviation	Meaning
DES	Data Encryption Standard
MD5	Message-Digest algorithm 5
Triple DES	Triple Data Encryption Standard
ISCN	International Standard Copy Number
TDES	Triple Data Encryption Standard
CRM	Customer Relationship Management
RSA	Initials for R. Rivest, A. Shamir, L. Adleman
CPU	Central Processing Unit
BSA	Business Software Alliance
IDC	International Data Corporation
RAM	Random Access Memory
IP	Internet Protocol Address
CD	Compact Disc
ROM	Read Only Memory
DRM	Digital Rights Management
DVD	Digital Versatile Disc
ID	Identification

Abstract

Many organizations face difficulties with piracy. There are a variety of systems available on markets to treat this problem but the fact that most of these system dos not free the problem. After surveying most of the current systems and after addressing their features and shortcomings, we proposed a new system to deal with these problems. The proposed system uses standard techniques to ease these difficulties such as Zero knowledge proof, RSA, MD5, and Triple DES. The finding result is dynamic, scalable, and efficient. It claims that this scheme is more acceptable compared with the already existed systems on the market.

The author implemented a method where a software application hashes hardware serial numbers to generate a unique Installation ID. This Installation ID is sent to the manufacturer to verify the authenticity of the application and to ensure that the product is not being used for multiple installations. The author implemented a generic program to enable this technique on any developed program.

Chapter One

1. Introduction

Software Piracy can be defined as the unauthorized use of commercial software product. It covers any product for which the software developer is not justly rewarded, and is understandably a major concern in the computer market (Albert and Morse, 1984). While the threat of legal action is effective against possible corporate pirates, it is ineffective against the individuals who pirate personal computer software.

The worldwide revenue of business-based personal computer applications was $21.6 billion in 1999, but global revenue losses due to piracy in the business application software market were calculated at $12 billion in the same year (Cowan, 2003).

In December 2005, a study by the U.S. research firm IDC(International Data Corporation) showed that a 10% reduction in software piracy in the European Union could encourage the Information Technology industry's growth rate from the current 30% to 38% through 2009 (Zoller and Renate, 2007).

Since piracy also depresses demand for software design, customization and support, the study estimated that a 10% reduction could add about $400 billion to economies worldwide and $67 billion in tax revenues (Zoller and Renate, 2007).

For software publishers, a less expensive method of copy protection is used. They programmed their software to check for certain evidence from the users to prove that they have actually purchased the software.

It is estimated that $29 billion of the total $80 billion of software installed on computers was actually installed illegally (BSA, 2007). Piracy rates in 2003 ranged from 92% in Vietnam and China compare to 22% in the United States with a global piracy rate of 36% (BSA, 2007).

The Business Software Alliance (BSA) defines five common types of software piracy (BSA, 2007):

1. End-user piracy occurs when a user reproduces copies of software without authorization. It can manifest itself in one of the following forms:

    - A user obtains a single licensed copy and uses it to install the software on multiple computers.

- The disks used to install the software are duplicated and then distributed.
- A user purchases and installs an upgrade without previously having a legal version.
- Within a commercial environment, employees use software with an academic license.

2. Client-server piracy occurs when a program is installed on a network and is simultaneously used by more people than the license entitles.
3. Internet piracy occurs when illegal copies of software are made available on the Internet either free of charge or for a fee. Examples of such sites include:
   - Sites which make software available for free or by exchanging uploaded programs.
   - Auction sites that offer illegal software.
   - Peer-to-peer network sites which enable the transfer of illegal software.
4. Hard-disk loading occurs when illegal software is installed on a new computer and sold. This activity often occurs when a business is trying to cut costs to make their products more attractive.
5. Software piracy occurs when copyrighted material is illegally duplicated and sold with the intent that the material passes as the original.

In the United States, and many other countries, there are legal ramifications associated with software piracy. A violation of the United States Copyright Act can result in damages of up to $150,000 for each program copied (BSA, 2007).

1.1 Problem Definition

Software is intellectual property. It should be protected from unauthorized users in order to ensure that the existing revenue runs. Software piracy continues to grow globally because it is cheap and easy to copy. The effects of this grew are devastating: not only does software piracy reduce revenues, it also results in less research and development, and in less investment in marketing and channel development.

It was found that losses due to piracy in the Middle East region equals to 1,997 million dollar in 2006 (IDC, 2007). It is important to minimize the piracy rates as possible. Since the legal and physical methods failed to prevent software piracy; it is necessary to

protect software using technical and mathematical methods. In this thesis, we propose a new scheme that will prevent software piracy based on public cryptography.

1.2 Objectives

The objectives of this thesis are as follows:

- To generate a secure and efficient scheme that prevents unauthorized users to pirate.
- To make a secure way to monitor and track the users when they follow certain software.
- To provide an easy way to the software vendors to protect their software by implementing a generic application technique.
- To evaluate the proposed system by implementing on computer program, and to check the usability and the trust of the system.

1.3 Motivations

We proposed this scheme for the following reasons:

- To discuss techniques that may generate new opportunities for software companies.
- To address several protection aspects unlike the current weak techniques which have single points of failure or have performance penalties. The suggested approach addressed several aspects which makes it uniquely effective method.
- To obtain remarkable results using new tools such as (.net) frame work, and a variety of good studies such as RSA scheme, Baghdad inverse method, and zero knowledge protocol. Without these methods and supporting researches, we would not be able to advance in the topic.
- Despite of the damage that piracy caused to the local markets, no scientific approach took place to study the local pirated community.
- To reach results that was hard to reach in the past. This is done with the obtained power of the three giga hertz Intel Pentium four processor, and the efficiency of the used programming languages C#.

1.4 Significance of study

This study lives to serve both the software developer and the software user in the following points:

- The developer will be able to develop without feeling threatened by the ghost of piracy, which will give him more time to concentrate on the development process rather than the protection process.
- The user will benefit from the provided technical support which will lead to the evolution of the program.
- Previous studies concentrated greatly on one problem and neglected the other problems while this study tries to examine several aspects and conditions.
- The enterprise administrator will be able to prevent piracy in his company. Where only the authorized personnel will be able to use their applications in the authorized places only.

1.5 Contributions

In this section we describe a number of contributions:

- The implemented system is efficient. It provided no execution time penalty on the protected programs.
- The implemented system is dynamic. It can be easily applied on any (.net) based software without the need for its source code.
- It is a scalable software based method that does not bind the user to a specific version of the operating system.
- The implemented system serves all kind of users. Home users who have no network connection, home users who have no permanent internet connection, high security enterprise users who should execute their applications only from a specific location, and other users who have mixed conditions.
- The implemented protection interface provides a scalable architecture that can adapt to changes and can integrate more techniques and features.

1.6 Thesis Organization

The thesis organizations are as follows:

- Chapter one provides general description for the problem. It also talks about previous related work.
- Chapter two gives theoretical analysis about the piracy and its prevention techniques.
- Chapter three talks about the suggested system. it explains its parts and how it works.
- Chapter four analysis the system's efficiency. Then it discusses the obtained results from the system.
- Chapter five talks about the conclusion and the future possible work for the system.

2. Related Work

In 1989, Ashileshwari N. Chandra, Liam D. Comerford, and Steve R. White (Chandra, and Comerford, 1989) proposed a system called Software protection using single key cryptosystem. A hardware based authorization system and a secure coprocessor. This method provides a software asset protection mechanism which is based on the separation of the software to be protected from the right to execute the software. Protected software can only be executed on composite computing systems in which a physically and logically secure coprocessor is associated with a host computer. The software to be protected is broken down into a protected that is encrypted portion and an optional unprotected or plain text portion. The software is distributed by any conventional software distribution mechanism for example a floppy disk including the files already identified along with an encrypted software decryption key. The coprocessor is capable of decrypting the software decryption key so it can thereafter decrypt the software, for execution purposes. However, the coprocessor will not perform these functions unless and until the user's right to execute is evidenced by presentation of a physically secure token. The physically secure token provides to the coprocessor token data in plain text form the physical security of the plain text token data is provided by the cartridge within which token data is stored. The physical properties of that cartridge taken together with the correspondence between the token

data provided by the cartridge and the encrypted token data evidence the user's right to execute. While the coprocessor can, thereafter, decrypt and execute the protected portion of the software, access to that software is denied the user by the physical and logical features of the coprocessor. Other properties of the cartridge specifically a destructive read property ensure that the act of transferring token data to the coprocessor obliterates that data from the cartridge so it cannot be revised. Further, the protocol for the coprocessor or cartridge exchange is arranged so that observation of even the entire exchange provides inadequate information with which to simulate or spoof the effect of an authentic, unused cartridge.

This method provided secure execution, where only the authorized user can run the program. It is commonly known as dongle based method. This method bounds the user to the coprocessor. It is irreplaceable unit, and proved to be not compatible with new operating systems. The users of this scheme had always suffered when they tried to upgrade their systems.

In 1996, Jon H. Barber, Ronald A. Woodward, Richard M. Burkley, Erwin L. Rehme, Matthew W. Jackson, and Douglas M. Young (Barber, Woodward and Burkley, 1996) proposed a system for controlling the number of concurrent copies of a program in a network based on the number of available licenses. License management systems and methods allow licenses for a computer program to be available for use at each of a plurality of nodes of a network. If a valid license file at a local node contains an unexpired, available license, a license manager at the local node permits the computer program to be executed at the requesting local node. If no such license is available in a valid license file at such local node, the license manager searches the other nodes for a valid license file containing an unexpired, available license. In one embodiment, if an unexpired available license is located in a valid license file at a second or remote node, the license manager transfers such license to the local node and assigns and encrypts a unique identification to such transferred license. The original record of the transferred license is modified by erasing it from the license file at the remote node so that the transferred license is no longer available there. In a second embodiment, the license manager modifies the license file to indicate use of the license at the local node without such transfer. The number of copies of the computer program that are authorized for execution simultaneously on the network is thus limited to the number of licenses that have been loaded into the license files on the network.

Many computer games vendors used this method to control concurrent execution of their games. They made some changes to it but the concept is the same. This scheme is effective in the case of online applications such as online multiplayer games, or in a network based system.

Despite that just the authorized users can execute the protected programs using this scheme. This scheme is not applicable in non-network based systems, or those systems which do not always have network connection.

In 2000, Matthew Schmid, Frank Hill, and A.K. Ghosh (Schmid, Hill, and Ghoshm 2000) proposed a system called Preventing the Execution of Unauthorized Win32 Applications. In this system they describe an approach and tool for providing administrative control over the execution of software on a Windows NT/2000 system. The kernel-driver–based approach provides the system administrator with a way of restricting users to running only approved applications. As a result, illegal, pirated, personal, and malicious software executables can be prevented from running on corporate machines. We describe the key issues involved in the development of this tool and the features that make this tool an important part of regaining enterprise-wide control over corporate machines. The approach and tool described in this study provide an enterprise-wide management facility for controlling the software that is allowed to run on users' machines. It returns control over information assets back to the system administrative staff and enables enforcement of corporate policies with regard to software. Bear in mind that because the control mechanism is based on actually running the software, it does not prevent unauthorized software installations, it merely prevents their execution. In addition to preventing unauthorized software from running, the authors believe that it is effective in stopping new and unknown malicious software executables from causing damage.

This method tends to virtually prevent unauthorized user from executing the application. It only achieves the goals within a closed environment. This method can not be applied on broadly used desktop computers.

In 2001, H. Chang and M. J. Attallah (Chang and Attalah, 2001) proposed a system called Protecting Software Codes by Guards. In this system, small pieces of code or guards are inserted throughout the code during compilation. Each guard is responsible for check-summing a particular piece of code. If tampering is detected, a special kind of repair guard is called. This approach is based on a distributed scheme, in which

protection and tamper-resistance of program code is achieved not by a single security module but by a network of smaller security units that work together in the program. These security units, or guards, can be programmed to do certain tasks for example check-summing the program code and a network of them can reinforce the protection of each other by creating mutual-protection.

The system does not by itself prevent unauthorized use of the program. It must be accompanied with another method to achieve that. This method is designed to prevent crackers from cracking the software. This method eliminates a single point to failure. However, it not only increases code size, but is also vulnerable to code analysis which can remove the guards before execution. Most runtime tools have a narrow focus on a particular security violation, although they take less of a performance penalty as compared to the static analysis tools.

In 2006, Olga Gelbart, Bhagirath Narahari and Rahul Simha (Gelbart, Narahari, and Simha, 2006) proposed a system called A Secure Program Execution Environment tool using code integrity checking. With the growing number of successful computer attacks, especially those using the internet and exploiting software vulnerabilities, software protection has become an important issue in computer security. This system is for software integrity protection and authentication and presents performance results. The system architecture utilizes key components from the compilation process as well as operating system support to provide static verification of executables. Code integrity checking is performed by means of a hierarchical hashing scheme, which not only detects changes but also efficiently isolates them. This scheme provides a higher level of protection against code injection or modification than a simple chaining of the program blocks. As an additional benefit, it also provides forensic information in case of a verification failure by providing the user with information about which part of the program has been modified. The SPEE tool is designed to function as part of the operating system kernel in order to provide a trusted computing system. This system combines concepts from compilers, operating systems and watermarking to provide code verification and authentication thereby preventing code tampering attacks. In addition, it provides forensic information to the user about what exact part of the code has been attacked. The tool can be integrated into the operating system kernel, thus making it possible to perform software verification at the system level.

The system contributes to the creation of a trusted computing system. However, it is extremely complex, and bounds the user to a specific Linux-based environment where the kernel is modified. It can not be applied on commercial applications.

Chapter Two

Software Piracy and Software Protection

In this chapter, we introduce software protection and its corresponding classes. In Understanding the effect of the Piracy introduced in this thesis, it is important to have an overview of the area of software protection. Software protection is concerned with making a program secure against reverse engineering and modification. This can be achieved through a number of different methods.

Some commonly used Technical protection methods for both software and other digital content are briefly discussed in this chapter.

In the meantime and to have a complete picture on software protection we would exam software cracking ideas and finally what the most cracking techniques used recently with a specific availability for each technique.

2.1 Technical Protection Methods

2.1.1 Media-Based Protections

These have been around since the 1980's. The media on which the copyrighted material is shipped, contains several specific features that allow verification of the authenticity of the media. In software distributions, the program checks if these features are present, whenever the program is executed. Since the 1980's much progress has been made in this field and nowadays Media-based protection is the primary copy protection used in the gaming industry. Media-based protections range from specific 'bad sectors' on floppy disks, to advanced (and expensive) low-level wrapping techniques for protection of executables and libraries using byte code and cryptography on DVDs (Starforce, 2007).

2.1.2 Serial-Based Protections

Using product serial numbers is one of the most common ways to verify the authenticity of legitimate users. The concept is simple: the author provides legitimate users with a serial, which is then checked by the program using a secret validation algorithm.

The boom of online available applications has boosted the popularity of the serial number protections enormously. To both software authors and end-users the scheme offers high flexibility and is relatively user-friendly. Smaller developers especially

benefit from these features: the scheme allows the end-user to download the program free of charge. The user can try the program and, if convinced of the program's quality, buy it by an act as simple as an online registration procedure. The author has the opportunity to market his program at low distribution costs, because no physical media is required. Most of the time the scheme is implemented by using certain restrictions on the freely available software in order to encourage registration. These include: time limits, crippled features, advertising and nag screens (for example a message being displayed every time the program starts) (Starforce, 2007).

The scheme, however, is not exclusively used for online distributions. In fact, it was originally used in over-the-counter software. During installation of the software, the installer asks the user to insert the serial, if it is invalid the installation process terminates. Usually such a serial is printed on something bundled with the software. In that case the key itself has a specific structure that allows a built-in key verification algorithm to decide on the user legitimacy.

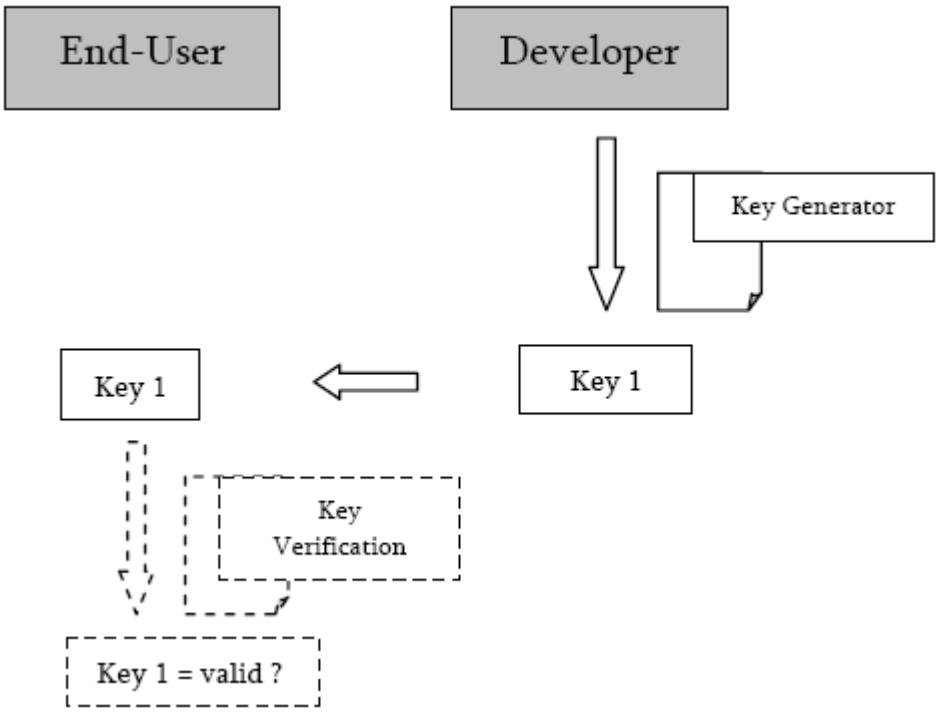

Figure 2-1 Non parameter-based verification scheme

In applications that can be registered online, the serial can be of a specific structure and use the described scheme in figure 2.1 or it can be implemented as a user parameter-

based verification scheme as in figure 2.2. In the latter case the serial is dependent on some of the user's parameters, like for instance his name. To register an application, the user then contacts the author by sending him for his name and the author provides the user with a key, created on the basis of the user's parameters. This serial was generated using the vendor's private key-generating algorithm, also present in the software (Starforce, 2007).

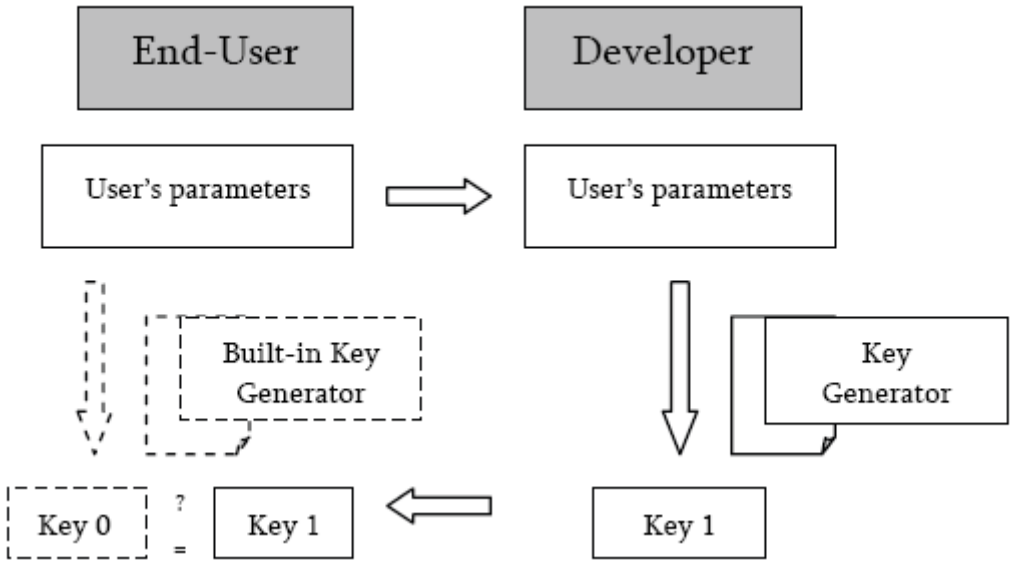

Figure 2-2 User parameter-based verification scheme

When the user enters his parameters and the key in the software's registration box, the program calculates the key by running the user's parameters through the built-in key generator and then compares the entered key with the one calculated in the background. When these two values match, the registration is successful.

