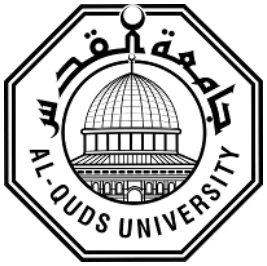
Deanship of Graduate Studies

Al-Quds University



Distributed Obfuscation Model for Software Protection (DOSP)

Mai Kamel Atef Amro

M.Sc. Thesis

Jerusalem-Palestine

1443/2022

Jerusalem – Palestine

Distributed Obfuscation Model for Software Protection (DOSP)

Prepared By:

Mai Kamel Atef Amro

B.Sc.: Computer System Engineering

Palestine Polytechnic University, Palestine

Supervisor: Dr. Rushdi Hamamreh

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Electronic and Computer Engineering, Faculty of Engineering at Al-Quds University

1443/2022

Al-Quds University

Deanship of Graduation Studies

Electronic and Computer Engineering

Thesis Approval

Distributed Obfuscation Model for Software Protection (DOSP)

Prepared By: Mai Kamel Atef Amro

Registration No.: 21710048

Supervisor: Dr. Rushdi Hamamreh

Master thesis submitted and accepted, Date: 09-01-2022

The names and signatures of the examining committee members are as follows:

1- Head of Committee: Dr. Rushdi Hamamreh. **Signature** _____.

2 -Internal Examiner: Dr. Nidal Kafri. **Signature** _____.

3 -External Examiner: Dr. Iyad Tumar. **Signature** _____.

Jerusalem – Palestine

1443/2022

Declaration

I Certify that this thesis submitted for the Degree of Master is the result of my research, except where otherwise acknowledged, and that this thesis (or any part of the same) has not been submitted for a higher degree to any other university or institution.

Signed by: 

Mai Amro

Date: 09-01-2022

Dedication

I dedicate this work to my great parents and my husband

Mai Amro

Acknowledgment

My work would never have been possible without Allah and the help of many people during my research. I would like to give special thanks to my supervisor for this project, Dr. Rushdi Hamamreh, for his guidance, support, and encouragement through my research semesters.

Special thanks to my great sister, Dima for all her support, help, and encouragement.

Deeply thanks to my husband, father, and mother for their infinite giving support and encouragement. I would also like to thank my dear sisters Reem and Renad.

ABSTRACT

This study suggests a new Distributed Obfuscation Model for Software Protection (DOSP-AES). DOSP-AES is proposed as a method of protecting software from a reverse engineering analysis. DOSP-AES is made up of software processes that are obfuscated and de-obfuscated. Three levels of obfuscation techniques are used in the obfuscated software process. DOSP-AES is applied to C++, Java, and Android programs.

The first level, **name obfuscation**, involves renaming identifiers and variables with meaningless names and obfuscating them using the AES algorithm with a random key length of 128 bits. Methods for obfuscating code include the renaming and removing process. Removing means removing unnecessary debugging information, methods, comments, and structures from the program. The transformation of a program's variables, constants, class, method names, and other identifiers to prevent attackers from understanding and analyzing it is referred to as renaming.

The second level**, data obfuscation,** proposed concealing data values by changing the statements in which variables are defined and used. DOSP-AES encrypts the values of constants, local and global program variables to complicate reverse engineering and protect sensitive data from disclosure. Data obfuscation with the AES algorithm and a key length of 256 bits. The most important aspect is that DOSP-AES obfuscates each variable differently from the other when it is mentioned on more than one site with the same application; each variable appears in a different form from the other, despite the fact that they are the same variable.

The third level, **bytecode obfuscation**, where bytecode is modified so that after the bytecode is compiled, it contains obscure compilation errors, but the compiled Java program still functions as expected. Java is compiled into bytecode. Decompiling bytecode files is easy because of the

names, fields, and methods within them. Obfuscation is one of the major defenses against decompilation. The goal of bytecode obfuscation is to make the decompiled program much harder to understand so that the attacker must spend more time Sand effort on the obscure bytecode. Identifier names of bytecode are replaced with illegal obfuscated identifiers, which cause syntax errors and compilation errors when decompiling. The DOSP-AES algorithm encrypts identifiers and class names in bytecode files with a key length of 192 bits using the AES algorithm.

By obfuscating the code on multiple levels, the attacker will have a more difficult time analyzing and analyzing the code.

The de-obfuscated process is a client-server model (distributed system), where clients download the obfuscated software and applications that were uploaded to the internet. The client requests the server for the obfuscation key to de-obfuscate the software, then the server complies by sending the secure key (obfuscated key) as one block of 72 bytes. The key is randomly generated using a key generator (keygen) in the Crypto++ simulator. There are three levels of obfuscation in each subkey. The first level (name obfuscation) is de-obfuscated using 16-byte key lengths, the second Level (data obfuscation) is de-obfuscated using 32-byte key lengths, and the third level (bytecode obfuscation) is de-obfuscated using 24-byte key lengths.

Thus, the experiment has produced successful and promising results since it is difficult for reverse engineering tools to read and analyze the obfuscated code. Even the revealed code did not perform as well as the original and obfuscated code.

TABLE OF CONTENTS

Declaration I
Dedication II
Acknowledgment III
ABSTRACT IV
TABLE OF CONTENTS VI
List of Figures XI
List of Tables XIII
List of Acronyms XIV
Chapter One 1
Introduction 1
1.1 Motivation and Problem Statement 2
1.2 Research Aims and Objectives 3
1.3 DOSP-AES Security Threat Model 3
1.3.1 Man in The Middle (MITM) attack 4
1.3.2 String Detection 4
1.3.3 Multiple Input Binary Files 5
1.3.4 One Input Binary File 5
1.4 Contributions 5
1.5 Thesis Organization 7
Chapter Two 8
Literature Review and Background 8
2.1 Literature Review 9
2.2 Software Development Tools 14
2.2.1 Compiler 14
2.2.2 Interpreter 17
2.2.3 Compiler Vs Interpreter 17
2.3 Cryptosystem Models 18

2.4 Cryptographic Techniques 19
2.4.1 Asymmetric Encryption (public-key encryption) 19
2.4.1.1 Rivest–Shamir–Adleman Algorithm (RSA) 20
2.4.1.2 Diffie –Hellman Algorithm (DH) 22
2.4.1.3 Elliptic Curve Cryptography Algorithm (ECC) 23
2.4.2 Symmetric Encryption (Private Key Encryption) 24
2.4.2.1 Advanced Encryption Standard (AES) 24
2.4.2.2 RC4 Algorithm 27
2.4.2.3 Triple DES (TDES) 28
2.5 Performance Comparison of Cryptosystem Model 30
2.6 Summary 32
Chapter 3 33
Reverse Engineering and Obfuscation Techniques 33
3.1 Reverse Engineering 34
3.2 Reversing Methodologies 34
3.2.1 Offline Code Analysis 35
3.2.2 Live (Online) Code Analysis 35
3.3 Reverse Engineering Tools 35
3.3.1 Disassemblers 35
3.3.2 Debuggers 36
3.3.3 Decompiler 36
3.3.4 System Monitoring 36
3.4 Technical Solution to Prevent Reverse Engineering 37
3.4.1 Code Encryption 37
3.5 Obfuscation Process 38
3.6 Obfuscation Techniques 38
3.6.1 Name Obfuscation 38
3.6.2 Data Transformation 39
3.6.3 Bytecode Obfuscation 39
3.6.4 Obfuscation of Assembly Code Instructions 39
3.6.5 Anti-Debug 40

3.7 Characteristic of Success Obfuscation Techniques 40
3.8 Summary 41
Chapter Four 42
Proposed Model Distributed Obfuscation Model for Software Protection (DOSP) 42
4.1 Introduction to DOSP-AES.....43
4.1.1 Generate Symbol Table..... 43
4.1.2 Obfuscated Process of DOSP-AES..... 48
4.1.3 De- Obfuscated process (DO) of DOSP-AES 51
4.2 Introduction to DOSP-RSA 57
4.2.1 Obfuscation process of DOSP-RSA 57
4.2.1.1 First level: Source code (Name, layout) obfuscation..... 57
4.2.1.2 Second level: Data 57
4.2.1.3 Third level: Bytecode obfuscation..... 57
4.2.2 De-obfuscation process of DOSP-RSA 58
4.3 Summary 58
Chapter Five.....59
Results and Analysis.....59
5.1 Results for Generate Symbol Table 60
5.2 All the Trials during prepare our model 61
5.2.1 First Trial using normal implementation of Encryption Algorithm..... 61
5.2.2 The Second Trial using Rajindi library and TPF_Math_Library..... 65
5.2.3 The Third Trial Using Chilkat Library 66
5.2.4 The Fourth Trial using crypto++ library..... 66
5.3 Analysis of DOSP-AES model 67
5.3.1 DOSP-AES model for C++ programs 67
5.3.2 DOSP-AES model for Java programs..... 69
5.3.3 DOSP-AES model for Android programs 71
5.4 Analysis of DOSP-RSA model.....73
5.4.1 DOSP-RSA model for C++ programs 73
5.4.2 DOSP-RSA model for Java programs 75

5.4.3 DOSP-RSA model for Android programs 77
5.5 Comparison between DOSP-AES and DOSP-RSA Depend on the Time in all levels 79
5.5.1 Comparison in the 1st level (Name Obfuscation)..... 79
5.5.2 Comparison in the 2nd level (Data Obfuscation) 82
5.5.3 Comparison in the 3rd level (Bytecode/machine code Obfuscation) 85
5.6 Comparison between DOSP-AES and DOSP-RSA Depend on the key's length 89
5.6.1 Comparison between DOSP-AES and DOSP-RSA using different key lengths in C++ programs in the obfuscation process. 89
5.6.2 Comparison between DOSP-AES and DOSP-RSA using different key lengths in C++ programs in the De-obfuscation process. 90
5.6.3 Comparison between DOSP-AES and DOSP-RSA using different key lengths in Java programs in the obfuscation process. 91
5.6.4 Comparison between DOSP-AES and DOSP-RSA using different key lengths in Java programs in the De-obfuscation process. 92
5.6.5 Comparison between DOSP-AES and DOSP-RSA using different key lengths in Android programs in the obfuscation process. 93
5.6.6 Comparison between DOSP-AES and DOSP-RSA using different key lengths in Android programs in the De-obfuscation process. 94
5.7 Comparison between DOSP-AES and DOSP-RSA Depend on the Programming languages C++, Java, Android.....95
5.7.1 1st level: Name Obfuscation..... 95
5.7.2 2nd level: Data Obfuscation..... 96
5.7.3 3rd level: Bytecode Obfuscation..... 97
5.8 Brute force attack.....98
5.9 Attack model.....101
5.10 Summary 102
Chapter Six.....103
Conclusion and future work.....103
6.1 Conclusion 104
6.2 Future Work 107
Bibliography.....109