It should be mentioned already that this protection, although flexible and user-friendly, has an inherent security risk, since the verification process includes generating the correct key on the end-user's machine.

In the case of parameter-based implementations, the verification algorithm can often be reversed to create a valid serial. Another weakness of serial-based protections is that there is no mechanism that prevents a same key to be used on different software installations, allowing users to share keys.

2.1.3 Hardware-Based Protections

In this scheme a tamper-proof, non software-based component is used to authenticate the running software. The customer attaches this dongle to his system through its external interface.

The safest approach is to have the dongle contain a part of the program that is required to run it. This often comes down to encryption of some kind. The dongle then contains a key to decrypt the program when running it. If this key is passed on to the system to let the CPU perform the decryption (this is how dongles used to work), this key can be easily intercepted and used to bypass the protection. That is the reason why, nowadays, most dongle manufacturers implement the decryption engine inside the dongle (using hardware-based decryption solutions). However, regardless of which implementation is used, the program still has to pass in an unencrypted form though processor and memory, which inevitably allows copying unprotected program code from memory to a storage device. This process is called dumping. Like this the whole program's code can be obtained, eventually leading to an unprotected copy of the software. Although hardware-based protections offer significantly stronger security compared to serial-based schemes, they fall short in other fields. Since a dongle needs to be installed on the user's system or network, the dongle approach cannot be considered user-friendly or end-user transparent. Flexibility is not one of its strongest points either. If a licensing model with free downloads of a limited edition of the program were employed, the protection scheme would not allow the same installed program to be turned into a fully functional one. Furthermore, every customer must be supplied with the mandatory hardware component, which adds extra costs to both shipment and production. Therefore, dongles are almost exclusively used to protect expensive, professional software packages (Eset, 2007).

2.1.4 Software as a Service

In this model the software runs on servers maintained and owned by the vendor of the software, the user has to be permanently connected to the Internet in order to use the program.

While this model provides excellent protection, the requirement of permanent connectivity is a serious drawback (nowadays mainly for security reasons). This makes the scheme, in its strictest form, not suitable for a variety of programs. However, a less

stringent variant of the software as a service-model has successfully been used by several update-reliant programs such as virus-scanners and other security related software. In that case a user must authenticate himself to the vendor's update-servers in order to obtain the updates.

A model that merely requires the user to authenticate his self every time he wants to run the program on the system cannot be considered software as a service, because the actual binary is executed on the user's system, which entails a less secure protection (Eset, 2007).

2.1.5 Digital Rights Management

Digital Rights Management (DRM) is an umbrella term for techniques that try to control the flow of digital copyrighted material. DRM is a developing branch of anti-piracy models that focus on controlling the flow of copyrighted media files.

These techniques for content protection rely on cryptographic techniques in which the decryption key should remain hidden to (illegitimate) users. Since the key is always required to enjoy the protected content, the main issue for DRM is how to hide the key from the users on an un-trusted and open system. The actual security code that performs decryption is however not present in the media files themselves, but in the player that is used.

The flexibility of this scheme allows vendors to limit the time during which the user can enjoy the copyrighted content by only supplying the player with the file's decryption key if the request is legitimate. However, for this feature a user requires an internet connection (Schneier, 2005).

2.1.6 Obfuscation

Obfuscation for software protection is useful in several areas. Obfuscation attempts to make the attackers job of understanding an application infeasible by protecting primarily against static analysis. Obfuscation gains its strength by combining a number of heuristics and algorithms which are designed to hide the true purpose and function of the machine language code. This hinders the attacker's ability to gain a high level understanding of program flow. The high level understanding is useful in extracting critical algorithms or sections of code from an application (for use in applications the attacker may be developing). A high level of understanding is also useful when the attacker wishes to modify the application for their own gain. For example, disabling

copy protection on digital rights management programs to allow distribution of copyrighted material (Collberg, et al., 2007).

2.1.7 Software Diversity

Most often, all copies of a specific application are made to look identical. The application is compiled once from pristine source and that compiled version is copied onto the distribution media. All customers who purchase the software product receive the same bit string encoding which is the application (variations in licensing are accounted for with license keys). In every copy of the application being identical, an attacker needs only break one copy of the application and distribute their break in order for all copies to be circumvented. Often, the crack itself can be distributed as a patch to the application, being much smaller than a complete copy. If software diversity was employed, it may be infeasible for the attacker to create a generic patch which can be applied to all systems running the application. This use of software diversity is very similar to that used to protect against wide exploitation of vulnerable code in protecting hosts against attack. Instead of the attacker distributing a patch, they would have to distribute their entire copy of the application. By removing the ability to create a small generic patch against all copies of an application, the costs of distributing a "cracked" application increase dramatically (for example a crack disabling a check for the original CD in the drive can no longer be distributed simply as a patch to the original application). Related to software diversity is fingerprinting. In our previous example, fingerprinting could be used to track the source of the attacked application. Fingerprinting is one form of software diversity which can be employed by an application developer, allowing them to track each copy of an application. This allows the company developing an application to pinpoint the copy of their application the attacker used to develop a crack. Legal action may then be possible against the attacker, if the attacker can be identified or located. Because of the legal recourse possible through fingerprinting, it has been proposed as a technique to deter software piracy (Eliam, 2005).

2.2 Software Cracking

In the field of computer science, software cracking is the action of defeating, circumventing or eliminating any kind of copy protection technique to obtain the application's full functionality for an illegitimate user. This is generally achieved by

posing as a legitimate user or by modification of the application's security code (Main, and Oorschot, 2003).

The process of devising an attack on an application with the intention of defeating the program's security code typically follows a specific pattern. This Section will discuss the consecutive steps in general to have an idea on how the attacking pattern work and then consider all these facts in our proposed scheme.

2.2.1 Analysis

This is usually the first step in an attack on an application's security code. The binary information which makes up the executable is transformed into a more humanly comprehensible format. In this representation of the binary data the attacker tries to develop an understanding on what measures have to be taken to bypass the built-in security mechanisms. However, sometimes the analysis step itself is the attacker's final goal. For example, if an attacker wants to adopt a certain algorithm used in the target application, he can extract it from the binary and steal the intellectual property by implementing it in his own program.

Binary analysis can be performed by using either static or dynamic analysis tools.

Static analysis decodes the binary to its corresponding assembly representation or high level language. Tools that decode the binary to assembly, and perform the reverse operation of assemblers, are called disassemblers; when a higher-level language is the target they are referred to as decompilers. Simply put, a disassembler is a translation program. It translates bytes into strings without executing the program. Therefore, static analysis is often referred to as offline code analysis or dead-listing. This poses some interesting problems in the field of disassembly (Linn, and Debray, 2003). The structure of disassembler is illustrated in figure 2.3.

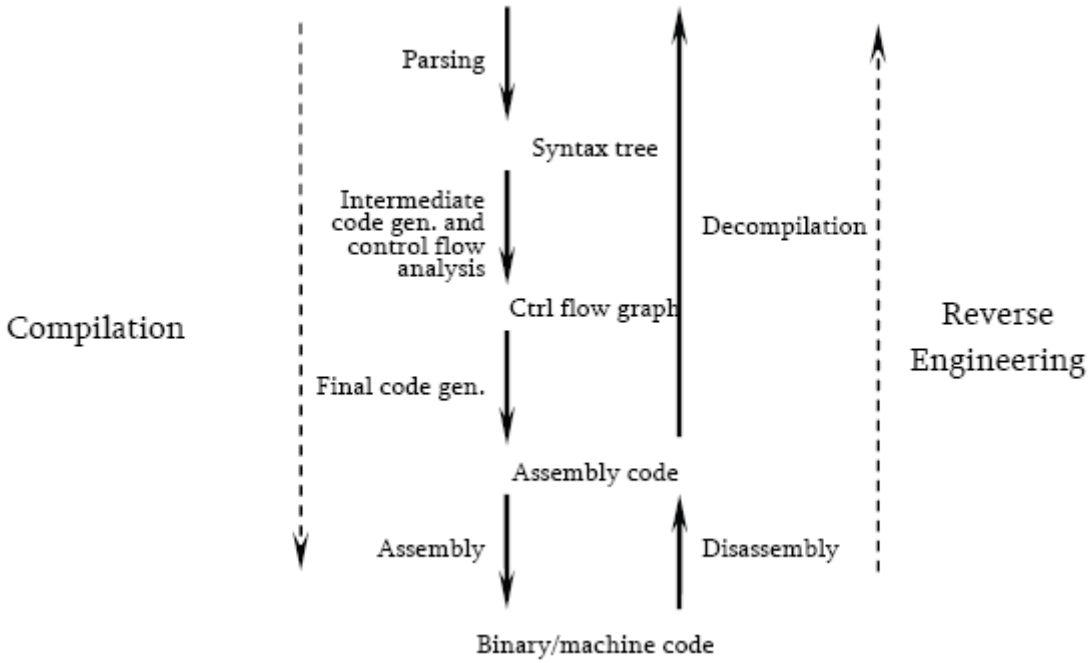

Figure 2-3 Compilation and reverse engineering flowchart

Compared to static analysis, dynamic analysis offers considerable advantages. Dynamic analysis, which is performed on executing code, enables the attacker to trace values through the application and to develop a better understanding of the program's data flow. However, dynamic analysis of a binary may be more time consuming and requires a platform that is able to run the target application. Dynamic analysis is typically achieved by attaching a debugger to an application. To get a human readable code listing, a good debugger requires a powerful dissembler. For a cracker, being able to view the code clearly is critical. For example, a debugger that provides cross-references that reveal the control flow over branch and call instructions is a great help in analyzing the application. It is also important that the built-in dissembler can be manually controlled and adjusted for successful data/code separation.

Successful dynamic analysis does always provide an attacker with the application's entry point, imports, and valuable data and control flow information.

Using both static and dynamic analysis, the attacker tries to locate critical points in the targets security code. A popular and effective way of finding such points is by placing debugger breakpoints on certain modules that are commonly used to provide graphical output (such as a message-box saying that the entered key is incorrect) or to retrieve user data. The program then breaks near the location of the registration code. Of course,

every programmer, who does a little research on protecting his security code will know how to avoid calling these modules directly.

If these calls are disguised or are hard to locate, an attacker will try to attach a debugger to the program when the running application is at such a critical decision point, potentially giving away its location.

2.2.2 Tampering

This step consists of the actual modification of the binary that intentionally results in the failure of the target's security code. Unfortunately, in today's software, most current copy protection techniques boil down to a yes-or-no decision when it comes down to granting the user full privileges or not. The analysis of the binary, which is usually by far the most time consuming part of the cracking process, has the aim of locating such specific decision points.

A decision point is typically created by use of an if-else structure.

```
if (serial%3 ==2 )
{
    printf("This is the correct serial\n");
    execute_program();
}
else
    printf("This is the wrong serial\n");
```

Figure 2-4 High-level creation of a critical decision point

An actual tampering process is depicted in figure 2.5. The target is a custom made program that makes use of the above if-else structure.

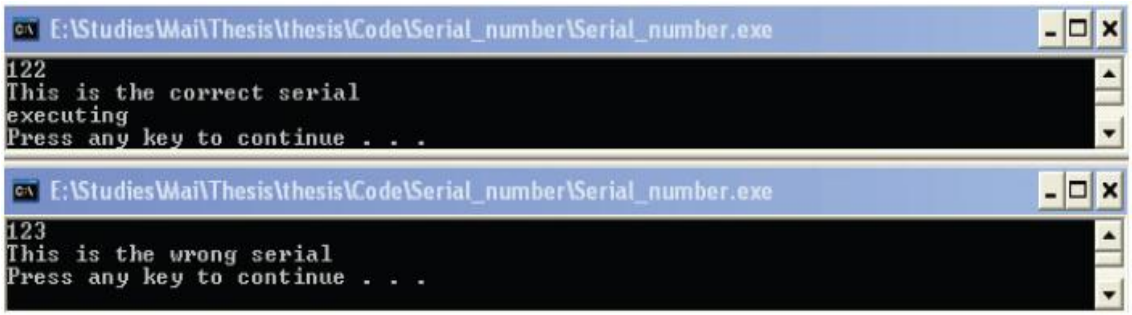

Figure 2-5 Normal program functionality

2.2.3 Automation

This involves the development of software to automatically apply the modifications described and illustrated to multiple installed copies of the target application (Schneier, 1996).

2.2.4 Distribution

The distribution of the automated attack allows other users with limited background in computer science to bypass the application's security code. The distribution of an attack is the most harmful thing that can happen for a software company. This combination is one of the major causes of the economical damage inflicted by piracy. If there was no way of automating and spreading an attack, the only illegitimate users would be the crackers themselves. It lies beyond doubt that software piracy would be drastically reduced to, at least for software vendors, manageable levels.

2.3 Cracks Classification

This Section discusses several kinds of commonly distributed, both manual and automated attacks, devised to bypass an application's security code. The following cracks can be encountered by a user who is planning to crack an application.

2.3.1 Leaked Serials

Of commonly encountered cracks, this is the least technical one. In a sense, it might not even be called a crack. However, it can be considered a form of a manual attack, so it will be briefly discussed. The attack consists of a license file or text file (often .nfo-files) containing a valid key. The original key or license can be obtained through legal means and shared by the legitimate user himself or it can be filched from him or the vendor. Some crackers succeed in 'fishing' serials from applications, by thoroughly analyzing the serial verification algorithm.

2.3.2 Key Generators

Key Generators (or 'keygens') are a quite efficient means of circumventing an application's security code. The concept is simple; just like the vendor the attacker is in possession of a tool that generates valid serial numbers, the key generator, which enables him to register the target application. By registering as a legitimate user would an attacker gains full privileges on the program without altering the program's binary.

The application of this attack does not modify the binary and thus will not be detected by checking the file for modifications. In fact, an illegitimate user flies completely under the radar if the application does not contain any online functionality. If it does, the vendor can keep track of the serials that were bundled with copies during manufacturing or that were supplied to end-users, and allow only these serials to be eligible for registration (for example in the online gaming industry).

The development of a key generator consists only of the first and third step of the piracy chain, namely analysis and automation. During analysis, the cracker tries to locate the algorithm that verifies the inserted key. Figure 2.6 shows a sample key generator.

Figure 2-6 A typical non parameter-based key generators

2.3.3 Backup-Media

A ‘backup’-medium of a certain application is an attempt to circumvent media-based protections by making a 1-on-1 copy of the original medium. The intention is to copy the medium on the lowest possible level to incorporate and replicate the distinctive features applied to differentiate between a legitimate and an illegitimate copy. The copy is made by special tools that try to simulate the techniques of introducing these distinctive features during manufacturing, using common optical medium burning technology. While some physical backups are sold as official ones, the majority of copied applications are spread in digital format, called a disk-image.

However, most companies that have the resources to invest in media-based copy protection also include a serial-based protection mechanism in their products. That

forces the attacker to include a working serial or key generator in his distribution, or to crack the application's installer on the disk-image directly (AlcoholSoft, 2007).

2.3.4 No-CD/DVD

Another approach to defeating medium-based copy protections is to perform an attack on the security code in the target application that is responsible for verifying the authenticity of the medium. Usually the attacker removes the need for the medium to be inserted altogether.

2.3.5 Patches

Patches are simple programs that contain the modifications that need to be applied to the binary in order to incapacitate the target's protection mechanisms. Patches usually have a very small file size and just like key generator templates, cracking-groups use their own assembly templates to develop them. Unlike key generators however, patches can be used to crack any kind of protection. The only parameters required to create a patch using a template are the target address (for example address of some critical point that needs to be modified), the data to overwrite it with and usually the original data, for verification or restoration purposes. Figure 2.7 shows a sample patch program.

Figure 2-7 A patch

2.3.6 Fixed EXEs

Is a truncation of 'fixed executable' and refers to a cracked binary. In a sense No-CD/DVD-cracks can be considered a sub-division of the cracked binaries, since the distributed file is also the directly modified binary. If properly cracked the use of a

modified binary can bypass all current copy protections. The only thing that limits its widespread availability is its significant file size. That is why fixed executables are used mainly to bypass highly complex protection schemes that require so many modifications the attacker is unwilling to develop a patch to apply them (Li, 2004).

2.3.7 Loaders

Loaders can rightfully be considered the most sophisticated of all cracks. Basically a loader is a program responsible for loading files in memory by properly relocating them to their reserved memory space and preparing them for execution. Every program a user executes on an operating system is allocated to memory by the operating system's loader.

The loader is initialized by supplying it with the attacker's data, the location where this data needs to be patched, and often, as a means of verification, the original data. Once the target program is loaded, the loader keeps monitoring it, and has the ability to suspend and resume the target application's threads by using Suspend Thread and Resume Thread (Shub-Nigurrath, 2005).

Loaders can be made to be very small and are an effective means to circumvent advanced software protections. Fortunately, coding loaders is quite complicated and few crackers venture in doing so (see Section 2.4).

2.4 Specific Availability

In order to target a large segment of the above discussed attack with our proposed protection mechanism it is of great importance to know the composition of the available crack population.

Analysis of 124 cracks, randomly downloaded from several crack-providing websites, resulted in the statistic shown in figure 2.8.

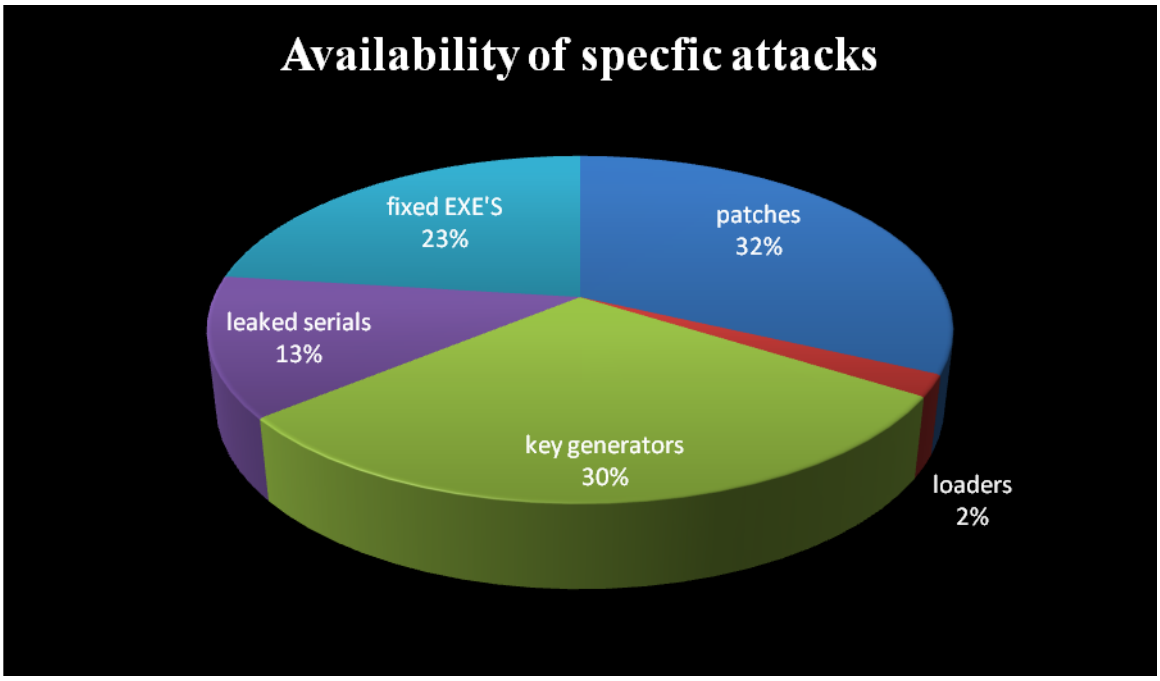

Figure 2-8 Graphs depicting the availability of cracks

A first interesting thing to notice is the fact that among 124 cracks only two loaders were encountered. Secondly, patching seems to be the most popular approach to cracking an application. Only sixteen leaked or fished serials were encountered, while key generators represented 30% of the population.

Chapter Three

The Proposed System

In this chapter, firstly, the used algorithms will be explained. Secondly, the used terms will be defined and explained. After that, the system parts will be explained in terms of process execution.