Appendix A 114

List of Figures

Figure Number	Figure Name	Page No
Figure 2.1	Life cycle of a computer program	12
Figure 2.2	Phase of compiler	13
Figure 2.3	Cryptography	15
Figure 2.4	A simplified model for asymmetric encryption	17
Figure 2.5	A simplified model for symmetric encryption	20
Figure 2.6	AES general encryption process	21
Figure 2.7	AES structure encryption process	22
Figure 2.8	Triple DES encryption process	25
Figure 2.9	Comparison between AES, RSA, TDES, RC4	27
Figure 2.10	Comparison between RSA and Diffie-Hellman Key Generation	27
Figure 4.1	Symbol table result	
Figure 4.2	workflow of DOSP-AES	40
Figure 4.3	DOSP-AES Block Diagram	41
Figure 4.4	General Block diagram of DOSP-AES model (Obfuscation and De-Obfuscation process)	
Figure 4.5	Obfuscation process of DOSP-AES	42
Figure 4.4	Name obfuscation	43
Figure 4.5	Data obfuscation	43
Figure 5.1	Obfuscation time in Source code level using AES algorithm	52
Figure 5.2	De- Obfuscation time in Source code level using AES algorithm	53
Figure 5.3	Obfuscation time in machine code level using AES algorithm	53
Figure 5.4	De-Obfuscation time in machine code level using AES algorithm	54
Figure 5.5	Obfuscation and Deobfuscation of DOSP-AES for C++ programs	56
Figure 5.6	Obfuscation and Deobfuscation of DOSP-AES for Java programs	58
Figure 5.7	Obfuscation and De-Obfuscation of DOSP-AES for Android programs	59
Figure 5.8	Obfuscation and Deobfuscation of DOSP-AES for C++ programs	61
Figure 5.9	Obfuscation and Deobfuscation of DOSP-AES for Java programs	64
Figure 5.10	Obfuscation and De-Obfuscation of DOSP-AES for Android programs	64

Figure 5.11	Name obfuscation technique of C++ programs	65
Figure 5.12	Name obfuscation technique of Java programs	66
Figure 5.13	Name obfuscation technique of Android programs	67
Figure 5.14	Data obfuscation technique of C++ programs	68
Figure 5.15	Data obfuscation technique of Java programs	69
Figure 5.16	Data obfuscation technique of Android programs	71
Figure 5.17	C++ Machine code obfuscation technique using 192-bit key length	71
Figure 5.18	Java bytecode obfuscation technique using 192-bit key length	72
Figure 5.19	Android bytecode obfuscation technique using 192-bit key length	73
Figure 5.20	Obfuscation of DOSP-AES and DSP2 using different keys length in C++ programs	74
Figure 5.21	Deobfuscation of DOSP-AES and DOSP-RSA using different keys length in C++ programs	75
Figure 5.22	Obfuscation of DOSP-AES and DOSP-RSA using different keys length in Java programs	76
Figure 5.23	Deobfuscation of DOSP-AES and DOSP-RSA using different keys length in Java programs	77
Figure 5.24	Obfuscation of DOSP-AES and DOSP-RSA using different keys length in Android programs	78
Figure 5.26	Name obfuscation using different key lengths in C++, Java, Android programs	98
Figure 5.27	Data obfuscation using different key lengths in C++, Java, Android programs	99
Figure 5.28	Machine-code/bytecode obfuscation using different key lengths in C++, Java, Android programs	99

List of Tables

Table Number	Table Name	Page Number
Table 2.1	Comparison Between Symmetric and Asymmetric Encryption	25
Table 5.1	Results of Obfuscation and De-Obfuscation process of the First Trial	52
Table 5.2	Obfuscation and De-Obfuscation time for DOSP-AES using different file sizes of C++ programs	55
Table 5.3	Obfuscation and De-Obfuscation time for DOSP-AES using different file sizes of Java programs	57
Table 5.4	Obfuscation and De-Obfuscation time for DOSP-AES using different file sizes of Android programs	58
Table 5.5	Obfuscation and De-Obfuscation time for DOSP-RSA using different file sizes of C++ programs	60
Table 5.6	Obfuscation and De-Obfuscation time for DOSP-RSA using different file sizes of Java programs	61
Table 5.7	Obfuscation and De-Obfuscation time for DOSP-RSA using different file sizes of Android programs	63
Table 5.8	Name obfuscation (Source code) technique	65
Table 5.9	Data obfuscation technique	67
Table 5.10	Bytecode / machine obfuscation technique	70
Table 5.11	Time to crack AES various key using Brute force analysis	81

List of Acronyms

DOSP-AES	Distributed obfuscation model technique for software protection
IDA	Interactive Disassembler
RC4	Rivest Cipher 4
DH	Diffie Hellman
RSA	Rivest–Shamir–Adleman Algorithm
AES	Advanced Encryption Standard
k_E	Encryption Key
K_D	Decryption Key
k_{Public}	Public key
K_s	Secret key
gcd	greatest common divisor
mod	Module's arithmetic
ECC	Elliptic Curve Cryptography algorithm
TDES	Triple Data Encryption Standard
P	Plain text (original program)
O(P)	Obfuscation Text/code
K_{Obf}	Obfuscation key
K_{DO}	De-Obfuscation key
BO	Bytecode Obfuscation
K_{256}	Key length 256 bit
K_{192}	Key length 192 bit
K_{128}	Key length 128 bit
K_{72}	Key length 72 byte

Chapter One

Introduction

Chapter One: Introduction

This chapter introduces the purpose of this thesis. It presents the research motivations, problem statements, aims and objectives, threat model, contributions, and finally, the thesis organization is presented.

1.1 Motivation and Problem Statement

Over the last years, a lot of software and programs have been suffering from copyright violations, as this software required hard work, a lot of time, intelligence, and a lot of money. It is estimated that software protection against piracy will cost billions of dollars, considering how much intellectual property and copyright is included within the software. Besides software piracy, many tools can provide illegal access to software data and enable reverse engineers and adversaries to analyze the software and steal the intellectual property.

It is the distribution of software (Client/Server) over the client devices that poses the biggest problem to software protection since the owners lose control. In recent years, client devices have become more powerful, allowing an attacker with malicious intent to violate copyrights and tamper with software using several analyses and reverse engineering tools. Illegal access could be obtained when the software is being cracked, where illegal copying and distributing of cracked software is a form of copyright infringement. The need for robust software protection techniques against reverse engineering analysis is highly needed today. These methods should address the lack of confidence in software in an untrusted environment.

There have been many studies on one-to-one protection. However, there are few studies evaluating many-to-one protection, which protects intellectual property and is treated as trade secret.

1.2 Research Aims and Objectives

1. Protect the intellectual property, licensing mechanisms, copyright and unauthorized modification of the software from unauthorized access and reverse-engineering analysis.
2. Develop robust techniques that integrate multi levels of obfuscations. Traditional techniques depend on only one level of obfuscation and these techniques suffered from reverse engineering analysis. So, depending on multi-levels of obfuscation makes reverse engineering analysis more complex.
3. To ensure data confidentiality from any type of reverse engineering while distributing the software over client device, and while transmitting data through network, so users can securely install all components of application and use it securely without any modifications during distribution of the software over client device.

1.3 DOSP-AES Security Threat Model

The purpose of threat modeling is to label and preserve the import utilities of a software model. A variety of attackers can access the software, particularly through reverse engineering. Software protection techniques are suffering from many threats that the DOSP-AES model will solve:

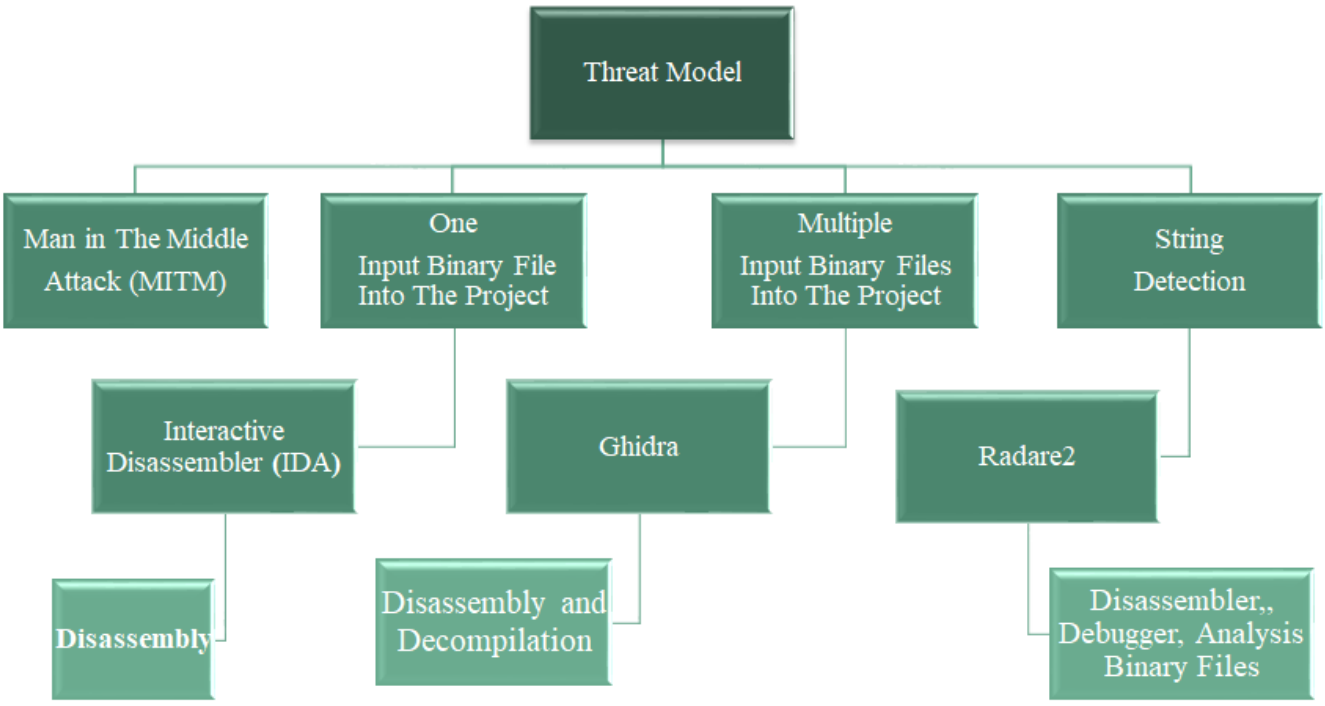


Figure 1.1: DOSP-AES Security Threat Model

1.3.1 Man in The Middle (MITM) attack

A MITM is a type of eavesdropping attack, in which attackers can read, write and alter secret information without any trees of manipulation [65].

1.3.2 String Detection

Radare2 is one of the reverse engineering techniques used for string detection. As part of radare2, disassemblers and debugging tools are made. In addition to running on both Linux and Windows, this software can also be scripted in Python and JavaScript. And perform disassembly, debugging, analysis, comparison, and manipulation of binary files [6].

1.3.3 Multiple Input Binary Files

Ghidra is an open-source reverse engineering tool. A potential attacker can load and analyze multiple binary files of this software at the same time using this software. This software can load and analyze multiple binary files at the same time. Gidra runs on many platforms such as Windows, macOS, and Linux. As part of this software, disassembly and decompilation are performed. Also, it provides an array of process instruction sets and executable formats that can run in interactive or automated modes [6].

1.3.4 One Input Binary File

Interactive Disassembler (IDA) is one of the most popular tools of the reverse engineering process. IDA tools load only one binary file at a time. IDA makes disassembler of software and supports a number of executable formats for operating systems and processors. IDA has many advantages that enable the user to change any part of displayed data:

1. The names of functions, variables, data structures and other.
2. Change data representation (as numbers, strings, data structures)
3. Easy to understand the disassembled code by providing diagrams and code flow graphs.
4. Automatically name variables and structure in the code.
5. Automatically name functions of the standard library in assembler code [**7**].

1.4 Contributions

While the Distributed Obfuscation Model with multiple levels of obfuscation is robust and more secure than traditional techniques.