1) Used algorithms

A- Improved version of original RSA Public Key Encryption Scheme

B- Triple DES

C- International Standard Copy Number

D- Zero Knowledge Authentication

E- MD5 hash

2) Codes definition

A- Machine ID

B- Installation ID

C- Copy ID

D- IP Address

E- Activation Code

3) Protecting EXE files

A- Protected file structure

B- How to protect the file(file.exe)

C- Checksum

4) How to Obtain Hardware information for each client

5) Authentication

6) Customer tracking system

A- Database

B- Web Site

C- Web Service

7) How the application work

(1) USED ALGORITHMS:

A- Improved version of original RSA Public Key Encryption Scheme

In 1976 Diffie-Hellmam creates the first revolutionary research in public key cryptography via presented a new idea in cryptography and to challenge experts to generate cryptography algorithms that faced the requirements for public key cryptosystems. However, the first reaction to the challenge is introduced in 1978 by RSA. The RSA scheme is a block cipher in which the original message and cipher message are integer values in the interval $[0..n-1]$ where n a composite modulus. In this developed scheme the original message and cipher message from the general linear group of $h \times h$ matrices over z_n indicated by $g(h, z_n)$ and the original message indicated by m. The process of encryption a message m is indicated via c. A private key is required to disclose the original message from the cipher message. However, the message in RSA scheme is encrypted in blocks after divide it to blocks, every block must convert to a value smaller than the modulus n. The intractability of the RSA assumption forms its security. The RSA assumption is the difficulty of solving the integer modulus n which is a product of two distinct odd large primes p and q with an assistance of another public key e and an integer cipher text c. In other words, the RSA difficulty is that of solving e^{th} roots mod a composite modulus n. The conditions determined the modulus n and the public key e are to guarantee that for every integer $c \in (0,1,\ldots,n-1)$ there is just one $m \in (0,1,\ldots,n-1)$ where $m^e = c \bmod n$. However, the RSA scheme is the most employed public key encryption compared with the other schemes. It can be employed for both encryption and digital signature schemes.

The RSA scheme is as follows:

Key generation algorithm

To generate the keys entity A must do the following:

1. Randomly and secretly choose two large prime numbers p and q with equally likely.
2. Compute the modulus $n = p * q$.
3. Compute $\theta(n) = (p-1)(q-1)$
4. Select random integer $e, 1 < e < n$ where $\gcd(e, \theta) = 1$

5. Use Baghdad method (Aboud, 2004) to compute the unique decrypted key $d, 1 < d < \theta(n)$ where $e * d \equiv 1 \bmod \theta(n)$
6. Determine entity A public and private key. The pair (d, θ) is the private key. While the pair (n, e) is the public key.

Public key encryption algorithm

Entity B encrypts a message m for entity A which entity A decrypts.

Encryption: entity B should do the following:

- Obtain entity A 's public key (n, e).
- Represent the message m as an integer in the interval $[0..n-1]$
- Compute $c = m^e \bmod n$
- Send the encrypted message c to entity A.

Decryption: To recover the message m from the cipher text c. Entity A must do the following:

- Obtain the cipher text c from entity B
- Recover the message $m = c^d \bmod n$

the developed scheme will widen the RSA scheme to a scheme that employs the general linear group of degree h is the set of hxh invertible matrices with ordinary matrix multiplication where n is a product of two large prime numbers such as with the example of general RSA scheme. The integer is co-prime with n form a group under multiplication mod n of order $GL(h, z_n)$, inverse square matrices of rank h on the ring of integer mod n will generate a group of order to this group, and this is unknown in the general scheme. But, in the general scheme where n is a product of two distinct prime numbers we can find the order of this group by the following theorem (Aboud, and AL-Fayoumi, 2008):

Theorem: Assume that $n = p * q$ is the product of two large prime numbers, and suppose that $GL(h, z_n)$ is the general linear group of $h \times h$ matrices over z_n. Then

$$/g/ = (p^h - 1)(p^h - p)...(p^h - p^{h-1}) * (q^h - 1)(q^h - q)...(q^h - q^{h-1})$$

Proof: Each matrix $x \in g$ decreased to two matrices x_p and x_q such that x_p and x_q are $h \times h$ matrices on the members z_p and z_q such that $x_p = x \bmod p$, $x_q = x \bmod q$. Actually, a mapping:

$f : g(h,n) \to g(h,p) \oplus g(h,q)$, is the identical function.

$|GL(h, z_n)|$

$= |GL(h, z_e)|$

$|GL(h, z_q)|$

Key generation algorithm

To generate the keys entity A must do the following:

1. Randomly chooses two large prime numbers p and q.
2. Compute the modulus $n = p * q$.
3. Compute $g = (h, z_n)$
4. Choose a random integer e where $\gcd(e, g) = 1$
5. Compute the inverse d where $ed \equiv 1 \bmod g$
6. Determine the entity A public and private key. The pair (g, d) is the private key. While the pair (n, e) is the public key.

Public key encryption algorithm

Entity B encrypts a message m for entity A which entity A decrypts.

Encryption: entity B should do the following:

- Obtain entity A public key (n, e).
- Represent the message m as a $h \times h$ matrix x
- Compute $h \times h$ matrix $c = m^e \bmod n$
- Send the encrypted message c to entity A.

Decryption: to recover the message m from the cipher text c. Entity A must do the following:

- Obtain the cipher text c from entity B
- Recover the message $m = c^d \bmod n$

Example

Key generation: suppose that entity A selects randomly the two prime numbers $p = 43$ and $q = 47$. Then find the modulus $n = p*q = 2021$ and compute

g = (2, 2021) = $(43^2-1)*(43^2-43)*(47^2-1)*(47^2-47)$ = 1848*1806*2208*2162=15932153115648

Then entity A picks $e = 17$ and using the Baghdad method to find d = 14994967638257 where $e*d \equiv 1 \bmod g$. So, $A's$ public key is the pair $(n = 2021, e = 17)$ while $A's$ private key is (g = 15932153115648, d = 14994967638257)

Encryption: Suppose entity B obtain $A's$ public key $(n = 2021, e = 17)$ and he determines a message $m = 741$ to be encrypted and finds $c = 741^{17} \bmod 2021$ then send $c = 1471$ to entity A.

Decryption: To recover and obtain the original message m entity A should the following:

- obtain the cipher text $c = 1471$ from entity B,
- then recover the original message m = (147114994967638257) mod 2021=741

Advantages of the Proposed Scheme

1. The proposed scheme can be used with Hill cipher method to obtain more intractable encryption system. Also, the suggested scheme can be employed using a subgroup instead of a full value of $g = (h, n)$, since the $\gcd(e, g) = 1$. In this case the suggested scheme will give more flexibility to entity to use more than one technique.
2. The intractability of the integer factoring of the modulus n in the propose scheme stays as same as in the RSA scheme.
3. The proposed scheme can be used as a digital signature scheme by inserted in the matrix x as an item.

B- Triple DES

In cryptography, Triple DES (Grabbe, 1997) is a block cipher formed from the Data Encryption Standard (DES) cipher by using it three times.

When it was found that a 56-bit key of DES is not enough to guard against brute force attacks, TDES was chosen as a simple way to enlarge the key space without a need to switch to a new algorithm. The use of three steps is essential to prevent meet-in-the-middle attacks that are effective against double DES encryption. Note that DES is not a group; if it were one, the TDES construction would be equivalent to a single DES operation and no more secure.

The simplest variant of TDES operates as follows: DES (k3; DES (k2; DES (k1; M))), where M is the message block to be encrypted and k1, k2, and k3 are DES keys.

DES works on bits, or binary numbers--the 0s and 1s common to digital computers. Each group of four bits makes up a hexadecimal, or base 16, number. Binary "0001" is equal to the hexadecimal number "1", binary "1000" is equal to the hexadecimal number "8", "1001" is equal to the hexadecimal number "9", "1010" is equal to the hexadecimal number "A", and "1111" is equal to the hexadecimal number "F".

DES works by encrypting groups of 64 message bits, which is the same as 16 hexadecimal numbers. To do the encryption, DES uses "keys" where are also *apparently* 16 hexadecimal numbers long, or *apparently* 64 bits long. However, every 8th key bit is ignored in the DES algorithm, so that the effective key size is 56 bits. But, in any case, 64 bits (16 hexadecimal digits) is the round number upon which DES is organized.

For example, if we take the plaintext message "8787878787878787", and encrypt it with the DES key "0E329232EA6D0D73", we end up with the cipher text "0000000000000000". If the cipher text is decrypted with the same secret DES key "0E329232EA6D0D73", the result is the original plaintext "8787878787878787".

This example is neat and orderly because our plaintext was exactly 64 bits long. The same would be true if the plaintext happened to be a multiple of 64 bits. But most messages will not fall into this category. They will not be an exact multiple of 64 bits (that is, an exact multiple of 16 hexadecimal numbers).

For example, take the message "Your lips are smoother than Vaseline". This plaintext message is 38 bytes (76 hexadecimal digits) long. So this message must be padded with some extra bytes at the tail end for the encryption. Once the encrypted message has been decrypted, these extra bytes are thrown away. There are, of course, different padding schemes--different ways to add extra bytes. Here we will just add 0s at the end, so that the total message is a multiple of 8 bytes (or 16 hexadecimal digits, or 64 bits).

The plaintext message "Your lips are smoother than Vaseline" is, in hexadecimal,

"596F7572206C6970 732061726520736D 6F6F746865722074 68616E2076617365 6C696E650D0A".

(Note here that the first 72 hexadecimal digits represent the English message, while "0D" is hexadecimal for Carriage Return, and "0A" is hexadecimal for Line Feed, showing that the message file has terminated.) We then pad this message with some 0s on the end, to get a total of 80 hexadecimal digits:

"596F7572206C6970 732061726520736D 6F6F746865722074 68616E2076617365 6C696E650D0A0000".

If we then encrypt this plaintext message 64 bits (16 hexadecimal digits) at a time, using the same DES key "0E329232EA6D0D73" as before, we get the cipher text:

"C0999FDDE378D7ED 727DA00BCA5A84EE 47F269A4D6438190 9DD52F78F5358499 828AC9B453E0E653".

This is the secret code that can be transmitted or stored. Decrypting the cipher text restores the original message "Your lips are smoother than Vaseline".

How DES Works in Detail

DES is a block cipher--meaning it operates on plaintext blocks of a given size (64-bits) and returns cipher text blocks of the same size. Thus DES results in a permutation among the 2^{64} possible arrangements of 64 bits, each of which may be either 0 or 1. Each block of 64 bits is divided into two blocks of 32 bits each, a left half block L and a right half R. (This division is only used in certain operations.)

Example: Let M be the plain text message M = 0123456789ABCDEF, where M is in hexadecimal (base 16) format. Rewriting M in binary format, we get the 64-bit block of text:

M = 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111

L = 0000 0001 0010 0011 0100 0101 0110 0111

R = 1000 1001 1010 1011 1100 1101 1110 1111

The first bit of *M* is "0". The last bit is "1". We read from left to right.

DES operates on the 64-bit blocks using *key* sizes of 56- bits. The keys are actually stored as being 64 bits long, but every 8th bit in the key is not used (such as bits numbered 8, 16, 24, 32, 40, 48, 56, and 64). However, we will nevertheless number the bits from 1 to 64, going left to right, in the following calculations. But, as you will see, the eight bits just mentioned get eliminated when we create sub keys.

Example: Let *K* be the hexadecimal key *K* = 133457799BBCDFF1. This gives us as the binary key (setting 1 = 0001, 3 = 0011, etc., and grouping together every eight bits, of which the last one in each group will be unused):

K = 00010011 00110100 01010111 01111001 10011011 10111100 11011111 11110001

The DES algorithm uses the following steps:

Step 1: Create 16 sub keys, each of which is 48-bits long.

The 64-bit key is permuted according to the following table, (3.1) PC-1. Since the first entry in the table is "57", this means that the 57th bit of the original key *K* becomes the first bit of the permuted key *K*+. The 49th bit of the original key becomes the second bit of the permuted key. The 4th bit of the original key is the last bit of the permuted key. Note only 56 bits of the original key appear in the permuted key.

Table 3.1 DES - PC-1

PC-1

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

Example: From the original 64-bit key

K = 00010011 00110100 01010111 01111001 10011011 10111100 11011111 11110001

We get the 56-bit permutation

$K+$ = 1111000 0110011 0010101 0101111 0101010 1011001 1001111 0001111

Next, split this key into left and right halves, C_0 and D_0, where each half has 28 bits.

Example: From the permuted key $K+$, we get

C_0 = 1111000 0110011 0010101 0101111
D_0 = 0101010 1011001 1001111 0001111

With C_0 and D_0 defined, we now create sixteen blocks C_n and D_n, 1<=n<=16. Each pair of blocks C_n and D_n is formed from the previous pair C_{n-1} and D_{n-1}, respectively, for n = 1, 2... 16, using the following schedule of "left shifts" of the previous block. To do a left shift, move each bit one place to the left, except for the first bit, which is cycled to the end of the block. Table 3.2 shows the result.

Table 3.2 DES Shifts

Iteration Number	Number of Left Shifts
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

This means, for example, C_3 and D_3 are obtained from C_2 and D_2, respectively, by two left shifts, and C_{16} and D_{16} are obtained from C_{15} and D_{15}, respectively, by one left shift. In all cases, by a single left shift is meant a rotation of the bits one place to the left, so

that after one left shift the bits in the 28 positions are the bits that were previously in positions 2, 3,..., 28, 1.

Example: From original pair C_0 and D_0 we obtain:

C_0 = 1111000011001100101010101111

D_0 = 0101010101100110011110001111

C_1 = 1110000110011001010101011111

D_1 = 1010101011001100111100011110

C_2 = 1100001100110010101010111111

D_2 = 0101010110011001111000111101

C_3 = 0000110011001010101011111111

D_3 = 0101011001100111100011110101

C_4 = 0011001100101010101111111100

D_4 = 0101100110011110001111010101

C_5 = 1100110010101010111111110000

D_5 = 0110011001111000111101010101

C_6 = 0011001010101011111111000011

D_6 = 1001100111100011110101010101

C_7 = 1100101010101111111100001100

D_7 = 0110011110001111010101010110

C_8 = 0010101010111111110000110011

D_8 = 1001111000111101010101011001

C_9 = 0101010101111111100001100110

D_9 = 0011110001111010101010110011

C_{10} = 0101010111111110000110011001

D_{10} = 1111000111101010101011001100

C_{11} = 0101011111111000011001100101

D_{11} = 1100011110101010101100110011

C_{12} = 0101111111100001100110010101

D_{12} = 0001111010101010110011001111

C_{13} = 01111111100001100110010101010

D_{13} = 01111010101010110011001111100

C_{14} = 11111110000110011001010101010

D_{14} = 11101010101011001100111100001

C_{15} = 11111000011001100101010101111

D_{15} = 10101010101100110011111000111

C_{16} = 11110000110011001010101011111

D_{16} = 01010101011001100111100011111

We now form the keys K_n, for 1<=n<=16, by applying the following permutation table to each of the concatenated pairs C_nD_n. Each pair has 56 bits, but PC-2 (Table 3.3) only uses 48 of these.

Table 3.3 DES - PC-2

PC-2

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Therefore, the first bit of K_n is the 14th bit of C_nD_n, the second bit the 17th, and so on, ending with the 48th bit of K_n being the 32th bit of C_nD_n.

Example: For the first key we have C_1D_1 = 1110000 1100110 0101010 1011111 1010101 0110011 0011110 0011110

which, after we apply the permutation PC-2, becomes

K_1 = 000110 110000 001011 101111 111111 000111 000001 110010

For the other keys we have

K_2 = 011110 011010 111011 011001 110110 111100 100111 100101

K_3 = 010101 011111 110010 001010 010000 101100 111110 011001

K_4 = 011100 101010 110111 010110 110110 110011 010100 011101

K_5 = 011111 001110 110000 000111 111010 110101 001110 101000

K_6 = 011000 111010 010100 111110 010100 000111 101100 101111

K_7 = 111011 001000 010010 110111 111101 100001 100010 111100

K_8 = 111101 111000 101000 111010 110000 010011 101111 111011

K_9 = 111000 001101 101111 101011 111011 011110 011110 000001

K_{10} = 101100 011111 001101 000111 101110 100100 011001 001111

K_{11} = 001000 010101 111111 010011 110111 101101 001110 000110

K_{12} = 011101 010111 000111 110101 100101 000110 011111 101001

K_{13} = 100101 111100 010111 010001 111110 101011 101001 000001

K_{14} = 010111 110100 001110 110111 111100 101110 011100 111010

K_{15} = 101111 111001 000110 001101 001111 010011 111100 001010

K_{16} = 110010 110011 110110 001011 000011 100001 011111 110101

So much for the sub keys, now we look at the message itself.

Step 2: Encode each 64-bit block of data.

There is an *initial permutation IP* of the 64 bits of the message data *M*. This rearranges the bits according to the following table, where the entries in the table show the new arrangement of the bits from their initial order. The 58th bit of *M* becomes the first bit of *IP*. The 50th bit of *M* becomes the second bit of *IP*. The 7th bit of *M* is the last bit of *IP*. As illustrated in table 3.4.

Table 3.4 DES -IP

IP

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Example: Applying the initial permutation to the block of text *M*, given previously, we get

M = 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111

IP = 1100 1100 0000 0000 1100 1100 1111 1111 1111 0000 1010 1010 1111 0000 1010 1010

Here the 58th bit of *M* is "1", which becomes the first bit of *IP*. The 50th bit of *M* is "1", which becomes the second bit of *IP*. The 7th bit of *M* is "0", which becomes the last bit of *IP*.

Next divide the permuted block *IP* into a left half L_0 of 32 bits, and a right half R_0 of 32 bits.

Example: From *IP*, we get L_0 and R_0

L_0 = 1100 1100 0000 0000 1100 1100 1111 1111
R_0 = 1111 0000 1010 1010 1111 0000 1010 1010

We now proceed through 16 iterations, for 1<=*n*<=16, using a function *f* which operates on two blocks--a data block of 32 bits and a key K_n of 48 bits--to produce a block of 32 bits. Let + denote XOR addition, (bit-by-bit addition modulo 2). Then for n going from 1 to 16 we calculate

$L_n = R_{n-1}$
$R_n = L_{n-1} + f(R_{n-1},K_n)$

This results in a final block, for $n = 16$, of $L_{16}R_{16}$. That is, in each iteration. We take the right 32 bits of the previous result and make them the left 32 bits of the current step. For the right 32 bits in the current step, we XOR the left 32 bits of the previous step with the calculation *f*

Example: For $n = 1$, we have

K_1 = 000110 110000 001011 101111 111111 000111 000001 110010
$L_1 = R_0$ = 1111 0000 1010 1010 1111 0000 1010 1010
$R_1 = L_0 + f(R_0,K_1)$

It remains to explain how the function *f* works. To calculate *f*, we first expand each block R_{n-1} from 32 bits to 48 bits. This is done by using a selection table that repeats some of the bits in R_{n-1} . We'll call the use of this selection table the function E. Thus E(R_{n-1}) has a 32 bit input block, and a 48 bit output block.

Let E be such that the 48 bits of its output, written as 8 blocks of 6 bits each, are obtained by selecting the bits in its inputs in order according to the table 3.5:

Table 3.5 DES- E Bit-Selection Table

E BIT-SELECTION TABLE

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Thus the first three bits of E(R_{n-1}) are the bits in positions 32, 1 and 2 of R_{n-1} while the last 2 bits of E(R_{n-1}) are the bits in positions 32 and 1.

Example: We calculate E(R_0) from R_0 as follows:

R_0 = 1111 0000 1010 1010 1111 0000 1010 1010
E(R_0) = 011110 100001 010101 010101 011110 100001 010101 010101

(Note that each block of 4 original bits has been expanded to a block of 6 output bits.)

Next in the *f* calculation, we XOR the output E(R_{n-1}) with the key K_n:

K_n + E(R_{n-1}).

Example: For K_1 , E(R_0), we have

K_1 = 000110 110000 001011 101111 111111 000111 000001 110010
E(R_0) = 011110 100001 010101 010101 011110 100001 010101 010101
K_1+E(R_0) = 011000 010001 011110 111010 100001 100110 010100 100111.