The main contribution of this thesis is presenting a suitable solution to protect software, copyright, and distributed software over client/server against reverse engineering analysis, and these contributions are summarized as follows:

1. Developing a comprehensive distributed obfuscation technique contains multi-levels of different obfuscation techniques. And so our model protects the intellectual property and copyright of the software while distributing it over client devices. Using this technique improves the protection and confidentiality of software
2. Another significant contribution is the ability to improve the security of AES and DH algorithms against reverse engineering analysis. This is carried out by using random key generation ideas based on key exchange and generation processes that utilize the Diffie Hellman principle.
3. Develop multi-level obfuscation techniques for distributed software, combined with AES encryption algorithms to make robust models and to increase security, thus hiding sensitive, valuable and secret data from attackers. Also, in a de-obfuscated model, the client requests a key from the server to download and open the obfuscated software from the internet. The key consists of subkeys, each subkey used to de-obfuscate only one level, each level de-obfuscated using a different subkey from another. Using subkeys makes the reverse engineering analysis more complicated. This technique is called “DOSP-AES”
4. Introduce another technique called "DOSP-RSA". The same as DOSP-AES but an RSA encryption algorithm is used.

1.5 Thesis Organization

This thesis has six chapters summarized as follows:

Chapter One: Introduction. This chapter describes the thesis idea, objectives, contributions, motivations, threat model, problem statement, research aims, and structure of research.

Chapter Two: Literature review and Background. This chapter presents literature review, software development tools, Cryptosystem models, and Attack analysis.

Chapter Three: Model of obfuscation. This chapter presents obfuscation, the obfuscation technique, the reverse engineering process, and the tool for reverse engineering.

Chapter Four: Research Methodology and procedure (Proposed model). This chapter talks about the obfuscation model this thesis uses. describes all levels that the DOSP-AES model contains. Also, describe the model DOSP-RSA, and describe all levels that the DOSP-RSA model contains.

Chapter Five: Results and analysis. Includes the testing results of the proposed model (DOSP-AES), testing, and results of the DOSP-RSA model. In addition, it presents a comparison between the DOSP-AES model and the DOSP-RSA model.

Chapter Six: Conclusions and future work. Contains the research conclusion followed by future work and recommendations.

Chapter Two

Literature Review and Background

Chapter Two: Literature Review and Background

This chapter provides a brief overview of the development environment tools, such as compiler and interpreter, for programming languages C++, Java, and Android. Then, it presents detailed cryptosystem models and attaches an analysis.

2.1 Literature Review

The study reviews the recent literature on obfuscation techniques used in mobile and software malware.

1. Hardening Code Obfuscation Against Automated Attacks

This research introduces LOKI which is a code obfuscation approach against all known automated deobfuscation attacks and unaccounted attack vectors (ex. program synthesis).

They introduce a set of obfuscation techniques that can be combined to protect code against reverse engineering analysis and deobfuscation attacks. The design of this model is based on the following principles:

1. Merging core semantics
2. Merging different, independent core semantics which increases the complexity
3. Diversifying the selection mechanism and adding syntactic and semantic complexity

For more details regarding these principles, check reference.

The research shows that it can protect code from reverse engineering and deobfuscation attacks and

ensure intellectual property rights. Through code obfuscation [8].

2. Obfuscation for Privacy-Preserving Syntactic Parsing

Homomorphic encryption aims to convert data such that another party can work on it without vividly revealing the content of the original data. This research proposes an idea for a privacy-preserving transformation of natural language data, based on homomorphic encryption. Their basic tool is obfuscation, depending on the characteristics of natural language. Mainly, a given English text is obfuscated using a neural model aiming to preserve the syntactic relationships of the primary sentence. This is the obfuscated sentence that can be interpreted instead of the initial one.

The pattern operates at the word level and retains to obfuscate each word independently by converting it into a new word that has a similar syntactic task. The text obfuscated by this model causes better function on three syntactic analyses (two dependencies and one constituency parser) in contrast to an upper-bound random replacement beginning.

As more terms are obfuscated (by their part of speech), the substitution upper bound greatly degrades, while the neural pattern retains remarkably high performance. All of this is done without much renounce of privacy compared to the random substitution upper bound. They analyzed the outcomes and detected that the substituted words have similar syntactic characteristics, but different semantic content, compared to the initial words [9].

3. Dynamic Analysis Techniques to Reverse Engineer Mobile Applications.

Mobile applications are becoming more significant with time. It is for this reason that many companies recommend creating more applications. As mobile applications get more complex than before, the problems of code maintenance and comprehension of poorly documented apps are arising. To solve this

problem techniques to reverse engineer mobile applications based on code are needed. In this study, they created a smartphone version for reverse engineering applications. Which is a complete set of techniques and tools analyzing the functional structure of the application, to improve the software understanding of it, thus its maintenance. For this purpose, they developed a functional structure view of the system that includes two components: first, the classes, their relationships, and the methods that implement a business-relevant scenario; second, the class's time series that allows them to recognize when a specific class or set of classes are involved in a scenario technique. Since execution trace files can contain very large amounts of events, they developed a trace segmentation technique to cope with the large volume of data. This trace segmentation proved to be efficient in analyzing the interactions between the components of the system. By applying this method to the analysis of iPhone apps. They expanded the interpretation of the results of the tools and methods used in the smartphone context.

The results show the creation of a trace file that applies to all mobile environments. Knowing how to build source code for a specific programming language in order to create a trace file is also necessary. Specifically, they have created an instrument for Java, to be able to analyze Android applications. This developed trace analyzer provides the maintenance engineer with a wide range of views to analyze how the code runs. The time series technique is useful in visualizing the mutual behavior of the classes in a suitable arrangement. And it gives an idea about the interaction of classes while running the use-case. There were some obstacles during this study since they used these techniques which are based on business-related use-cases. This enabled them to make a hypothesis about the business semantics of the observed patterns of execution of classes. However, as the use-cases are taken from the user´s observation, this analysis is not complete. Due to the complexity of the task, they studied a semi-automated technique to recover the use case directly from the code for desktop applications. In addition, use-case recovery from source code is still a problem that is not solved [10].