We have not yet finished calculating the function *f* . To this point we have expanded R_{n-1} from 32 bits to 48 bits, using the selection table, and XORed the result with the key K_n . We now have 48 bits, or eight groups of six bits.

We now do something strange with each group of six bits: we use them as addresses in tables called "S boxes". Each group of six bits will give us an address in a different *S* box. Located at that address will be a 4 bit number. This 4 bit number will replace the original 6 bits. The net result is that the eight groups of 6 bits are transformed into eight groups of 4 bits (the 4-bit outputs from the *S* boxes) for 32 bits total.

Write the previous result, which is 48 bits, in the form:

K_n + E(R_{n-1}) =$B_1B_2B_3B_4B_5B_6B_7B_8$,

where each B_i is a group of six bits. We now calculate

$S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8)$

where $S_i(B_i)$ refers to the output of the i-th S box.

To repeat, each of the functions *S1, S2,..., S8,* takes a 6-bit block as input and yields a 4-bit block as output. The table to determine S_1 is shown and explained in table 3.6.

Table 3.6 DES- S_1 Determination table

S1

Row No.	Column Number															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

If S_1 is the function defined in this table and B is a block of 6 bits, then $S_1(B)$ is determined as follows: The first and last bits of B represent in base 2 a number in the decimal range 0 to 3 (or binary 00 to 11). Let that number be i. The middle 4 bits of B represent in base 2 a number in the decimal range 0 to 15 (binary 0000 to 1111). Let that number be j. Look up in the table the number in the i-th row and j-th column. It is a number in the range 0 to 15 and is uniquely represented by a 4 bit block. That block is the output $S_1(B)$ of S_1 for the input B. For example, for input block B = 011011 the first bit is "0" and the last bit "1" giving 01 as the row. This is row 1. The middle four bits are "1101". This is the binary equivalent of decimal 13, so the column is column number 13. In row 1, column 13 appears 5. This determines the output; 5 is binary 0101, so that the output is 0101. Hence $S_1(011011) = 0101$.

The tables defining the functions $S_1,...,S_8$ are the following table 3.7:

Table 3.7 DES- The definitions for $S_1,...,S_8$

S1

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

S2

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

S3

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

S4

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

S5

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

S6

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

S7

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

S8

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

Example: For the first round, we obtain as the output of the eight *S* boxes:

K_1 + E(R_0) = 011000 010001 011110 111010 100001 100110 010100 100111.

$S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8)$ = 0101 1100 1000 0010 1011 0101 1001 0111

The final stage in the calculation of *f* is to do a permutation P of the S-box output to obtain the final value of *f*:

f = P($S_1(B_1)S_2(B_2)...S_8(B_8)$)

The permutation P is defined in the table 3.8. P yields a 32-bit output from a 32-bit input by permuting the bits of the input block.

Table 3.8 DES – P definition

P			
16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

Example: From the output of the eight *S* boxes:

$S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8)$ = 0101 1100 1000 0010 1011 0101 1001 0111

we get

f = 0010 0011 0100 1010 1010 1001 1011 1011

$R_1 = L_0 + f(R_0 , K_1)$

= 1100 1100 0000 0000 1100 1100 1111 1111
+ 0010 0011 0100 1010 1010 1001 1011 1011
= 1110 1111 0100 1010 0110 0101 0100 0100

In the next round, we will have $L_2 = R_1$, which is the block we just calculated, and then we must calculate $R_2 = L_1 + f(R_1, K_2)$, and so on for 16 rounds. At the end of the sixteenth round we have the blocks L_{16} and R_{16}. We then *reverse* the order of the two blocks into the 64-bit block

$R_{16}L_{16}$

and apply a final permutation IP^{-1} as defined by the table 3.9:

Table 3.9 DES- IP^{-1} Definition

IP^{-1}							
40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

That is, the output of the algorithm has bit 40 of the pre-output block as its first bit, bit 8 as its second bit, and so on, until bit 25 of the pre-output block is the last bit of the output.

Example: If we process all 16 blocks using the method defined previously, we get, on the 16th round,

L_{16} = 0100 0011 0100 0010 0011 0010 0011 0100
R_{16} = 0000 1010 0100 1100 1101 1001 1001 0101

We reverse the order of these two blocks and apply the final permutation to

$R_{16}L_{16}$ = 00001010 01001100 11011001 10010101 01000011 01000010 00110010 00110100

IP^{-1} = 10000101 11101000 00010011 01010100 00001111 00001010 10110100 00000101

In which in hexadecimal format is:

85E813540F0AB405.

This is the encrypted form of M = 0123456789ABCDEF: namely, C = 85E813540F0AB405 (www.aci.net/kalliste/des.htm).

Decryption is simply the inverse of encryption, following the same steps as above, but reversing the order in which the sub keys are applied.

Triple-DES

Triple-DES is just DES with two 56-bit keys applied. Given a plaintext message, the first key is used to DES- encrypt the message. The second key is used to DES-decrypt

the encrypted message. (Since the second key is not the right key, this decryption just scrambles the data further.) The twice-scrambled message is then encrypted again with the first key to yield the final cipher text. This three-step procedure is called triple-DES.

Triple-DES is just DES done three times with two keys used in a particular order. (Triple-DES can also be done with three separate keys instead of only two. In either case the resultant key space is about 2^{112}.)

C- International Standard Copy Number (ISCN)

The international standard computer number (ISCN) is an identifying number assigned to virtually every software copy. A new edition receives its own ISCN. It serves to uniquely identify the copy.

An ISCN for example has four parts:

1. Language country code
2. Manufacturer code
3. Copy number assigned by publisher
4. Check digit

For example, a total of 10 digits ISCN 0-387-95045-1 has language/county code 0, publisher code 387, copy number 95045 and a check digit 1. Table below lists some language country code.

0	English (UK, USA, NZ, Australia, Canada)
1	English (South Africa, Zimbabwe)
2	French (France, Belgium, Canada, Switzerland)
3	German (Germany, Austria, Switzerland)
4	Japan
5	USSR
6	China
7	India
8	Arabic (All Countries)

It's clear that widely used languages are assigned a short language country codes, thereby allowing for a long publisher code, while other countries and languages have been assigned long language country codes, so their publisher codes must be short. When the number space of language country code is exhausted, another code is assigned to the language country. For example you can give to Spain codes 81 and 93.

Check Digit calculation

The check digit is computed by multiplying the leftmost ISCN digit by 10, the next digit by 9, and so on up to the ninth digit from the left, which is multiplied by 2. The products are then added, and the check digit is determined as the smallest integer that when added to this weighted sum will make it a multiple of 11. The check digit is therefore in the interval [0, 10]. If it happens to be 10, it is replaced by the Roman numeral X in order to make it a single symbol.

If we denote the nine leftmost ISCN digits by d_1 through d_9 (from left to right), then the ISCN I is computed by first calculating the weighted sum:

$$T = (10d_1 + 9d_2 + 8d_3 + 7d_4 + 6d_5 + 5d_6 + 4d_7 + 3d_8 + 2d_9) \bmod 11$$

Notice that T is in the interval [0, 10] because of the use of the mod and then subtracting $I = 11 - T$. For example, given the nine digit 038795045, the two steps produce.

$$T = (10*0 + 9*3 + 8*8 + 7*7 + 6*9 + 5*5 + 4*0 + 3*4 + 2*5) \bmod 11 = 241 \bmod 11 = 10 \text{ and}$$

$I = 11 - 10 = 1$ yielding ISCN 0-387-95045-1

D- Zero Knowledge of Authentication

In cryptography, a zero-knowledge proof or zero-knowledge protocol is an interactive method for one party to prove to another that a (usually mathematical) statement is true, without revealing anything other than the veracity of the statement.

A zero-knowledge proof must satisfy three properties:

Completeness: if the statement is true, the honest verifier (that is, one following the protocol properly) will be convinced of this fact by an honest prover

Soundness: if the statement is false, no cheating prover can convince the honest verifier that it is true, except with some small probability.

Zero-knowledge: if the statement is true, no cheating verifier learns anything other than this fact. This is formalized by showing that every cheating verifier has some simulator

that, given only the statement to be proven (and no access to the prover), can produce a transcript that "looks like" an interaction between the honest prover and the cheating verifier.

The first two of these are properties of more general interactive proof systems. The third is what makes the proof zero-knowledge.

Research in zero-knowledge proofs has been motivated by authentication systems where one party wants to prove its identity to a second party via some secret information (such as a password) but doesn't want the second party to learn anything about this secret. This is called a "zero-knowledge proof of knowledge". However, a password is typically too small or insufficiently random to be used in many schemes for zero-knowledge proofs of knowledge. A zero-knowledge password proof is a special kind of zero-knowledge proof of knowledge that addresses the limited size of passwords.

Zero-knowledge proofs are not proofs in the mathematical sense of the term because there is some small probability, the soundness error, that a cheating prover will be able to convince the verifier of a false statement. In other words, they are probabilistic rather than deterministic. However, there are techniques to decrease the soundness error to negligibly small values.

In the following protocol the existence of a trusted authority T is assumed. The only purpose of the agency is to publish a modulus n which equals the product of two large primes p and q but to keep the primes themselves secret. For a technical reason to be explained later, the primes are assumed to be congruent with 3mod4. After publishing n the trusted authority may cease to exist. Entity A secret identification d_A consists of k numbers $d_1,...,d_k$ with $1 \le d_j < n$. His public identification e_A consists of k numbers $e_1,...,e_k$ with $1 \le e_j < n$ and each d_j satisfying one of the congruence $d^2_j * e_j \equiv \pm 1 \bmod n$. The verifier entity B knows the public n and e_A.

Entity A wants to convince her that he knows d_A. The following four steps constitute one round of the protocol. The number of rounds decreases the probability of entity A cheating (Fiat-Shamir, 1986).

1. Entity A chooses random number r and computes the number $r^2 \bmod n$ and tells one of them call it x to entity B
2. Entity B chooses a subset s of the set $\{1,...,k\}$ and tells it to entity A

3. Entity *A* tells entity *B* the number $y = r * T_d \bmod n$ where T_d is the product of the numbers d_j such that *j* belongs to *s*

4. Entity *B* verifies the condition $x \equiv \pm y^2 T_e \bmod n$ where T_e is the product of the numbers e_j such that *j* belongs to *s*. If it is not satisfied, entity *B* rejects. Otherwise, an eventual new round is begun.

Example:

- Entity *A* wishes to prove to entity *B* his identity in order to access a resource, obtain a services etc.
- Entity *B* may ask the following: prove that you're entity *A*!

The initial authentication problem is fully solved by the trusted authority T published the

Modulus $n=p*q$

n=47*59= 2773

The trusted authority can distribute the identification material n in a secure fashion, for example by hand, or over encrypted and authenticated lines.

<table>
<tr><th>Entity A</th><th>Entity B</th></tr>
<tr><td></td><td>Entity B gets from the Trusted Authority n=2773 but not its factorization(such as p and q as they must keeping secret with the trusted authority and no one must know about them)</td></tr>
<tr><td>Will generate secret identification d_j which consist of 6 tuples at random:
<table><tr><td>d_1</td><td>d_2</td><td>d_3</td><td>d_4</td><td>d_5</td><td>d_6</td></tr><tr><td>1901</td><td>2114</td><td>1509</td><td>1400</td><td>2001</td><td>0119</td></tr></table>**Entity A will keep $d_1, d_2, \ldots, d_6$ secret.</td><td></td></tr>
</table>

<table>
<tr>
<td>Now entity A will chooses his public identification e_j to consist of 6 tuples:
<table><tr><td>e1</td><td>e2</td><td>e3</td><td>e4</td><td>e5</td><td>e6</td></tr><tr><td>81</td><td>2678</td><td>1207</td><td>1183</td><td>2681</td><td>2595</td></tr></table>* e_j computed here to satisfy the congruence:
$d_j^{\,2}$ x e_j mod n= 1 for j=1,3,4,5 or n-1 for j=2,5,6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Entity B will have now $e_1,e_2,\ldots,e_6$.
<table><tr><td>e1</td><td>e2</td><td>e3</td><td>e4</td><td>e5</td><td>e6</td></tr><tr><td>81</td><td>2678</td><td>1207</td><td>1183</td><td>2681</td><td>2595</td></tr></table></td>
</tr>
<tr>
<td>Now entity A chooses his r=1111 randomly, tell Entity B the number X=(r^2 mod n) Thus X=2437</td>
<td>X=2437</td>
</tr>
<tr>
<td>According to the sequence that we have from entity B,
Entity A will select d_1,d_4,d_5, and d_6</td>
<td>Entity B will chooses for example a sequence
S=1,4,5,6 and inform entity A to compute according to
this sequence.
** The challenge will start from here! **</td>
</tr>
<tr>
<td>Entity A compute T_d
T_d=1901*1400*2001*0119 mod 2773= 96</td>
<td>Entity B will compute T_e and wait!
T_e= 81*1183*2681*2595 mod 2773= 1116</td>
</tr>
<tr>
<td>then entity A will compute Y= T_d*r mod n and tell
Entity B the number Y= 96*1111 mod 2773=1282</td>
<td>Y= 1282</td>
</tr>
<tr>
<td></td>
<td>Entity B will verify now:
X= Y^2* T_e mod n
$(1282)^{\,2}$ * 1116 mod 2773=2437 And this is the value of X
So the system has a probability now 50% that the prover
is Entity A
system will ask her to continue!</td>
</tr>
<tr>
<td>Now Entity A chooses another r=1990 randomly, tells Entity B the number X=r^2 mod n Thus X=256</td>
<td>X=256</td>
</tr>
<tr>
<td>According to this sequence, Entity A will select now
d_2,d_3,and d_5</td>
<td>Entity B will chooses now the sequence S=2, 3, 5 and
inform Entity A to compute according to this sequence.</td>
</tr>
</table>

Entity A will compute T_d T_d=2114*1509*2001 mod 2773= 1228	Entity B will compute T_e and wait! T_e= 2678*1207*2681 mod 2773= 688
then Entity A will compute Y= T_d*r mod and tell Entity B the number Y= 1228*1990 mod 2773=707	Y= 707
	Entity B will verfiy now: X=(Y^2* T_e mod n)+n (707^2 * 688 mod 2773) + 2773=256 the system has another 50% probability that the prover is Entity A Thus the system will verify that the prover is Entity A with a probability 100%. END!

E- MD5 hash

The MD5 (Berson, 1992) hash also known as checksum for a file is a 128-bit value, something like a fingerprint of the file. There is a very small possibility of getting two identical hashes of two different files. This feature can be useful both for comparing the files and their integrity control.

Let us imagine a situation that will help to understand how the MD5 hash works.

Entity A and Entity B have two similar huge files. How do we know that they are different without sending them to each other? We simply have to calculate the MD5 hashes of these files and compare them.

MD5 Hash Properties

The MD5 hash consists of a small amount of binary data, typically no more than 128 bits. All hash values share the following properties:

Hash length

The length of the hash value is determined by the type of the used algorithm, and its length does not depend on the size of the file. The most common hash value lengths are either 128 or 160 bits.

Non-discoverability

Every pair of un-identical files will translate into a completely different hash value, even if the two files differ only by a single bit. Using today's technology, it is not possible to discover a pair of files that translate to the same hash value.

Repeatability

Each time a particular file is hashed using the same algorithm; the exact same hash value will be produced.

Irreversibility

All hashing algorithms are one-way. Given a checksum value, it is infeasible to discover the password. In fact, none of the properties of the original message can be determined given the checksum value alone.

Example:

We begin by supposing that we have a b-bit message as input, and that we wish to find its message digest. Here b is an arbitrary nonnegative integer; b may be zero, it need not be a multiple of eight, and it may be arbitrarily large. We imagine the bits of the message written down as follows:

m_0 m_1 ... m_{b-1}

The following five steps are performed to compute the message digest of the message.

1. Append Padding Bits

The message is "padded" (extended) so that its length (in bits) is congruent to 448, modulo 512. That is, the message is extended so that it is just 64 bits shy of being a multiple of 512 bits long. Padding is always performed, even if the length of the message is already congruent to 448, modulo 512.

Padding is performed as follows: a single "1" bit is appended to the message, and then "0" bits are appended so that the length in bits of the padded message becomes congruent to 448, modulo 512. In all, at least one bit and at most 512 bits are appended.

2. Append Length

A 64-bit representation of b (the length of the message before the padding bits were added) is appended to the result of the previous step. In the unlikely event that b is greater than 2^{64}, then only the low-order 64 bits of b are used. (These bits are appended

as two 32-bit words and appended low-order word first in accordance with the previous conventions.)

At this point the resulting message (after padding with bits and with b) has a length that is an exact multiple of 512 bits. Equivalently, this message has a length that is an exact multiple of 16 (32-bit) words. Let M[0 ... N-1] denote the words of the resulting message, where N is a multiple of 16.

3. Initialize MD Buffer

A four-word buffer (A,B,C,D) is used to compute the message digest. Here each of A, B, C, D is a 32-bit register. These registers are initialized to the following values in hexadecimal, low-order bytes first):

word A: 01 23 45 67

word B: 89 ab cd ef

word C: fe dc ba 98

word D: 76 54 32 10

4. Process Message in 16-Word Blocks

We first define four auxiliary functions that each take as input three 32-bit words and produce as output one 32-bit word.

F(X,Y,Z) = XY v not(X) Z

G(X,Y,Z) = XZ v Y not(Z)

H(X,Y,Z) = X xor Y xor Z

I(X,Y,Z) = Y xor (X v not(Z))

In each bit position F acts as a conditional: if X then Y else Z. The function F could have been defined using + instead of v since XY and not(X)Z will never have 1's in the same bit position.) It is interesting to note that if the bits of X, Y, and Z are independent and unbiased, the each bit of F(X,Y,Z) will be independent and unbiased.

The functions G, H, and I are similar to the function F, in that they act in "bitwise parallel" to produce their output from the bits of X, Y, and Z, in such a manner that if the corresponding bits of X, Y, and Z are independent and unbiased, then each bit of

G(X,Y,Z), H(X,Y,Z), and I(X,Y,Z) will be independent and unbiased. Note that the function H is the bit-wise "xor" or "parity" function of its inputs.

5. Output

The message digest produced as output is A, B, C, D. That is, we begin with the low-order byte of A, and end with the high-order byte of D.

(2) CODES DEFINITIONS:

A- Machine ID

Our program will read hardware serial numbers for the CPU, BIOS, and Hard Disk. Then generate a unique Machine ID function to CPU, BIOS, and Hard Disk.

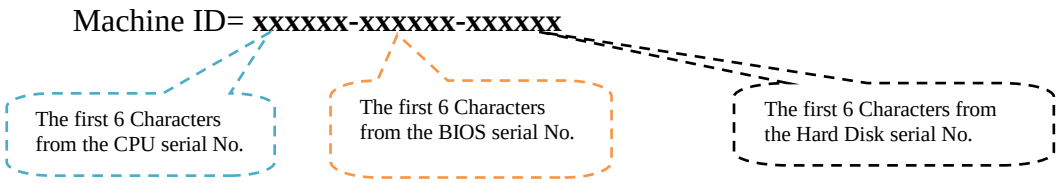

Figure 3.1 Machine ID structure

B- Installation ID

The obtained Machine ID will encrypted using Triple DES Algorithm.

The simplest variant of Triple DES operates as follows: DES(k_3;DES(k_2;DES(k_1;M))), where M is the message block (here will be the Machine ID) to be encrypted and k_1, k_2, and k_3 are DES keys. See figure 3.2.

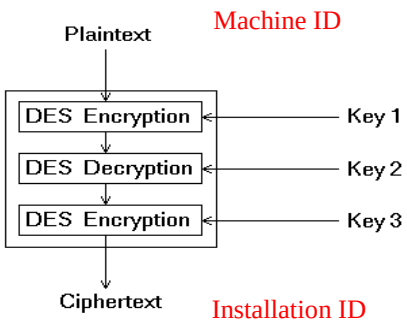

Figure 3.2 Installation ID

C- Copy ID

Each client could have unlimited number of copies for each purchased application (ex. We have 10 copies from Microsoft Office). To identify that, we added a code called "Copy ID". This code is unique among items. So each software package could have its own and only copy ID. The copy ID is mandatory for the activation process as we will see. We can generate such code using many ways; here, it is generated through a special developed algorithm The International Standard Copy Number (ISCN). The ISCN is an identification number assigned to virtually every produced copy. It serves to uniquely identify the copy.