4. **A Parameterized Flattening Control Flow Based Obfuscation Algorithm with Opaque Predicate for Reduplicate Obfuscation.**

This study suggests a repetitive code obfuscation algorithm to protect the source code and enhance white box security. They employed the parameter decomposition tree to shape the code, then they used a flattening control flow system to decompose the source code into a multi-branch WHILE-SWITCH structure. Finally, opaque predicate code depiction and diverse ways of adding opaque predicates into program branches and sequence blocks were applied. Furthermore, the time-space cost of source code and obfuscated code was compared. The results showed that the suggested algorithm made the code´s anti-attack ability better and also elevated the difficulty of reverse engineering [11].

5. **An Adaptive Approach to Recommending Obfuscation Rules for Java Bytecode Obfuscators**.

Many techniques make the decompiled program uncompilable. The obfuscation effects cannot be easily undone by other cracking tools. A cracker has to spend lots of time debugging the decompiled buggy program manually. The main objective of the obfuscation techniques proposed is to scramble the symbolic names and the symbolic references in the bytecode. This paper sheds light on the OR Chooser (Obfuscation Rule Chooser), as an adaptive approach to endorse a small number of obfuscation rules for java bytecode obfuscation. To randomly choose\unchosen obfuscation rules for the obfuscation and to compute the obfuscation distance between the bytecode before and after obfuscation. Moreover, OR Chooser considers a constant process to adaptively obfuscate the bytecode file f, as the obfuscated code is far from f. They applied and observed OR Chooser on bytecode obfuscators: Android R8.

The results of the observation exhibit the strength of ORChooser. In particular, with 5 repetitions, OR Chooser selects about 25% of obfuscation rules for R8, minimizing more than 29% of the bytecode size. The likeness between the bytecode files before and after obfuscation is less than 27%, implying that

ORChooser – supported obfuscators have obfuscated bytecode competently and minimized the comprehensibility greatly [12].

6. A Study of Encryption Algorithms AES, DES and RSA for Security

Using AES, DES, and RSA encryption algorithms, this research compared their use as encryption tools in terms of their prompt time for encryption and decryption. Using distinct file sizes, they encrypt and decrypt the files based on AES, RSA, and DES algorithms. Simulation outcomes suggest that AES performs better than DES and RSA. The AES algorithm utilizes the least encryption and RSA utilizes the longest encryption time. The decryption of the AES algorithm is more effective than other algorithms[13].

7. Hybrid obfuscation using signals and encryption

In this paper, they proposed an algorithm that aimed to implement the obfuscation in both methods the signal and the encryption to enhance the complexity level and inhibit the detection potency as possible. On one hand, an advantage of this algorithm is its structure that makes the control graph of the program vague. On the other hand, the disadvantage is the high costs. It's a combination of both signal and encryption methods that is why it’s called S&E.

This proposed algorithm S&E applied a signal method in order to convert the tree- and graph-like structure of the program into a star structure and conceal the control flow graph of the problem. The problem of the signal method is the high number of call and return instructions. However, this study proposed inserting a dispatcher to the program in order to change the signal program to the original control flow graph. And so, the problem of the signal method was solved. The dispatcher that is used was encoded to keep it secure from hackers. In addition, a new approach has been proposed to calculate the complexity and resilience. There are five functions that were suggested to be able to calculate the values

of complexity and resilience. The results of the difference of obfuscation data similarities with the initial codes, based on Mishra's method, represent a functioning advantage of the suggested and hybrid algorithm obfuscation [14].

2.2 Software Development Tools

Software developers deploy software development techniques to produce, debug, keep, and support other programs. Development tools are utilized during the actual coding process. This entails original code production. The most essential tools are a source code editor and a compiler or interpreter. This section aims to concentrate on the compiler and interpreter tools

2.2.1 Compiler

Before the computer processes the program, The source code of the program must be converted into a suitable instruction sequence. It is necessary to convert the source code of the program into an instruction sequence before it is processed by the computer. Compilers convert source code (a set of rules, symbols, and special words) written in any high-level language into binary instructions (machine code) that can be used by the computer processor. Its main function is to translate the source code of the program into machine-understandable code and preserve the semantics of the program source code [15] [16] [17] [18].

Many compilers are a combination of five different tools, Editor, debugger, compile, linker, loader. But in this thesis, our scope is limited to debugging and compiling only.

The **editor** for writing a software program or modifying an existing program is provided by IDE (Integrated Development Environment) which is a software application that combines all of the features and tools (graphical user interface and developer tools) needed by a software developer like a Visual Studio program. IDE is created when the compiler is activated. When editing is completed, if the source code of the program does not contain any syntax error, the compiler **compiles** the program successfully

and translates it into object code (.obj). **Object code** is machine language code but it's not executable. On the other hand, if the source code contains any syntax error, the debugger tool is activated and produces a list of syntax errors. Correcting source code and eliminating the syntax errors in the program can be done only manually through re-editing the program source code. After the program compiles successfully and the object code is obtained. The **liker** tool links the object code with the library functions that are used in the program and also it combines all the modules as required. So, when the linker process is completed, the object code is converted into executable code (.exe). Finally, the **loader** tool loads this exe file in the main memory for execution [17]. All compiler functions are shown by the flow chart in Figure 2.1.

The compiler operates in six phases as shown in Figure 2.2. Each phase converts the source program from one representation to another. **Symbol table** and **error handler** interact with these phases [15] [17].