Copy ID conditions: (all Copy IDs stored in Company Database)

1- Copy ID doesn't exist in company Data Base => error message "invalid copy ID"

2- Copy ID in company Data Base, and

   a- No one used it before => Continue with activation process (such as add Machine Information to the Database etc…).

   b- Used by the same user who is trying to activate his machine => Continue with activation.

      If that user tried to install the same copy and run the program from other machine so this mean different Hardware and thus different Machine ID and Different Installation ID. The program will not run and error message will appear" you cannot use the program on another machine"

   c- Used by another person(on another machine) => error message : "this copy is already registered"

D- IP Address

An IP address (Internet Protocol address) is a unique address that certain electronic devices use in order to identify and communicate with each other on a computer network utilizing the Internet Protocol standard (IP)—in simpler terms, a computer address.

Our application has the ability to authenticate along with IP Address and allow or deny access to the company web server based on the public IP of the client.

In current practice, an IP address is not always a unique identifier that always uniquely identifies a particular device, due to technologies such as dynamic assignment and network address translation.

When a computer uses the same IP address every time it connects to the network, it is known as a Static IP address. Static IP addresses are manually assigned to a computer by an administrator. In contrast, in situations when the computer's IP address changes frequently (such as when a user logs on to a network through dialup or through shared residential cable) it is called a Dynamic IP address.

Our application will keep using IP Address as an option and up to the clients who are good candidates for IP authentication such as schools, libraries and other organizations that don't go through a common or dynamic IP Address and thus they will have more secure access to activate and run their owned copy and make it much harder to piracy.

E- Activation Code

This is the required code to run the program.

For each client who has a unique Machine ID there is a unique Activation Code will be in the company Database with the end of transactions between the two parties and will be send back to the client *only and only if* the Authentication processing is Valid.

Also this code can be regenerated inside our program. As we need to regenerate this code inside the computer client in order to make our main comparison with what we have from the window registry (see flowchart).

To generate a unique Activation Code, we need the following steps:

1- Read hardware serial numbers for the CPU, BIOS, and Hard Disk.
2- Generate Machine ID, then,
3- Encrypt step 2 using TripleDES, then,
4- Hash step 4 using MD5 hash.

Note: in step 2 we will use different keys (not the same keys that have been used to generate Installation ID from the machine ID encryption), the reason is, if we used the same keys anyone who can see the Installation ID and know the generation sequence for the Activation Code will hash it using MD5 hash and have the activation code.

Note: hash is not a reversible function (not a symmetric function) which means anyone want to regenerate activation code will not know how this process done.

(3) PROTECTING EXE FILES

1- Protected File Structure

The file that we are talking about is a combination of 3 files:

1- **protect.exe**: a file contains the protection interface (our protection technique)

2- **file.exe**: Exe file is used to install and run our program and routines. We want to protect this file (applying our protection techniques on), it might take any name ends with the extension ".exe".

3- **checksumsfile.txt**: a file contains the checksum values of "file.exe". This file is generated during the encapsulation process.

The resulted file will have the same name"file.exe" and will replace the existed file.exe

2- Protection technique(EXE Encapsulator)

EXE Encapsulator is a program will encapsulates any existed "EXE" file without having an access to its internal source.

The encapsulated file will use our methods to validate the authenticity of the copy and protect it from any non authorized execution. This program is very helpful if we want to use the method with a currently developed program which we do not have its source code.

The Encapsulator combines the given executable with our protection interface which is stored externally on a file named "protect.exe". The protection interface and its required libraries shall be present at the same folder as the required executable. When combining the files, the resulted file will run to show our protection interface which is responsible for the logic behind the authentication process. Figure 3.3 illustrates how the encapsulation works.

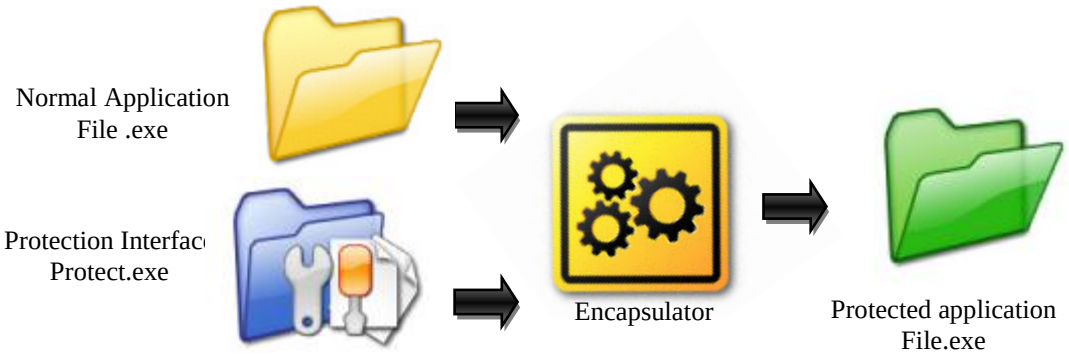

Figure 3.3 The Encapsulation Process

3- Checksum

A checksum is a form of redundancy check, a simple way to protect the integrity of data by detecting accidental modification such as corruption to stored data. It works by adding up the basic components of a message, typically the asserted bits, and storing the resulting value. Anyone can later perform the same operation on the data, compare the result to the authentic checksum and (assuming that the sums match) conclude that the message was most likely not corrupted.

The MD5 hash is commonly used to verify the integrity of files. The 128-bit (16-byte) MD5 hashes are typically represented as a sequence of 32 hexadecimal digits.

It is extremely unlikely that any two non-identical files existing in the real world will have the same MD5 hash.

The following example demonstrates a 43-byte ASCII input and the corresponding MD5 hash:

MD5 ("The quick brown fox jumps over the lazy dog") =
9e107d9d372bb6826bd81d3542a419d6

Even a small change in the message will result in a completely different hash, due to the avalanche effect (When a single bit is changed the hash sum becomes totally different).

For example, changing d to e:

MD5 ("The quick brown fox jumps over the lazy eog") =
ffd93f16876049265fbaef4da268dd0e

The checksum calculation and storage will serve as a mean to detect file tampering.

Back to the flowchart, after the main comparison for Activation code, checksum will be an additional obstacle to the cracker, if cracker tried to change the file and add some code that will bypass the verification of "activation" he would face another verification short after.

(4) HOW TO OBTAIN THE HARDWARE INFROMATION FOR ANY CLIENT'S COMPUTER:

1- The Company's System already obtained the hardware information for the client's computer cause the computer has been sent (physically) to the company. The company will obtain the Hardware information (such as serial numbers for the CPU,

BIOS, and Hard Disk), install the Software on client's computer and send it back to the client ready to use.

Also computer assembly manufactures can bundle our application with other software packages (like windows, Microsoft Office) as pre-install software.

2- If we couldn't send before the client's computer to the company to install the software, the client should install the software manually. After the installation the client will access the internet and thus the Company's System will obtain the hardware information at the first run of the program, bases on that the copy will be locked for a specific user on his specific machine.

If there is no internet connection, the client can update the company with the hardware information by telephone or fax as he will inform them the installation ID and the Copy ID which the client will have through the program execution on his computer.

(5) AUTHENTICATION

Activation process takes place by generate and send an Activation code in case the client has:

- ✓ Validate Copy ID (not already used and an error message will appear if the system dictate that)
- ✓ Validate Installation ID (Once the company obtained Machine ID, it will be checked with the company Data Base to detect wither the client is installing the software package on his machine or on another machine).
- ✓ Validate IP Address (our application will consider this validation if and only if the client activate IP Address authentication).
- ✓ Validate File Checksum value (any tampering in the file will lead to failure in the authentication process)

Any fail in steps above, company will know that the user is using a pirated copy. No activation code will send in this case.

(6) CUSTOMER TRACKING SYSTEM

1- **DATABASE:** Store the information about the clients and their purchased products. This database can be accessed through either the web service or through the web interface.

Here, Data Base created specifically to enable us testing our system. The Data Base and its system could be extended in several ways. However, this is out of our system's scope.

Entity-relationship model: is an abstract conceptual representation of structured data. Entity-relationship modeling is a relational schema database modeling method, used to produce a type of semantic data model of a system, often a relational database, and its requirements in a top-down fashion. See figure 3.4

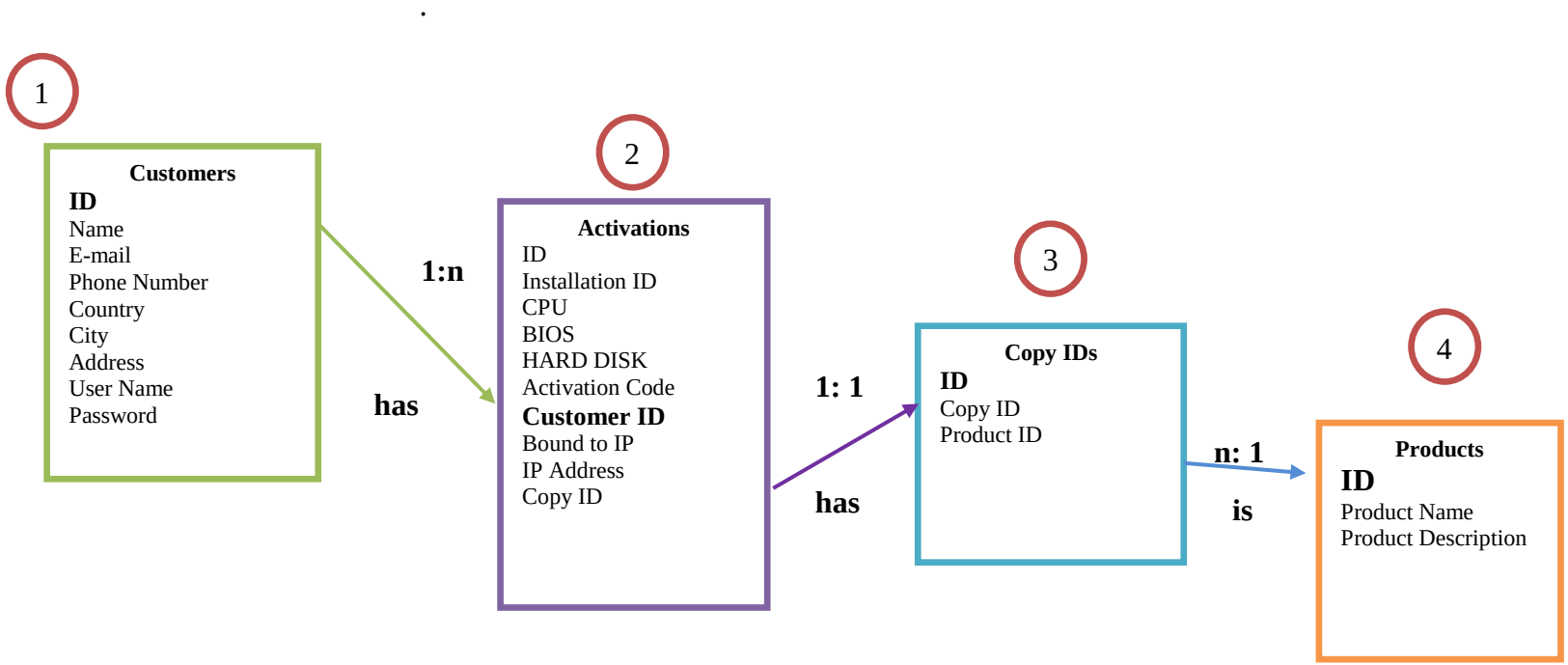

Figure 3.4 Entity Relationship Model

1 **Customers**: This table stores primary information about each customer in the system. Check Table 3.10

Table 3.10 Customers table structure

Field Name	Field Type	Field Data Type	Description
ID	Primary Key	Auto Number	A unique ID for each customer. (Since that it is a primary key, the value can't be duplicated in this table)
Name		Text	Customer name.
E-mail		Text	Customer's e-mail. The validation is held in the interface level (customer registration through the system webpage). So the interface (web page) will not insert any email address if it doesn't satisfy the naming rules for the emails (xxx@xxx.com)
Phone Number		Text	Customer phone number. The validation will check if the number subject to the following international standard form (xxx) xxx-xxxxxxxx . if the form not much, system will held at the interface level.
Country		Text	Customer country. (Rather than using the country name as a linked value to other table, it is used directly to indicate that the focus is not on the Data Base design rather than on the overall system of security)
City		Text	Same as country
Address		Text	The address for each customer.
User Name	Primary key	Text	The login Name for the customer. No duplication allowed at the interface level.
Password		Text	The password for the customer. In our system we don't store the direct password, Instead we store a series of e_A values that represent the *public key* for the customer. Where the direct password is a series of d_A values that represent the *private key* and is only known by the customers themselves. With that scheme. Only our protocol (zero Knowledge Algorithms) can authenticate the customer. And there is no way for any intruder on the database to steal the passwords of customers. This shows the high safety level of using our protocol.

2 **Activations:** This table stores information about each "Activation Process" for each customer. Each customer could have more than one "activation Process" such as more than one program which is activated using our system. Check table 3.11

Table 3.11 Activations table structure

Field Name	Field Type	Field Data Type	Description
ID	Primary Key	Auto Number	A unique ID for each Activation Process. (Since that it is a primary key, the value can't be duplicated in this table)
CPU		Text	store the serial number for CPU
Hard Disk		Text	store the serial number for Hard Disk
BIOS		Text	store the serial number for BIOS
Installation ID		Text	The installation ID is stored directly rather than computed to enhance the database performance. (Although each installation ID can be generated from the 3 serials. This will create a system over load when the user tries to fetch his list of activations)
Activation Code		Text	Stored for the same purpose as the installation ID.
Bond to IP		Boolean	This value tells the system whether to strict the program access to the given IP or not. (it takes one of two values, true or false)
IP		Text	This values will not be used if the user didn't select "bond to ip " option
Customer ID	Foreign Key	Number	This value stores the ID of the customer who has this activation. (named foreign key to indicate that it is unique in other table)
Copy ID	Foreign Key	Number	This value will connect the two tables (user codes, and this table). (named foreign key to indicate that it is unique in other table)

3 **Copy IDs:** This table stores the valid list of "Copy ID"s and links each one with a specific product. Check table 3.12

Table 3.12 Copy IDs table structure

Field Name	Field Type	Field Data Type	Description
ID	Primary Key	Auto Number	A unique ID for each copy ID. (Since that it is a primary key, the value can't be duplicated in this table). This value is used for database indexing purposes.
Copy ID	Primary key	Text	The actual copy ID value. (Since that it is a primary key, the value can't be duplicated in this table)
Product ID	Foreign key	Number	The unique ID of the product which this copy ID uses. Since that each product could use different Copy IDs. (products are listed in other table)

4 **Products:** The system can be used with several products. For that purpose this table was created. However, due to the nature of the system that is security system more effort done on the security issues rather than on the sample database prototype. Check table 3.13

Table 3.13 Products table structure

Field Name	Field Type	Field Data Type	Description
ID	Primary Key	Auto Number	A unique ID for each Product. (Since that it is a primary key, the value can't be duplicated in this table)
Product name		Text	The name of the product
Product Description		Text	Short description for each product

Relations in our Entity Relationship Model:

Table 3.1) shows the relations defined in our entity relationship model.

Table 3.14 Entity Relationship Model's relations

Relation	Type	Description
Customer has an Activation	1:n	Each customer could have one or more (one to many) associated activations.
Activation has a copy ID	1:1	Each activation process uses only one associated copy ID
Copy ID is a product	n:1	Many Copy IDs could be of one product.

1- WEBSITE:

A. Register New User page: create a typical set of information record for each client.

B. Login page: access to the Database using our Protocol Authentication the Zero-Knowledge Authentication.

C. Products page: list all the client's activated products.

D. New Copy activation page: a sub page from the Products page.

2- WEBSERVICE: the essential part of Web services is the *Interact* relationship between a Service provider and Service requestor.

So when I need to get some programming task done, I can make use of a web service by calling it over the Internet. By passing parameter data with the request, I can expect to receive a response containing the result generated by the web service.

Web Sites are just the user interface of your application, and Web Service are intended to expose some functionality to the outside or to some other layer as a service.

We are using the web service in the path where the protection interface needs to interact with the server directly using our secure protocol (Zero-Knowledge Authentication) in order to activate the copy. In which the client is directly connected to the net and has a reg.ini file in his directory. And this is the case which the client had already sent his pc to the company.

(7) HOW THE APPLICATION WORKS

Figure 3.5 Shows how the application works.

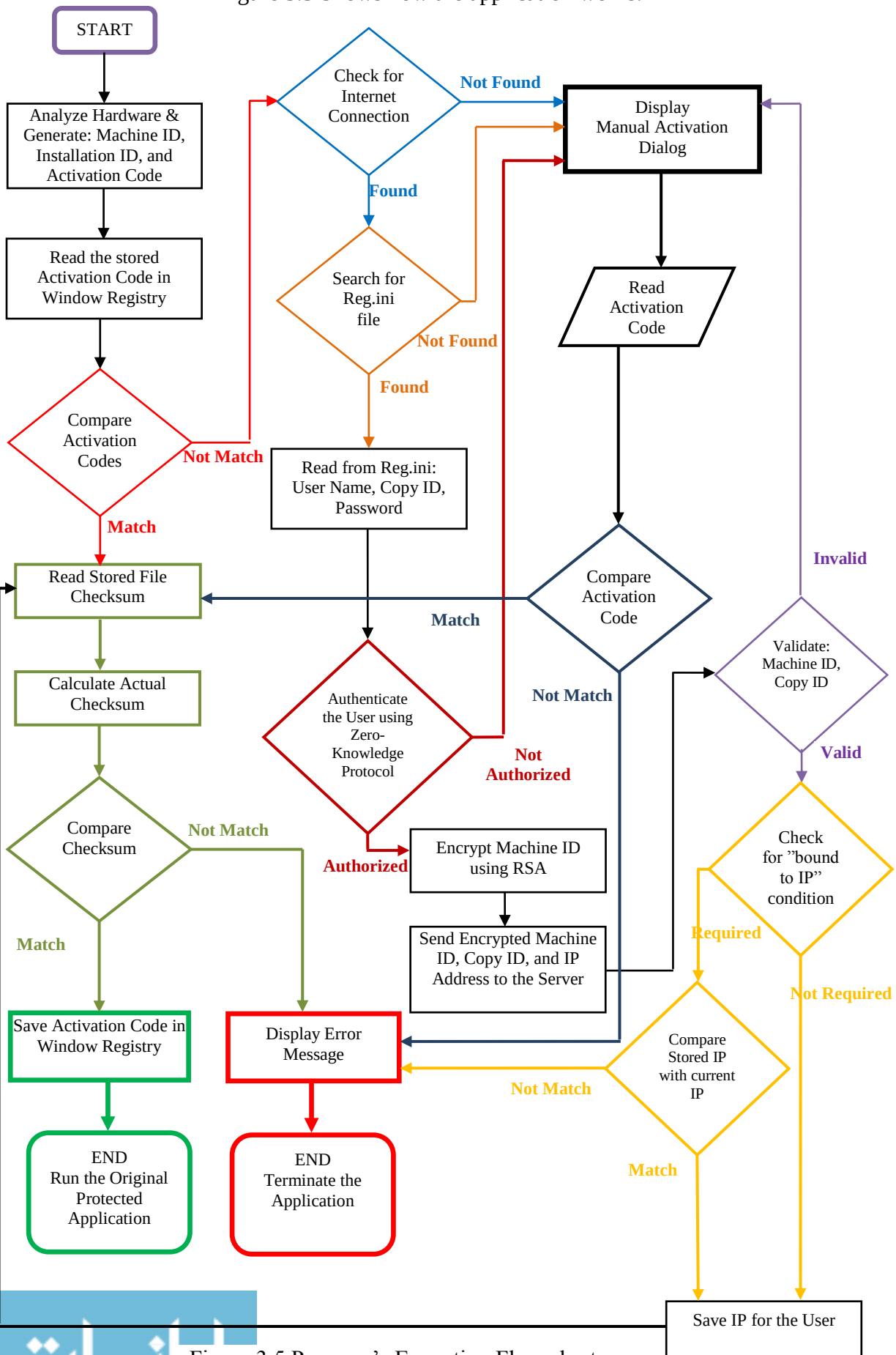

Figure 3.5 Program's Execution Flow chart

Chapter Four

Analyzing and Testing the Proposed Scheme

Due to the commercial nature of the problem, we are unable to obtain analysis to the implemented systems. Thus we failed to conduct comparisons between them. Additionally, there is a lack of analysis in related system we discuss in related work section since there is no source code. Despite this difficulty, we are focused on analyzing and testing the proposed scheme as follows:

4.1 System Analysis

For each component of the suggested system we will discuss the following parts:

1. **Time Complexity in term of big O Notation**

It is important to know the efficiency of an algorithm used in the proposed system. The system overall complexity is the sum of the sub systems (algorithms) complexity analysis.

2. **Time Execution**

In order to estimate the required time for the suggested application, it should be add some time break points in the code. Figure 4.1 shows the screenshot and the elapsed time for the operations as taken from the debug screen.

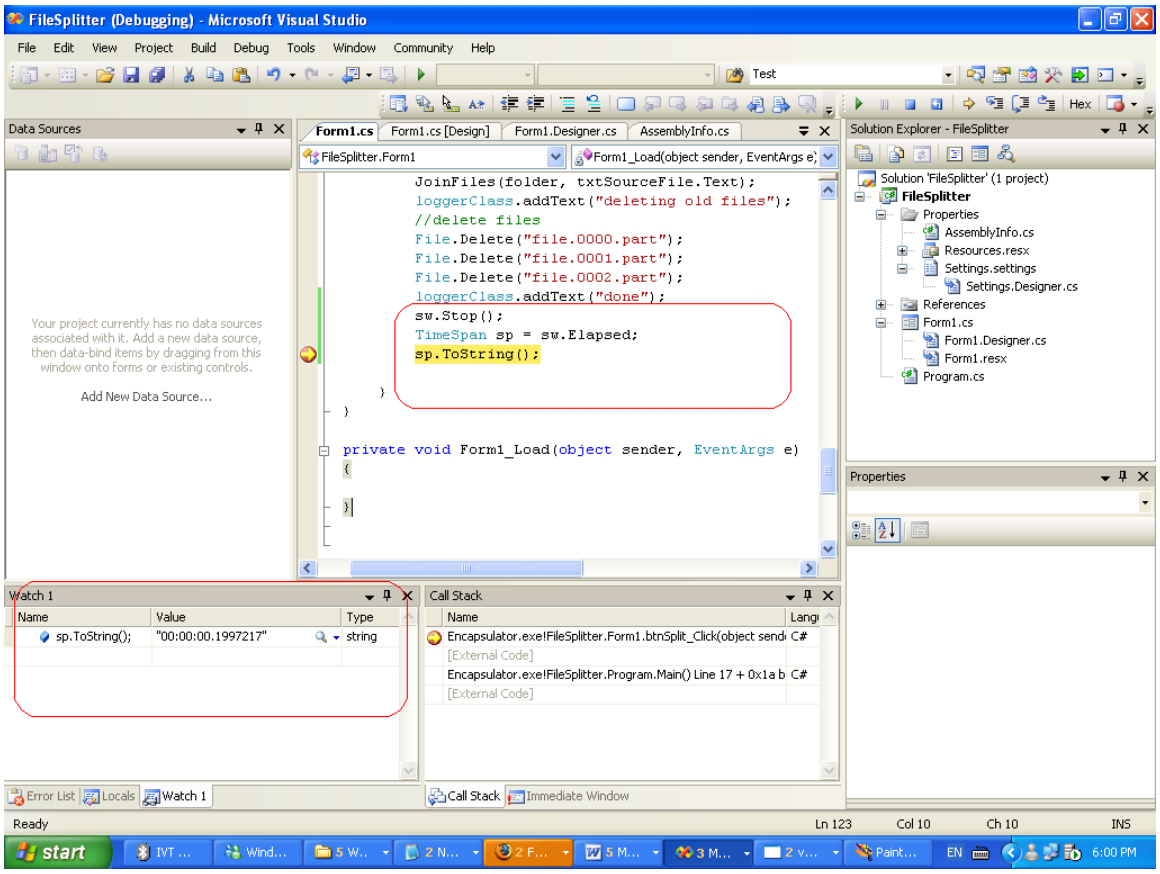

Figure 4.1 Execution Time Snapshot

3. Number of Iterations

Some of the used algorithms require more than one iteration to catch the target. Thus, we are counted the actual number of iterations in order to get the result. The computer used is an HP laptop that is powered by Intel Centrino CPU running at 1.8 giga hirtz. It has 512 mega bytes of memory running on Windows XP. It also has an internet connection. The used compiler is Microsoft Visual Studio 2005. The languages used are C#, and VB .Net.

The sub systems that will be tested are as follows:

1. Encapsulator
2. International Standard Copy Number
3. Protection Interface

This will be divided into two parts:

In the first part the following algorithms are discussed:

a- Zero Knowledge Proof of identity

b- Enhanced RSA

c- Md5 hash and TDES

In the second part the Overall performance for the protection interface is discussed.

1. Encapsulator

I. Complexity

It consists of two major set of functions which are as follows:

A. Files combining: This function combines three files into one file.

- Algorithm:
  1. Read the contents of the first file a
  2. Write the contents into the output file
  3. Read the contents of the second file
  4. Write the contents into the output file
  5. Read the contents of the third file
  6. Write the contents into the output file
- Pseudo Code

Illustrated in Figure 4.2

```
        //joining files in directory into one file
                DirectoryInfo diSource = new DirectoryInfo(FolderInputPath);
        FileStream fsSource = new FileStream(FileOutputPath, FileMode.Append);
        for(int i =0 ;i<=2;i++)
           {
              Byte[]  bytePart  =  System.IO.File.ReadAllBytes("file.000"+i.ToString
()+".part");
           fsSource.Write(bytePart, 0, bytePart.Length);
           }
                fsSource.Close();
     }
```

Figure 4.2 Pseudo Code for files combining

- Input size of the algorithm: The input size for this algorithm is the size in bytes of the three files. This can not be determined because the file that we want to protect has an unknown size. The total size of the input will be N.
- Time Efficiency analysis:

Running this algorithm will execute six steps regardless to the input size N. It will execute the reading and writing of the files for three times.

$$T_G(N) = \sum_{i=0}^{2} 2 = 2^3 - 1$$
$$= 7 \; Bit \; Operations$$

- Big O Notation:

This algorithm always requires seven operations. It is in the order of O(7)

B. MD5 hash calculations

- This is a standard algorithm. It has already been discussed and studied.
- Input size of the algorithm: the input size is the number of bytes to be hashed=n.
- Big O Notation:

  It is in the order of O(n).

Thus the over all program operates in the order of O(n)+O(n) = O(n). Table 4.1 shows the calculations.

Table 4.1 Complexities used in the Encapsualtor

Part	Complexity
A	O(n)
B	O(n)
Total	O(n)+O(n) = O(n)

II. Execution Time

The complete encapsulation process takes 0.2 second.

2. International Standard Copy Number

I. Complexity

We will calculate the complexity for the ISCN check digit here.

- Algorithm:

  1. Obtain the number of digits *N.*

  2. Read the digits $d_1, d_2, ..., d_N$

  3. Loop on *i* where *i* = 1 to *(N-1)*

  a. Compute $T = T + d_i * i$

  4. Compute *T*=*T* mod 11

  5. Compute the digit= 11-*T*

- Pseudo Code:

  Figure 4.3 shows the code of the ISCN

```
long T=0;
                    for (int i=(N-1);i>=1;i--)
                    {
                        T+=Int16.Parse( allDigits[N-i].ToString())*(i+1);


                    }
                    T=T%11;
                    long cdigit=11-T;
```

Figure 4.3 Pseudo Code for ISCN

- Input size of the algorithm: The input size of this algorithm is the number of digits N.
- Example:

Digits = 038795045

$$T = (10*0+9*3+8*8+7*7+6*9+5*5+4*0+3*4+2*5) \bmod 11 = 241 \bmod 11 = 10 \quad I = 11-10 = 1$$

The check digit = 1

- Time efficiency analysis:

  Running time of this algorithm is spent on the loop which is dependant on the size of *N*. This will lead to the following time efficiency function:

$$T_G(N) = (\sum_{i=0}^{N-1} 2) + 2 = (N-1) * 2 + 2$$
$$= 2N - 2 + 2$$
$$= 2N \; Bit \; Operations$$

  In this algorithm the actual *N* can not be more than 10. This leads that the actual number of bit operations can not exceed 20.

- Big O Notation:

  The check digit calculation has an order of O(*N*). Table 4.2 shows the calculations.

Table 4.2 Complexities used in ISCN

Part	Complexity
Total	O(N)

II. Execution Time

To calculate one ISCN, the author tries to stop the timer and see how time it took repeatedly the answer is zero and this mean that the time is smaller than the single increment ”tick” of the system.

III. Number of iterations

It takes a constant number of iterations which is fixed at 20 iterations.

3. Protection Interface

We can divide the test and analysis for the protection interface into two parts.

Part I: The protection interface uses the following algorithms

A. Zero Knowledge Protocol Authentication

B. Enhanced RSA

C. Md5 hash and TDES

Part II: the overall performance of the protection interface

We will analyze the algorithms mentioned above one by one, and then we will analyze the whole protection interface.

Part I: The protection interface uses the following algorithms

A. Zero Knowledge Authentication Protocol

I. Complexity

As we mentioned before, the protocol divided into smaller functions. The complexity of each one is calculated below;

1) The conversion of the password (characters array) into an integer array *B*:

a. Algorithm:

1. Read input array of the size *N*
2. Let *i* = 1 to the size of the array *N*
    1. Compute the ASCII code for the array element
    2. Store the code in the output array *B*
3. Return *B*

b. Example:

Let *N*=2 , *A*=[1,2]

Let *i* = 1 to 2

B[1]=97

B[2]=98

Return *B*

c. Pseudo Code:

Figure 4.4 shows the code.

```
int[] res = new int[15];
        int j = 0;
        for (int i = 0;i<15;i++)
        {

           int b = ascii(input[i]);
           res[i]=b;

        }
        return res;
```

Figure 4.4 Pseudo Code for password conversion algorithm

d. Input size of the algorithm: The input size of this algorithm is the number of the elements in the array *N*.

e. Time efficiency analysis:

Running time of this algorithm is spent on the loop which is based on the size of *N*. This will lead to the following time efficiency function:

$$T_G(N) = (\sum_{i=0}^{N} 2) = (N) * 2$$
$$= 2N$$
$$= 2N \textit{ Bit Operations}$$

For example

Let *N*=4

$T_G(N)$ =2*4= 8 Bit Operations

f. Order of growth:

Order of growth is the rate of increase in operations for an algorithm to solve a problem as the size of the problem increases. The conversion to ASCII has an order of growth of $T_G(N)$. Where N can not be more than 15

Let *N*=15

$T_G(N)$ =2*15 = 30 Bit Operations

In this algorithm the actual N can not be more than 15. This leads that the actual number of bit operations can not exceed 30.

g. Big O Notation:

The conversion to ASCII has an order of O(1)

2) The calculation of the inverse for each element in *B* using Baghdad Method:

a. Algorithm:

1. Read the input array *B* of the size *N*
2. Set *x*=1 , *d*=1
3. for each element *b* in *B*, do the following
    1. $b = b^2$
    2. Repeat
        a. Set $x = x + n$
        b. Set $d = \frac{x}{b}$
    3. Until d is integer
    4. set the inverse value to d

b. Pseudo Code:

Figure 4.5 shows the code

```
For each b in B[i] do {
        {
           int sqr = b*b ;


           double x = 1;
           double D = 1;
            do
           {
              x = x + Theta;
              D = x / sqr;
           } while (x % sqr != 0);
         e[i] = (int)D;
        }
```

Figure 4.5 Pseudo Code for Baghdad Method

c. Example:

Examples for Baghdad method are provided in the experimental tests section.

d. Input size of the algorithm: This algorithm takes N number to calculate their inverse using Baghdad. Baghdad method's input size is the value of the number *e*.

e. Time efficiency analysis:

Running time of this algorithm is spent on the loop which is dependant on the size of *N*. Inside the loop Baghdad method depends on the input size e This will lead to the following time efficiency function:

$$T_G(N) = \sum_{i=0}^{N} \sum_{j=1}^{e} \frac{1}{j} = (N) * ((\ln(e)) + c) \approx n * e$$

$$= N \ln(e) + 0.577 \; Bit\ Operations$$

f. Order of growth:

It is related to the number of elements and the maximum possible number to inverse.

g. Big O Notation:

It is in order of $O(n*e)$.

3) The rest of the operations can be summarized in this algorithm:

a. Algorithm:

1. Compute *r*
2. Compute *x*
3. Compute T_e
4. Compute T_d
5. Compute *y*
6. check verification condition

b. Pseudo Code

Figure 4.6 shows the code

```
r = randomizer.Next(100)+ 1;
        x = (r*r)%n;
        Td=1;
        for(int i=0;i<ssize;i++)
          Td=Td*d[s[i]];
Te=1;
            for(int i=0;i<ssize;i++)
                Te=Te*e[s[i]];
y=(r*Td)%n;
```

```
if(x%n==((y*y*Te)%n))||x%n+((y*y*Te)%n)==n)
return true;
```

Figure 4.6 Pseudo Code for the rest of the operations in the authentication protocol

c. Example:

A complete example for the zero knowledge is mentioned in the experimental tests section.

d. Input size of the algorithm: This algorithm based on the values of *s*(array of integers), *M*(the length of the array),*n* (modulus). *M* is less than or equal to the password length

e. Time efficiency analysis:

$$T_G(N) = 1+1+\sum_{i=1}^{M} 2 + \sum_{j=1}^{M} 2 + 2$$
$$= 4 + 2 * M + 2 * M$$
$$= 4 * M + 4 \, Bit \, Operations$$

For example:

M=3

$T_G(N)$=4*3+4=16 Bit Operations

f. Big O notation:

It is in the order of O(4*M*+4)

The overall operations in the authentication protocol will be O(2*N*) + O(*N**ln (*e*)) + O(4*N*+4)

It notices that the complexity for the algorithm is in the same order as O(log *N*). Table 4.3 shows the calculations.

Table 4.3 Complexities used in the zero knowledge protocol

Part	**Complexity**
1	O(2*N*)
2	O(*N**ln *e*)
3	O(4*M*+4)
Total	O(2*N*) + O(*N**ln *e*) + O(4*N*+4)

III. Execution Time

1. Authentication for a large password using the proposed protocol: 0.253 sec.
2. Authentication for a small password using the proposed protocol.: 0.246 sec.

IV. Number of Iterations

Since the major operation in the authentication is the inverse calculation using Baghdad method, the author will compute the actual number of iterations used by Baghdad method to produce the equivalent *d* for a selected set of *e*'s taking from a real example.

P=47

Q=59

N=*P***Q*=2773

$(d^2 * e) \bmod 2773 = \pm 1$

Table 4.4 shows the calculations

Table 4.4 Number of iterations to calculate the inverse using Baghdad method

D	***e***	**Number of real iterations**
4	520	3
7	1245	22
11	275	12

B. Enhanced RSA

I. Complexity

The major obstacle in this algorithm comes form the massive multiplications. But since that we used Square and Multiply Algorithm. The modular exponentiation is in order of O(log *N*)

- Algorithm

1- Read p , q , n

2- Compute $g = (p^2 - 1) * (p^2 - p) * (q^2 - 1) * (q^2 - q)$

3- Compute *e* (public key)

4- Compute *d*(private key) using Baghdad method for multiplicative inverse

5- Read *m* (message)

6- Compute $c = m^e \bmod n$ using modular exponentiation

7- Compute $m = c^d \bmod n$ using modular exponentiation

8- Print *c*

- Pseudo Code

Figure 4.7 shows the code

```
Read p,q
N=p*q
g = (p * p - 1) * (p * p - p) * (q * q - 1) * (q * q - q);
get e
d=baghdad(e)
read m
c= modpow(m, e, n);
m= modpw(c,d,n);
modpow function
long modpow(long b, long e, long m)
    {
        long result = 1;
            while (e > 0)
        {
            if ((e & 1) == 1) result = (result * b) % m; // multiply in this bits' contribution while using modulus to keep result small
            e >>= 1;
            b = (b * b) % m;
                }
        return result;
    }
```

Figure 4.7 Pseudo Code for the enhanced RSA

- Example:

  p=47

  q=43

  n=2021

  g=15932153115648

  let e=17

  d=14994967638257 using Baghdad method

  Read m=5

  Compute c= 5^{17} mod 2021=931

  Compute m= $931^{14994967638257}$ mod 2021

- Input size of the algorithm:

  This algorithm depends on the size of n(modulus)=p*q where p,q are primes. It mainly depends on the value of e(public key). Through that value It will generate d(private key) ,and g(see above).

- Time efficiency analysis

  $$T_G(N) = 1+1+4+\sum_{j=1}^{d}\frac{1}{j}+2\sum_{i=1}^{e}\frac{1}{i}$$

  $$= \frac{3}{2}\lfloor \lg e \rfloor$$

  $$or \frac{3}{2}\lfloor \lg d \rfloor$$

  $$= 7.68 + \ln(d) + 2\ln(e)\ Bit\ Operations$$

  Let e=17

  $T_G(N)$=7.68+30.338+2*2.833

  = 40.851 Bit Operations

- Order of growth:

  Order of growth is the rate of increase in operations for an algorithm to solve a problem as the size of the problem increases. RSA encryption/decryption has an order of growth of $T_G(N)$.

Let e=17

$T_G(N)$ =7.68+30.338+2*2.833

=40.851 Bit Operations

- Big O Notation:

It is in order of O(ln(d)+2ln(e)+7.68). Table 4.5 shows the calculations

Table 4.5 Complexities used in the enhanced RSA

Part	Complexity
Total	O(ln(d)+2ln(e)+7.68)

II. Execution time

Encryption took 0.0001843 sec.

Decryption took 0.0047025 sec

III. Number of iterations

This algorithm only depends on the large amount of mathematical operations performed to produce the powers.

p=43

q=47

n= p*q = 2021

g=15932153115648

$c = m^e \bmod n$

$m = c^d \bmod n$

e=17

d=14994967638257

$(e * d) \bmod g = 1$

Table 4.6 shows the calculations.

Table 4.6 Number of iterations to encrypt/decrypt using enhanced RSA

M	***c***	**Iterations for *m***	**Iterations for *c***
97	1976	5	44
98	1578	5	44
1578	1653	5	44

C. Md5 hash and TDES

I. Complexity

These are standard algorithms. They have an order of O(*N*).

II. Execution time

For the Md5 hash it takes 0.0013987 sec.

For the TDES it takes 0.0729318 sec.

Part II: Overall performance for the protection interface

I. Complexity

We can examine the overall performance complexity of the protection interface through two paths which the protection interface works. The algorithms are studied above. Here is a summarization only.

Path 1:

In this path, the program will go through the following operations, but we will consider that;

- There is no internet connection.
- No authentication process (which means there is no actions for the Zero Knowledge protocol).

a- Generate installation ID, and activation code using TDES and MD5 hash.

Installation ID generation has an order of O(*N*) + O(*N*) , where *N* is the number of bytes obtained from the reading values.

b- The activation code comparison is a single operation with a complexity of O(1).

c- Read the stored checksum then calculate File Checksum and do a checksum comparison which has a total order of O(*N*). Where *N* is the number of bytes in the file.

As noticed that all of the used algorithms are in the order of O(*N*) where *N* is the number of bytes in the file. There are also some additional operations of the order O(1), since that O(1)+O(*N*)=O(*N*) Therefore This path is in the order of O(*N*). Table 4.7 shows the calculations.