1. **Lexical Analysis**, it interprets the program and changes it into tokens. Tokens in programming languages include keywords, identifiers, operator symbols such as +, -, /, =, punctuation symbols such as (,), {,}, parentheses, or commas.
2. **Syntax Analyzer**, sometimes it is called a parser. It builds the parse tree from the token created. Also, it checks if the expression made by tokens follows the syntax or not.
3. **Semantic Analyzer**, it proves the parse tree, whether it's meaningful or not.
4. **Intermediate Code Generator**, compilers create intermediate code from source code.
5. **Code Optimizer**, it changes the intermediate code so that it utilizes fewer resources, less power, generates more speed, and makes better target code.

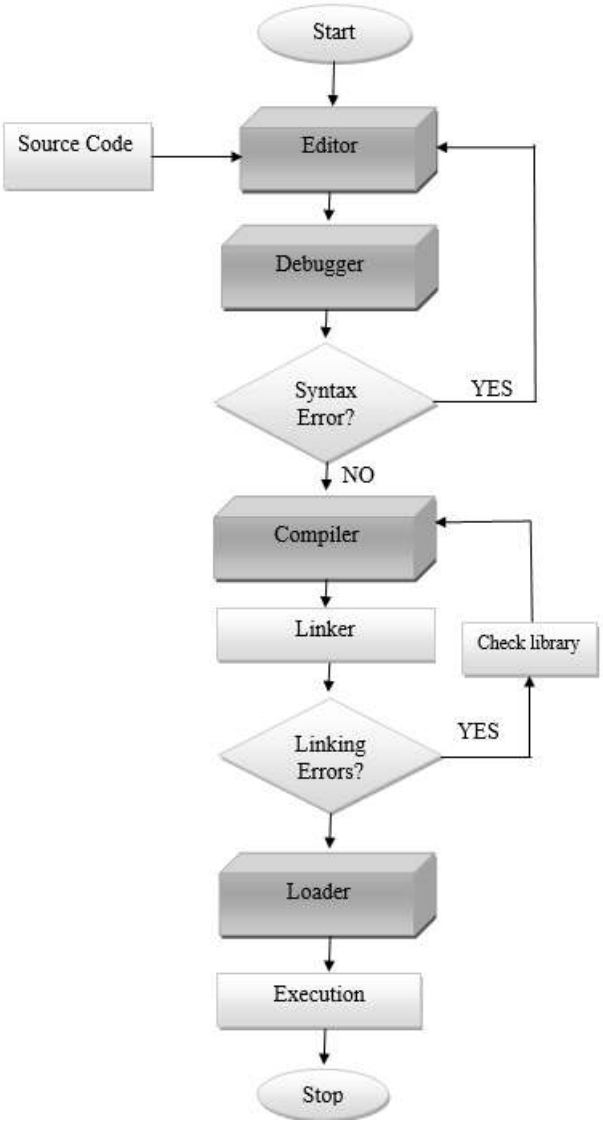


Figure 2.1: Life cycle of a computer program.

6. **Symbol Table**: It is a data structure produced by the compiler. This allows the compiler to find identifiers quickly. It entails all the identifier names, their types, the scopes, functions, and methods names.

❖ **Error Handler**: There are many types of errors some of them, first compiler runtime errors, occur when the compiler detects syntax error in the source code of the program, logic errors, occur if there

is strange output, crash in the program and if the program operates in an unexpected and incorrectly way, finally run time error, it occurs because of invalid input data. In all cases, the compiler issues an error message about the error that occurred [17] [18].

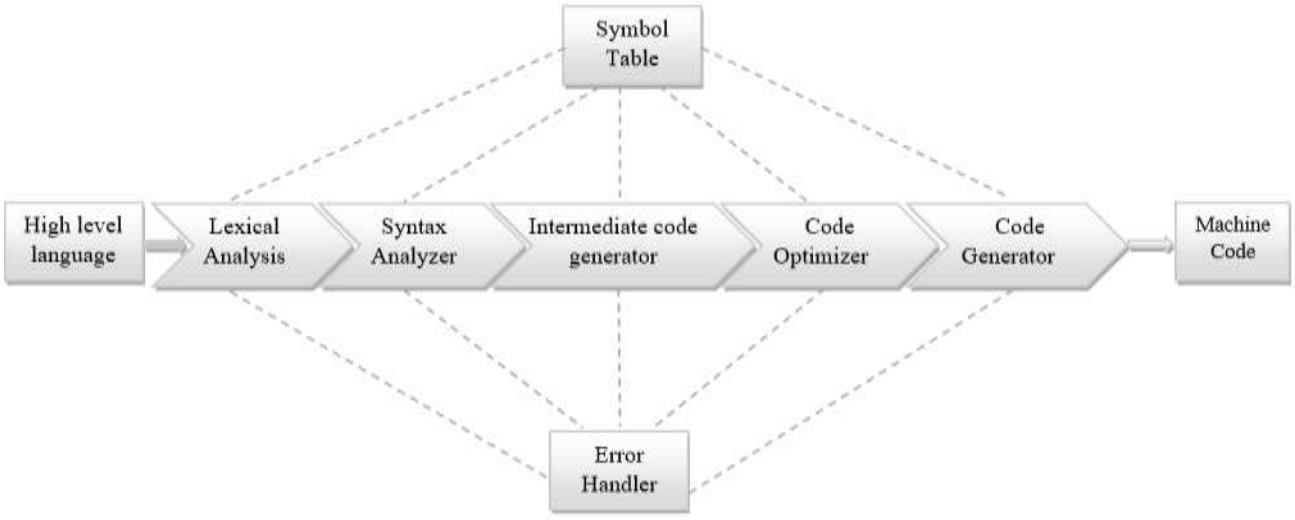


Figure 2.2: Phases of compiler.

2.2.2 Interpreter

The interpreter is a computer program that translates source code directly from a high-level language into some other form, one statement at a time, without translating it into machine language first, so no intermediate code is generated [19].