Table 4.7 Complexities used in path-1 of the protection interface

Part	Complexity
A	O(*N*)
B	O(1)
C	O(*N*)
Total	O(1)+O(*N*)=O(*N*)

Path 2:

In this path, the program will go through the following major operations, but here we will consider the following;

- There is an internet connection.
- The file reg.ini in its place.
- A correct value for the user name and password.
- A successful authentication process.

a- Generate installation ID, and activation code using TDES and MD5 hash. Installation ID generation has an order of O(*N*) + O(*N*) . where *N* is the number of bytes obtained from the read values generate installation ID, and activation code using TDES and MD5 = has an order of O(*N*) + O (*N*) . where *N* is the number of bytes obtained from the read values.

b- The activation code comparison is a single operation with a complicity of O(1).

c- Read the stored checksum then calculate File Checksum and do a checksum comparison which has a total order of O(*N*). Where *N* is the number of bytes in the file.

d- Authenticate the user using the suggested protocol which has the order of O(ln *N*).

e- Exchange data using the enhanced RSA implementation which in order of O (ln *N*) where *N* is the encrypted message value.

The total program is in the order of: O(ln *N*) + O(ln *N*) + O(*N*)+O(1) . where that maximum number of operations is done in the checksum comparisons and calculations. Therefore the program is in the order of O(*N*). Table 4.8 shows the calculations.

Table 4.8 Complexities used in path-2 of the protection interface

Part	Complexity
A	O(*N*)
B	O(1)
C	O(*N*)
D	O(ln *N*)
E	O(ln *N*)
Total	O(ln *N*) + O(ln *N*) + O(*N*)+O(1) =O(*N*)

III. Program Execution Time

A complete autonomous execution for the protection interface: 1.00 sec.

4.2 Experimental Test

In this part we will test the system practically in order to prove the work of the functionalities.

Methodology:

Since the suggested system is based on a combination of new standards. We need to implement a sample construction for the new standards. We developed and implemented a new authentication protocol based on zero knowledge. We also developed an encryption scheme based on an enhanced RSA scheme. The proposed standards ISCN is also implemented and we will also implement some tools to generate and verify it.

Tests:

We will test three sub components.

1. Encapsualtor

Input Conditions:

The required files are in the same directory (encapsulator.exe, protect.exe, file.exe).

Expected Result

The file (file.exe) will run and show “protection interface dialog” upon the successful encapsulation.

Experimental Results:

The file (file.exe) executed and showed “protection interface dialog”.

2. Customer Tracking System

Input Conditions

- Connected to the internet.
- The user registered to the system and his password was not stored (only his public key).

Expected result

The system will authorize the user without knowing his password by using the implementation of zero knowledge protocol.

User Name	ahmadF
Password	ahmad123

After entering the username and password in their fields in the web page, the following results were obtained. (details of one zero knowledge round).

***N*=2773**

User private key: 97, 104, 109, 97, 100, 49, 50, 51

User public key: 491, 854, 362, 491, 1064, 82, 1290, 661

Round 1:

r=40

X= 1600

Y=520

S=2,7

T_e= 239282

T_d=5559

X mod n = 1600 mod 2773 = 1600

Y^2 T_e mod n =270400*239282mod 2773= 64701852800 mod 2773 = 1173 = (-1600)

X mod n = Y^2 T_e mod n is satisfied and the user were authenticated.

Round 2:

r=27

X=729

Y=465

S=7,0

T_e=324551

T_d=4947

X mod n = 729 mod 2773 = 729

Y^2 T_e mod n =216225 *324551 mod 2773= 70176039975 mod 2773 = 729

X mod n = Y^2 T_e mod n is satisfied and the user were authenticated.

Round 3:

r= 24

X= 576

Y= 1134

S=4,5

T_e= 87248

T_d= 4900

X mod n = 576 mod 2773 =576

Y^2 T_e mod n =1285956*87248 mod 2773= 112197089088 mod 2773 = 576

X mod n = Y^2 T_e mod n is satisfied and the user were authenticated.

3. Protection Interface

In each of the following test, a set of conditions will be present and the results are based on the combination of the available conditions.

Part A: Manual mode/web tests.

Case 1:

System Conditions: Table 4.9 shows the conditions

Table 4.9 System Conditions for case 1

Hard Disk Serial Number	3753566138
BIOS Serial Number	CNF5141SNX
CPU Serial Number	AFE9FBFF000006D8
IP condition	False
IP	-
Stored IP	-
Current IP	-
Copy ID	0-387-95045-1
Copy ID is valid and in data base	Yes
Copy ID is used for different user	No
Copy ID is used for the same user	No
Installation ID	ASQl46NXVpU=-ZZN+JbqHj38=-cT80sfpbkes=
Required activation Code	CNa/Y82FqI27+lz1ZC3b5w==
Provided activation code	A
Reg.ini file available	No
Tampered on purpose	No
Internet connection	No

System Response

Error Message: Invalid Activation Code.

Case 2:

System Conditions: Table 4.10 shows the conditions

Table 4.10 System Conditions for case 2

Hard Disk Serial Number	3753566138
BIOS Serial Number	CNF5141SNX
CPU Serial Number	AFE9FBFF000006D8
IP condition	False
IP	-
Stored IP	-
Current IP	-
Copy ID	0-387-95045-1
Copy ID is valid and in data base	Yes
Copy ID is used for different user	No
Copy ID is used for the same user	No
Installation ID	ASQl46NXVpU=-ZZN+JbqHj38=-cT80sfpbkes=
Required activation Code	CNa/Y82FqI27+lz1ZC3b5w==
Provided activation code	CNa/Y82FqI27+lz1ZC3b5w==
Reg.ini file available	No
Tampered on purpose	No
Internet connection	No

System Response

The original protected file executed successfully.

Case 3:

System Conditions: Table 4.11 shows the conditions

Table 4.11 System Conditions for case 3

Hard Disk Serial Number	3753566138
BIOS Serial Number	CNF5141SNX
CPU Serial Number	AFE9FBFF000006D8
IP condition	False
IP	-
Stored IP	-
Current IP	-
Copy ID	0-387-95045-1
Copy ID is valid and in data base	Yes
Copy ID is used for different user	No
Copy ID is used for the same user	No
Installation ID	ASQl46NXVpU=-ZZN+JbqHj38=-cT80sfpbkes=
Required activation Code	CNa/Y82FqI27+lz1ZC3b5w==
Provided activation code	CNa/Y82FqI27+lz1ZC3b5w==
Reg.ini file available	No
Tampered on purpose	Yes
Internet connection	No

System Response

System error message were displayed: your file has been tampered.

Case 4:

System Conditions: Table 4.12 shows the conditions

Table 4.12 System Conditions for case 4

Hard Disk Serial Number	3753566138
BIOS Serial Number	CNF5141SNX
CPU Serial Number	AFE9FBFF000006D8
IP condition	False
IP	-
Stored IP	-
Current IP	-
Copy ID	0-487-95045-1
Copy ID is valid and in data base	No
Copy ID is used for different user	No
Copy ID is used for the same user	No
Installation ID	ASQl46NXVpU=-ZZN+JbqHj38=-cT80sfpbkes=
Required activation Code	CNa/Y82FqI27+lz1ZC3b5w==
Provided activation code	CNa/Y82FqI27+lz1ZC3b5w==
Reg.ini file available	No
Tampered on purpose	No
Internet connection	No

System Response

Web error: Invalid Copy ID

Case 5:

System Conditions: Table 4.13 shows the conditions

Table 4.13 System Conditions for case 5

Hard Disk Serial Number	3753566138
BIOS Serial Number	CNF5141SNX
CPU Serial Number	AFE9FBFF000006D8
IP condition	False
IP	-
Stored IP	-
Current IP	-
Copy ID	0-387-95045-1
Copy ID is valid and in data base	Yes
Copy ID is used for different user	Yes
Copy ID is used for the same user	No
Installation ID	ASQl46NXVpU=-ZZN+JbqHj38=-cT80sfpbkes=
Required activation Code	CNa/Y82FqI27+lz1ZC3b5w==
Provided activation code	CNa/Y82FqI27+lz1ZC3b5w==
Reg.ini file available	No
Tampered on purpose	No
Internet connection	No

System Response

Web error: This copy is registered for different user.

Case 6:

System Conditions: Table 4.14 shows the conditions.

Table 4.14 System Conditions for case 6

Hard Disk Serial Number	3753566138
BIOS Serial Number	CNF5141SNX
CPU Serial Number	AFE9FBFF000006D8
IP condition	False
IP	-
Stored IP	-
Current IP	-
Copy ID	0-387-95045-1
Copy ID is valid and in data base	Yes
Copy ID is used for different user	No
Copy ID is used for the same user	Yes
Installation ID	ASQl46NXVpU=-ZZN+JbqHj38=-cT80sfpbkes=
Required activation Code	CNa/Y82FqI27+lz1ZC3b5w==
Provided activation code	CNa/Y82FqI27+lz1ZC3b5w==
Reg.ini file available	No
Tampered on purpose	No
Internet connection	No

System Response

The application executes.

Part B: Automatic Mode /Web Service Test

Case 1:

System Conditions: Table 4.15 shows the conditions.

Table 4.15 System Conditions for case 1 in the automatic mode

Hard Disk Serial Number	3753566138
BIOS Serial Number	CNF5141SNX
CPU Serial Number	AFE9FBFF000006D8
IP condition	True
Stored IP	192.168.1.3
Current IP	192.168.1.3
Reg.ini file available	No
Tampered on purpose	No
Internet connection	Yes

Expected result

No User information files were stored. The system shall fail with the automatic operation and proceed to the manual mode.

System Response

The manual activation dialog were displayed.

Case 2:

System Conditions: Table 4.16 shows the conditions.

Table 4.16 System Conditions for case 2 in the automatic mode

Hard Disk Serial Number	3753566138
BIOS Serial Number	CNF5141SNX
CPU Serial Number	AFE9FBFF000006D8
IP condition	True
Stored IP	192.168.1.3
Current IP	192.168.1.3
Reg.ini file available	Yes with invalid user information
Tampered on purpose	No
Internet connection	Yes

Expected Result

The system will fail to authenticate the user and will proceed to the manual mode.

System Response

The manual activation dialog was displayed.

Case 3:

System Conditions: Table 4.17 shows the conditions.

Table 4.17 System Conditions for case 3 in the automatic mode

Hard Disk Serial Number	3753566138
BIOS Serial Number	CNF5141SNX
CPU Serial Number	AFE9FBFF000006D8
IP condition	True
Stored IP	192.168.1.3
Current IP	192.168.1.3
Reg.ini file available	Yes
Tampered on purpose	No
Internet connection	Yes

System Response

The protected program worked normally.

Case 4:

System Conditions: Table 4.18 shows the conditions.

Table 4.18 System Conditions for case 4 in the automatic mode

Hard Disk Serial Number	3753566138
BIOS Serial Number	CNF5141SNX
CPU Serial Number	AFE9FBFF000006D8
IP condition	True
Stored IP	192.168.1.3
Current IP	204.168.1.4
Reg.ini file available	Yes
Tampered on purpose	No
Internet connection	Yes

System Response

Error message were displayed: you are using a different IP.

Case 5:

System Conditions: Table 4.19 shows the conditions.

Table 4.19 System Conditions for case 5 in the automatic mode

Hard Disk Serial Number	3753566138
BIOS Serial Number	CNF5141SNX
CPU Serial Number	AFE9FBFF000006D8
IP condition	True
Stored IP	192.168.1.3
Current IP	192.168.1.3
Hard Disk Serial Number	3753566138
BIOS Serial Number	CCC5141SNX
CPU Serial Number	AFE9FBDDD06D8
Reg.ini file available	Yes
Tampered on purpose	No
Internet connection	Yes

Expected Result

The user did change his original hardware. Thus the system will not authorize the software execution.

System Response

Manual activation dialog were displayed.

Chapter Five

Conclusion and Future Work

5.1 Conclusion:

In this thesis along with its developed applications, the author managed to create a piracy prevention technique that will help the developer, end user, and high security enterprise user. The developer can now use the proposed system to protect his future products with a very easy way. The end user can run his protected program with no obstacles. The enterprise user can experience new levels of security. The suggested system came as a structure for future systems where other people can build and extend its features in the same context. We combined famous techniques in addition to implementing other suggested techniques especially the authentication protocol which hopefully to be used in other systems as well.

5.2 Future work:

In this section, the author describes some of the future enhancements that the proposed system could have.

- Implementation into hardware

  In the case of high risk systems, the author could implement the suggested authentication protocol into a chip rather than software which will increase the speed and will make the system more convenient for high security places. Even though this will lead the system is not suitable for all types of users.

- Browser support

  The current implementation of the system is done in the user's level due to the unavailability of the source code of the browsers. But it is possible in the future to publish the used techniques as standards which will lead the browser to implement them. By this, other developers could use the suggested protocol for more than one purpose. This also applies to the use of the enhanced RSA scheme.

- Used as a customizable framework

  Any part of the system could be enhanced in one way or another in the future. Since our system was designed to address all parts of the equation. Each part (developer, end user, enterprise user) can have additional features that suit him better. And that

would be implemented either using another research or inside a development company.

- Licensing Management

  The frame work could benefit from other competing ideas such as advanced license management where the copy id could allow more than one license. Or more than one IP bound to it. Another idea is the expiration date for each license. We can manage to bind some products to an expiration date, or license renewal period.

- Virus attack rather than just an error message.

  In the current implementation we used to display an error message in the case of failures. But we can prevent piracy with another idea in the future. A virus-like program would be executed in the system rather than just informing the user that he is an unauthorized user. It might have some sort of attack on the system. either a friendly attack such as displaying an annoying pop up each couple of seconds or more seriously to destroying some important files in the system. Such idea could be used as a defense mechanism for high security situations where only the authorized user is allowed to use such important program and in any other cases the system will self destruct.

- Customer Relationship Management System

  Real life situations will not make greater usage of the embedded customer tracking system due to its prototype nature. In such situations we could build a complete Customer Relationship Management solution where more detailed information are stored for each user and the ability to sell products that benefit from the suggested techniques online. An additional automated customer support system is a welcomed idea. Many well known small features can be added to such system.

References:

1. Alcohol Soft. **Alcohol 120, http://www.alcohol-soft.com**, 2007.
2. Andrew, S. Tanenbaum. **Computer Networks**, 4th edition, Prentice Hall, 2002.
3. Ashileshwari, N. & Chandra, Liam, D. Comerford and Steve R. White, **Software Protection System Using a Single-Key Cryptosystem, a Hardware-Based**, IBM 1989.
4. Berson, Thomas A. (1992). "Differential Cryptanalysis Mod 2^{32} with Applications to MD5". *EUROCRYPT*: 71–80. **ISBN 3-540-56413-6**.
5. Bruce Schneier. **Applied Cryptography**, 2nd ed., p.312, p.415, John Wiley & Sons, 1996.
6. Bruce Schneier. **Schneier on Security: Sony's DRM Rootkit: The Real Story**, http://www.schneier.com/blog/archives/2005/11/sonys_drm_rootk.html, 2005.
7. BSA website **http://www.bsa.org/country/Anti-Piracy/What-is-Software-Piracy/Types%20of%20Piracy.aspx**, 2007.
8. Chang H. & Attallah, M. **Protecting Software Code by Guards**, in: Proceedings of the 1st International Workshop on Security and Privacy in Digital Rights Management, 2000, pp.160-175, 2000.
9. Collberg, C., Thomborson & Low, D. **A Taxonomy of Obfuscating Transformations**. Technical Report 148, University of Auckland, July 1997. http://www.cs.auckland.ac.nz/~collberg/Research/Publications/CollbergThomborsonLow97a/index.html.
10. Collberg, C., Myles & Huntwork, Sandmark. **A Tool for Software protection Research**, A. Arizona Univ., Tucson, AZ, USA, 2003.
11. Cowan, C., **Software Security for Open-Source Systems, IEEE Security and Privacy**, 2003.
12. Cullen, L. & Saumya, D. **Obfuscation of Executable Code to Improve Resistance to Static Disassembly**, Department of Computer Science University of Arizona, 2003.
13. Douglas, J. & Stephen, P. **Combating Software Piracy by Encryption and Key Management**, 1984.

14. Eldad, E. **Reversing: Secrets of Reverse Engineering**, pp.312-314, John Wiley & Sons, 2005.

15. Eset. **NOD32 Anti-Virus System**, **http://www.eset.com**, 2007.

16. Fiat. A. & Shamir, A. **How to Prove Yourself: Practical Solutions to Identification and Signature Problems**. In CRYPTO '86, vol.263 of LNCS, p.186–194, 1986.

17. IDC, **BSA/IDC Global Software Piracy Study**, Transitions Online, 2007.

18. Jon, H., Barber, R., Woodward, R., Burkley, E., Rehme, M., Jackson & Douglas, M. **System for Controlling the Number of Concurrent Copies of a Program in a Network Based on the Number of Available Licenses**, 1996.

19. Keet, E. **Preventing Piracy: Business Guide to Software Protection**, 1985.

20. Main, A. & Van Oorschot, C. **Software Protection and Application Security: Understand the Battleground**, Carleton University, 2003.

21. Matthew, S., Frank, H. & Ghosh, A. **Preventing the Execution of Unauthorized Win32 Applications**, DARPA Information Survivability Conference and Exposition (DISCEX II'01)Volume II-Volume 2, p.1175, 2001.

22. Microsoft, **MSDN – System-Management Namespace**, 2007.

23. Olga, G, Bhagirath, N. & Rahul, S. **SPEE, A Secure Program Execution Environment tool using code integrity Checking**, The George Washington University, Washington, *DC, USA*, Journal of High Speed Networks, 15(2006), 21-32, 2006.

24. Orlin, J. **The DES Algorithm Illustrated**, Laissez Faire City Times, 2(28), 44-60, 1997.

25. Samir, N. **Piracy and Terrorism in the Arab World**, Kuwait University, Transitions Online , 2006.

26. Sattar, J**. Baghdad Method for Calculating Multiplicative Inverse,** Information Technology: Coding and Computing, Proceedings. ITCC 2004. **International Conference**, 2(4), 816-819, 2004.

27. Sattar, J. **Software Piracy in Jordan**, Journal of Applied Science, 3(6), 34-45, 2001.

28. Sattar, J., Mustafa Al-Fayoumi. **An Efficient RSA Public Key Encryption Scheme,** Information Technology: New Generations, 2008. ITNG 2008. **Fifth International Conference**, p.127-130, 2008

29. Shengying, Li. **A Survey on Tools for Binary Code Analysis**, Stony Brook University, 2004.

30. Shin, S., Gopal, S., Lawrence, G., Whinston & Andrew, B. **Global Software Piracy Revisited, Communications of the ACM**, Jan2004, 47(1), 103-107, 2004.

31. Shub-Nigurrath. **Cracking with Loaders: Theory, General Approach and a Framework**, ARTeam, 2005.

32. StarForce. **http://www.star-force.com/**, 2007.

33. Tomas, S. & Christian Tschudin, F. **LNCS On Software Protection via Function Hiding**, , Transitions Online ,1998.

34. Wikipedia, **http://en.wikipedia.org/wiki/Copy_protection**, 2007.

35. Wikipedia, **http://en.wikipedia.org/wiki/Md5**, 2007.

36. Wikipedia,**http://en.wikipedia.org/wiki/Windows_Management_Instrume-ntation**, 2007.

37. www.aci.net/kalliste/des.htm.

38. Zheng C. & Xue, H. **Software Protection in China: A Complete Guide**, Transitions Online ,1999.

39. Zoeller & Renate. **Nest of Pirates**, Transitions Online, p.5-5, 2007

الملخص

تعاني الكثير من الدول من مشكلة القرصنة. ورغم وجود العديد من الأنظمة في الأسواق لحل هذه المشكلة، إلا أنها لم تصل إلى مستوى الحل المتكامل. بعد إجراء دراسة على الأنظمة الموجودة. و بعد التعرف على ميزاتها و سيئاتها. قام الباحث بتطوير نظام متكامل يعتمد على المواصفات القياسية مثل Zero knowledge proof, RSA, MD5, و Triple DES لحل هذه المشكلة. وبعد الدراسة تبين أن النظام فعال وديناميكي، وبأنه يعطي نتائج أكثر قبولاً من الأنظمة المتوفرة حالياً في الأسواق.

لقد قام الباحث بتطوير نظام يقوم بقراءة معلومات الأجهزة المتواجدة داخل الحاسوب، ومن ثم بتشفيرها. وبعد ذلك يتم التأكد من أن النسخة المستخدمة من قبل المستخدم لم يتم استخدامها من قبل مستخدمين آخرين، وذلك من خلال إرسال المعلومات إلى المصنع. لقد قام الباحث أيضاً بتطوير أداة تمكن من استخدام هذا النظام على أي برنامج متواجد في الأسواق.

Appendix 1 – Installation Manual

End user's Requirements:

1. .Net runtime 2.0
2. Internet Connection (Optional): to test the automatic mode.

Developer's Requirements:

1. Visual Studio 2005
2. Internet Connection: to test and setup the automatic mode.

The program requires installing a website and a webservice on the server.

Main Projects:

Project Location	Description
SplitJoin\fileSplitter.sln	The Encapsulator
Db\webmanagement\webmanagement.sln	The website, DB access Layer, webservice
Protect app\papp\ui client.sln	The protection Interface
ISCNgenerator\cdkeygenerator.sln	ISCN generator/tester

To set the projects to run from your machine:

1. copy all the files in the CD to the any location on your hard drive (for example: c:\khaldoon)
2. if you want to use this machine as a server to test the automatic mode, make sure that the database file is defined properly in the datalayer project. The default directory for the database is (C:\khaldoon\db\) if you want to change it open the website's project (Db\webmanagement\webmanagement.sln) and then open the sub project (dblayer\dblayer.cs) and change the line (string connectionstring = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\\khaldoon\\db\\access-prototype.mdb";) to the new location. After that rebiuld the website and the webservice (both are sub projects of (Db\webmanagement\webmanagement.sln)
3. the protection interface searchs for the "web service" in the default location http://localhost:4816/authservice. if you have changed the location of the service or the name of the server then you have to rebuild the protection interface and generate a new file that can connect to the correct location
4. to rebuild the protection interface open the project (Protect app\papp\ui client.sln) and then go to "Solution Explorer" then click on "webReferences\localHost" , after that right click on it and then select properties. When the properties window displays

change “web reference URL” property to the new location of the service. Now rebuild the application. The new “protect.exe” file will be found under the debug sub folder. Use it in the encapsulation process to access to the advised location.

5. To get some valid Copy IDs which are stored in the database open the file “copy id.txt”.

Preparing a test folder:

To do the tests you need to have all of the required libraries in the proper location.

There is already a folder on the cd built with the default options in mind. You can find it under (Tests) directory.

The required files are:

File name	File description
Encapsulator.exe	The encapsulator Application
Protect.exe	Protection interface file.
Corruptor.exe	A program that tampers any given file on purpose.
Debugwindow.dll	Library to display debug window
EncDec.dll	TDES encryption library
RSAEncDec.dll	RSA encryption library
Protocol.dll	Zero knowledge protocol library
iniReader.dll	Ini files reader library
Hardwareserial.dll	Hardware information library.
Lauch.dll	Application launching library
SampleApp.exe	Sample application (to test it)
Reg.ini	Informations file –Optional(for automatic mode only)

To test the automatic Mode:

1. Make sure that the settings are well configured as mentioned above. (or copy all the files to the default location)
2. Open the website project (Db\webmanagement\webmanagement.sln)
3. Run the project by choosing debug from the menu.
4. Make sure that you have internet connection even if you’re testing it on a local machine.
5. Go the test folder and run the application.

Appendix 2 – Major Code

RSA CODE:

```
    public class RSAProvider
    {

        Random randomizer = new Random();
        public long p, q, n, Theta;
        private long g,d;
        public long  e;

        public void pickRandomTwoPrimes()
        {

            this.p = 43;
            this.q = 47;
            n = p * q;///statticlly assigned

            g = (p * p - 1) * (p * p - p) * (q * q - 1) * (q * q - q);

        }


        long modpow(long b, long e, long m)
        {

            long result = 1;
            while (e > 0)
            {
                if ((e & 1) == 1) result = (result * b) % m; //
multiply in this bits' contribution while using modulus to keep result
small
                e >>= 1;
                b = (b * b) % m;
             }

            return result;
        }
        public void generateKeys()
        {

            long num = randomizer.Next((n / 2));
            while (GCD(num, g) != 1)
            {
                num = randomizer.Next((n / 2));
            }
            e = num;
            d = generate_d_baghdad(e);

        }
        public long encrypt(long m)
        {
            //c=(m^e) mod n
            long a = m;
            a = modpow(m, e, n);
            return a;
        }
        public long decrypt(long c)
        {
            //m=(c^d) mod n
```

```
            long a = c;
            a = modpow(c, d, n);
            return a;
        }


        public long GCD(long a, long b)
        {
            long Remainder;

            while (b != 0)
            {
                Remainder = a % b;
                a = b;
                b = Remainder;
            }

            return a;
        }

        public void setPublicKey(long nn, long ee)
        {
            n = nn;
            e = ee;
        }

//baghdad inverse calcuclation
        public long generate_d_baghdad(long e)
        {


            //Compute Theta
            //this function uses baghdad algorithm to compute the
inverse of d


            double Theta = g;
            long sqr = e;
            double x = 1;
            double D = 1;

            do
            {
                x = x + Theta;
                D = x / sqr;

            } while (x % sqr != 0);

            return (long)D;

        }


    }
TRIPLE DES CODE :
//this is the code that I used
EncDec.cTripleDES enc = new EncDec.cTripleDES ();
enc.Encrypt(part1);

//it depends on the following library (built in visual studio .NET
2005)
```

```
Public Class cTripleDES

    ' define the triple des provider

    Private m_des As New TripleDESCryptoServiceProvider

    ' define the string handler

    Private m_utf8 As New UTF8Encoding

    ' define the local property arrays

    Private m_key() As Byte
    Private m_iv() As Byte

    Public Sub New(ByVal key() As Byte, ByVal iv() As Byte)
        Me.m_key = key
        Me.m_iv = iv
    End Sub

    Public Sub New()
        Dim key() As Byte = _
  {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, _
  15, 16, 17, 18, 19, 20, 21, 22, 23, 24}
        Dim iv() As Byte = {8, 7, 6, 5, 4, 3, 2, 1}
        Me.m_key = key
        Me.m_iv = iv

    End Sub
    Public Function Encrypt(ByVal input() As Byte) As Byte()
        Return Transform(input, m_des.CreateEncryptor(m_key, m_iv))
    End Function

    Public Function Decrypt(ByVal input() As Byte) As Byte()
        Return Transform(input, m_des.CreateDecryptor(m_key, m_iv))
    End Function

    Public Function Encrypt(ByVal text As String) As String
        Dim input() As Byte = m_utf8.GetBytes(text)
        Dim output() As Byte = Transform(input, _
                        m_des.CreateEncryptor(m_key, m_iv))
        Return Convert.ToBase64String(output)
    End Function

    Public Function Decrypt(ByVal text As String) As String
        Dim input() As Byte = Convert.FromBase64String(text)
        Dim output() As Byte = Transform(input, _
                         m_des.CreateDecryptor(m_key, m_iv))
        Return m_utf8.GetString(output)
    End Function

    Private Function Transform(ByVal input() As Byte, _
        ByVal CryptoTransform As ICryptoTransform) As Byte()
        ' create the necessary streams

        Dim memStream As MemoryStream = New MemoryStream
        Dim cryptStream As CryptoStream = New _
            CryptoStream(memStream, CryptoTransform, _
            CryptoStreamMode.Write)
        ' transform the bytes as requested
```

```
            cryptStream.Write(input, 0, input.Length)
            cryptStream.FlushFinalBlock()
            ' Read the memory stream and convert it back into byte array

            memStream.Position = 0
            Dim result(CType(memStream.Length - 1, System.Int32)) As Byte
            memStream.Read(result, 0, CType(result.Length, System.Int32))
            ' close and release the streams

            memStream.Close()
            cryptStream.Close()
            ' hand back the encrypted buffer

            Return result
        End Function

    End Class
```

ISCN CODE:

```
public string generateICSN(string langCode,string manCode,string
iNumber)
            {

                  string allDigits=langCode+manCode + iNumber;
                  //to calcualte checkdigit
                  //1. multiplication mod 11
                  //2. nearest integer to make the mod divides 11

                  long T=0;
                  for (int i=9;i>=1;i--)
                  {
                        T+=Int16.Parse( allDigits[9-
i].ToString())*(i+1);

                  }
                  T=T%11;
                  long cdigit=11-T;
                  string cd="";
if (cdigit==10) cd="X"; else if (cdigit==11) cd="A"; else
cd=cdigit.ToString();
                 string ICSN=langCode +"-"+manCode+"-"+iNumber+"-"+cd;
                  return ICSN;
            }


//VALIDATE CHECKDIGIT
            private void button2_Click_1(object sender,
System.EventArgs e)
            {

                  try
                  {
            string inputICSN= tCode.Text.Replace("-","");
            string cd=inputICSN[inputICSN.Length -1].ToString();
                        int cDigit=0;
                        if (cd.Equals("X") || cd.Equals("x") )
cDigit=10; else if (cd.Equals("A") || cd.Equals("a") ) cDigit=11;else
cDigit=Int16.Parse(cd );

            int correctCDigit=0;
                  string
allDigits=inputICSN.Substring(0,inputICSN.Length -1);

                  int T=0;
                  for (int i=9;i>=1;i--)
                  {
                        T+=Int16.Parse( allDigits[9-
i].ToString())*(i+1);

                  }
                  T=T%11;
                  correctCDigit=11-T;


                  if (correctCDigit.Equals (cDigit))
                  {
                        status.Text="valid";
```

```
                    }else status.Text="inValid, check digit should be =
"+correctCDigit.ToString();
                    }
                    catch (Exception ex)
                    {
                          status.Text="inValid";
                    }
```

Zero Knowledge Code:

```
   public  class prover
{

       Random randomizer = new Random();
        public int p,q,n,k,r,x,ssize,Td,y;
    int[] d=new int[15];
    int []e= new int[15];
       int[] s= new int [15];//[100],e[100],s[100];

    public void pickRandomTwoPrimes()
    {

         this.p = 59;
         this.q = 47;
         n = p * q;///statticlly assigned
    }

    public void pickTheValueOf_k(int val)
    {
        this.k=val;
    }
       public int[] get_eA()
       {
           return e;
       }

       public  int GCD(int a, int b)
       {
           int Remainder;

           while (b != 0)
           {
               Remainder = a % b;
               a = b;
               b = Remainder;
           }

           return a;
       }
       public void generate_e_baghdad_take3(int[] val)
       {

           //Compute Theta
           //this function uses baghdad algorithm to compute the
inverse of d^2

           double Theta = (p - 1) * (q - 1);
           Theta = n;
           for (int i = 0; i < k; i++)
           {


               //d[i]=num;
               d[i] = val[i];
               int num = d[i];

               //long  sqr = num*num ;
               int sqr = num * num;// ((num % n) * (num % n)) % n;
               if (GCD(n, sqr) == 1)
```

```
                {
                    double x = 1;
                    double D = 1;
                    do
                    {
                        x = x + Theta;
                        D = x / sqr;

                    } while (((x-2) % sqr != 0)&&(x % sqr != 0)) ;

                    e[i] = (int)D;

                }
            }

        }
    public void choose_r_cal_x()
    {
        r = randomizer.Next(100)+ 1;
        x = (r*r)%n;
    }

    int getx()
    {
        return x;
    }

    public void sets(int[] p)
    {
         for(int i=0;i<ssize;i++)
             s[i]=p[i];
    }

    public void setssize(int ssize)
    {
        this.ssize=ssize;
    }

    public void cal_Td()
    {
        Td=1;
        for(int i=0;i<ssize;i++)
            Td=Td*d[s[i]];
    }

    public void cal_y()
    {
         y=(r*Td)%n;
    }


};

public class vrifier
{
    Random randomizer = new Random();
public       int n,x,y,k;
    int [] e= new int[15];
    int [] s = new int[15];
    public int Te, ssize;
```

```
public void setn(int n)
{
     this.n=n;
}

public void setx(int x)
{
     this.x=x;
}

public void setk(int k)
{
     this.k=k;
}

public void sete(int[] temp)
{
     for(int i=0;i<k;i++)
         e[i]=temp[i];
}

public void pick_subset_s()
{
     int temp=5;
     if(k<=5)temp=k;
     int num = randomizer.Next(temp) + 1,j;
     ssize=num;
     for(int i=0;i<num;i++)
     {
         s[i]=randomizer.Next(k);
         while(true==true)
         {
             for(j=0;j<i;j++)
                 if(s[j]==s[i])break;
             if(i==j)
                 break;
             s[i]=randomizer.Next(k);
         }
     }
}

public int [] get_s()
{
     //int *p=s;
     //return p;
    return s;
}

public void cal_Te()
{
     Te=1;
     for(int i=0;i<ssize;i++)
         Te=Te*e[s[i]];
}

public void sety(int y)
{
     this.y=y;
}

public bool verify()
```

```
        {
        //      cout<<endl;
        //      cout<<"x%n = "<<x%n<<endl;
        //      cout<<"y*y*Te+x%n = "<<(y*y*Te+x)%n<<endl;

if(x%n==((y%n)*(y%n)*(Te%n))%n||x%n+((y%n)*(y%n)*(Te%n))%n==n)
                return true;
             return false;
        }

};
```

MD5 code:

```
MD5 myHash = MD5.Create(); ;

byte[] hash = myHash.ComputeHash(bin);
string newhash = BitConverter.ToString(hash); //to read it as string
```

```
Getting Machine ID code:
//HDD2 method 2 wmi
            SelectQuery query = new SelectQuery("Win32_DiskDrive");
            ManagementObjectSearcher searcher = new
ManagementObjectSearcher(query);
            ManagementObjectCollection coll = searcher.Get();

            foreach (ManagementObject obj in coll)
            {
                lHDD.Text =
obj.Properties["Signature"].Value.ToString();
                loggerClass.addText("HDD ID= " + lHDD.Text);
            }
            //cpu id
            query = new SelectQuery("Win32_Processor");
            searcher = new ManagementObjectSearcher(query);
            coll = searcher.Get();

            foreach (ManagementObject obj in coll)
            {
                lCPU1.Text =
obj.Properties["ProcessorID"].Value.ToString();
                loggerClass.addText("CPU ID= " + lCPU1.Text);
            }

            //bios
            query = new SelectQuery("Win32_BIOS");
            searcher = new ManagementObjectSearcher(query);
            coll = searcher.Get();

            foreach (ManagementObject obj in coll)
            {
                lBIOS.Text =
obj.Properties["SerialNumber"].Value.ToString();
                loggerClass.addText("BIOS ID= " + lBIOS.Text);
            }
```

```
int codeLength = 6;
            String part1 = "", part2 = "", part3 = "";

part1 = lCPU1.Text.Substring(0,codeLength);
part2 = lHDD.Text.Substring(0, codeLength);
part3 = lBIOS.Text.Substring(0, codeLength);
string machineID = part1 + "-" + part2 + "-" + part3;
```

```
//Installation ID
EncDec.cTripleDES enc = new EncDec.cTripleDES ();

                        outCode = enc.Encrypt(part1) +"-"
+enc.Encrypt(part2)+"-"+enc.Encrypt(part3);
```

IP ADDRESS CODE:

```
          // Getting Ip address of local machine...
                // First get the host name of local machine.
                        strHostName = System.Net.Dns.GetHostName ();

          // Then using host name, get the IP address list..
                        IPHostEntry ipEntry =
System.Net.Dns.GetHostByName(strHostName);
          IPAddress [] addr = ipEntry.AddressList;
                                  string ip=addr[0].ToString ();
```

Activation Code (code):

```
public void calculateActivationCode()
        {
            //genrate activation code
            try
            {
                String part1 = "", part2 = "", part3 = "";
                int codeLength = 6;
                string outCode;
                part1 = cpuID.Substring(0, codeLength);
                part2 = HDDID.Substring(0, codeLength);
                part3 = BiosID.Substring(0, codeLength);
                EncDec.cTripleDES enc = new EncDec.cTripleDES();
                outCode = enc.Encrypt(part1) + "-" + enc.Encrypt(part2)
+ "-" + enc.Encrypt(part3);
                //hash result
                byte[] bText = Encoding.Unicode.GetBytes(outCode );
                MD5 hash = new MD5 ();
                byte[] bEncText = hash.ComputeHash(bText);
                outCode= Convert.ToBase64String(bEncText);
                activationCode = outCode;

                errorCode = 1;//all ok
            }
            catch
            {
                errorCode = 2;
            }
        }
```

Encapsulator Code:

```
        private void JoinFiles(string FolderInputPath, string
FileOutputPath)
        {
            //joining files in directory into one file

            FileStream fsSource = new FileStream(FileOutputPath,
FileMode.Append);
            for(int i =0 ;i<=2;i++)
                {
                    Byte[] bytePart =
System.IO.File.ReadAllBytes("file.000"+i.ToString ()+".part");

                fsSource.Write(bytePart, 0, bytePart.Length);
                }

            fsSource.Close();
        }
```

Encapsulation process:

```
                //prepare the 3 files to be combined
                //file 1 : protection interface
                File.Copy("protect.exe", "file.0000.part",true);
                // File 2: file to protect
                File.Move(txtSourceFile.Text, "file.0002.part");
                //file 3: checksum file
                //create file
                System.IO.StreamWriter fs = new
System.IO.StreamWriter("file.0001.part", false);
                //write to it

                //file length
                // 1. get file length
                FileStream fs2 = new FileStream("file.0002.part",
FileMode.OpenOrCreate);

                byte[] fs2Bytes = new byte[fs2.Length ];

                fs2.Read(fs2Bytes, 0, (int)fs2.Length);
                //2. get checksum
                MD5 myHash;

                myHash = MD5.Create();


                byte[] arr2 = myHash.ComputeHash(fs2Bytes);



                    //write
                loggerClass.addText("Original Length in bytes = " +
fs2.Length);
                loggerClass.addText("Checksum value = " +
BitConverter.ToString(arr2));
                loggerClass.addText("file 3: checksum file");

                    //fs.WriteLine(fs2.Length);
                    fs.WriteLine(BitConverter.ToString(arr2));
                    fs.Close();
```

```
            fs2.Close();
            loggerClass.addText("Joining Files in one file");
        JoinFiles(folder, txtSourceFile.Text);
        loggerClass.addText("deleting old files");
        //delete files
        File.Delete("file.0000.part");
        File.Delete("file.0001.part");
        File.Delete("file.0002.part");
```

Authentication Code:

```
//the stored password is e(a)
            protocol.prover p = new protocol.prover();
            p.pickRandomTwoPrimes();//n always equals to 2773
            string pw = txtPassword.Text;
            p.pickTheValueOf_k(pw.Length);

            int[] passwordArray =
protocol.utils.convertTextToArray2(pw);
//            p.choose_k_random_numbers(passwordArray);
            p.generate_e_baghdad(passwordArray);
            p.choose_r_cal_x();

            protocol.vrifier v1= new vrifier ();
            v1.setn(p.n);

            v1.setx(p.x);
            v1.setk(p.k);
            //v1.sete(p.e);

v1.sete(protocol.utils.convertTextToArray( db.getPasswordForCustomer(t
xtUserName.Text )));
            String b = db.getPasswordForCustomer(txtUserName.Text );

            v1.pick_subset_s();
            v1.cal_Te();


            p.setssize(v1.ssize);
            p.sets(v1.get_s());

            p.cal_Td();
            p.cal_y();


            v1.sety(p.y);
            bool allClear = true;
            if (v1.verify() == false) allClear = false;
            //round 2
            p.choose_r_cal_x();
            v1.setx(p.x);
            v1.setk(p.k);
            v1.pick_subset_s();
            v1.cal_Te();
            p.setssize(v1.ssize);
            p.sets(v1.get_s());
            p.cal_Td();
            p.cal_y();
            v1.sety(p.y);

            if (v1.verify() == false) allClear = false;

            //round 3
            p.choose_r_cal_x();
            v1.setx(p.x);
            v1.setk(p.k);
            v1.pick_subset_s();
            v1.cal_Te();
            p.setssize(v1.ssize);
            p.sets(v1.get_s());
```

```
p.cal_Td();
p.cal_y();
v1.sety(p.y);
if (v1.verify() == false) allClear = false;

if (allClear==true )
{
    //valid user
    int customerID = db.getCustomerID(txtUserName.Text);
    Session["customerID"] = customerID;
    Response.Redirect("items.aspx");

}
else
{
    lblResponse.Text = "Invalid User name or Password";
}
```