2.2.3 Compiler Vs Interpreter

There is a difference between compiler and interpreter. A compiler scans the entire program and translates it into machine language code, while an interpreter takes one line of source code at a time and executes it if there is no syntax error in that line. So, interpreters execute programs line by line and the execution time is slower, So, the execution time of the compiler is faster and much more effective than

interpreters. No intermediate code is generated (memory efficient) by the interpreter but the compiler requires more memory because it generates intermediate code. In the early days of computers, the interpreters were in much practice because of low speed of hardware and less memory. At a very low cost a faster processor and a large amount of memory are available [17] [18] [19].

2.3 Cryptosystem Models

This section presents an overview about Cryptography models. Cryptography is derived from the Greek word 'crypto' means secret, 'graphy' means writing. It is the science and study of the techniques that protect the data from the access of unauthorized users by hiding the content of the message, it ensuring integrity, availability, identification, confidentiality, authentication of user, security and privacy of the data. As shown in Figure 2.3, encryption algorithms encrypt the plaintext into cipher text using a key. A decryption algorithm is used for decryption and restoring the original plaintext from the ciphertext [20] [21].

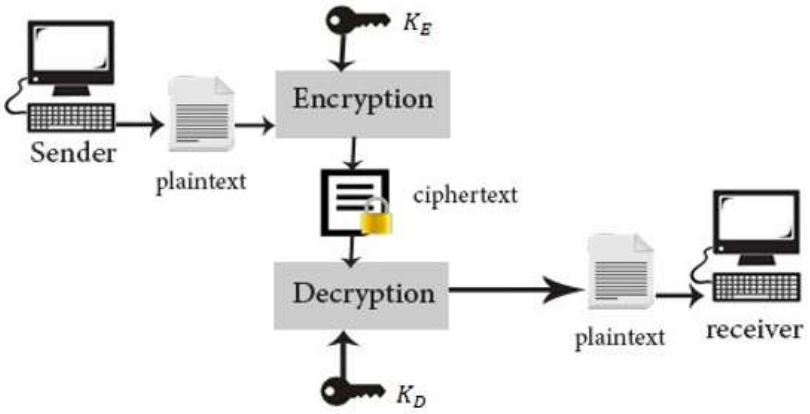


Figure 2.3: Cryptography

Cryptology entails two main sorts. Cryptography which is the study of securing the information. And Cryptanalysis which is a branch of analyzing secure communication. Cryptography is utilized for

different reasons that can be either all accomplished at the same time in one approach, or only one of them These reasons are [20] [22] [23]:

1. **Confidentiality** is a service to maintain the information secret and make sure that no one can understand the accepted information/message except the one who has the decipher key.
2. **Authentication** helps in recognition. that means the user or the system can verify their identities to different parties who don't possess personal information of their recognition
3. **Data Integrity** helps to label the uncertified change of data and it makes sure that the accepted data has not been converted in any way from its initial form.
4. **Non-Repudiation**: is deployed to verify that the sender really sent specific information, and the information was accepted by the specified party, so the recipient cannot say that the message was not sent.
5. **Access Control**: it is preventing an uncertified utilization of means. This purpose is to control who can have access to the resources, if one can access, under which restrictions and terms the access can be carried out, and what are the permission stages of a given access.

2.4 Cryptographic Techniques

Cryptosystems have two types based on the number of keys used, they are either symmetric (private key encryption), in which case both the encryption and decryption keys must be kept secret, or asymmetric (public-key encryption) in which case one of the keys can be made public without compromising the other [20] [21] [23].

2.4.1 Asymmetric Encryption (public-key encryption)

Each person has a pair of keys (a public key and a private key). A person's public key is published but the private key is kept secret and hidden. data is converted using the recipient's public key and can only

be decrypted using his private key. In this method there is no need to share secret keys between sender and receiver. All communications utilize only public keys, and no private key is transmitted [24] [25], see Figure 2.4.

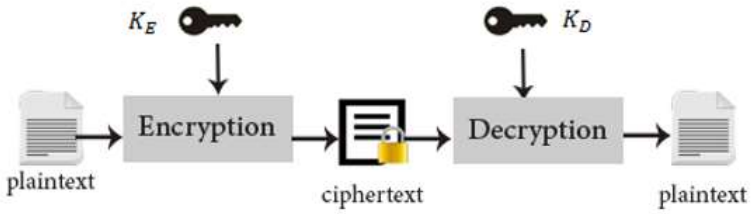


Figure 2.4: A simplified model for asymmetric encryption.

2.4.1.1 Rivest–Shamir–Adleman Algorithm (RSA)

RSA algorithm introduced by **R**ivest, **S**hamir, and **A**dleman in 1978. The important things of RSA, it implements asymmetric key /public-key cryptosystem for key exchange or as digital signatures [25] [26] [27].

1. **Public-key encryption**

   In RSA, encryption keys are public, and the decryption keys are private. Only the person who has the decryption key can decrypt the ciphertext.

2. **Digital signatures**

   The receiver checks and verifies that the transmit message was created by the sender (signature). This works using the sender's decryption key.

RSA used two keys, public key and private key. Public key can anyone know it and it is used to encrypt the ciphertext, but the ciphertext only decrypts using the private key. The RSA algorithm is characterized by a high degree of security, difficulty of getting original text from ciphertext, due to the difficulty of factoring large products of two prime numbers [25] [26] [27].

RSA operation involves four steps: key **generation**, **key distribution**, **encryption** and **decryption** [26] [28].

❖ **Key generation**

key generation can be done using the following steps:

a. Select two large prime numbers, $\boldsymbol{p}$ and $\boldsymbol{q}$.

b. Compute n, as equations 2.1:

$$n = p \times q \tag{2.1}$$

c. Compute phi function (Euler's totient function)**,** as equation 2.2:

$$\phi(n) = (p-1)(q-1) \tag{2.2}$$

d. Select the public exponent e, such that $1 < e < \phi(n)$, and e must be coprime with $\phi(n)$, as equation 2.3:

$$gcd\ gcd\ (\ e, \phi(n)\ ) = 1 \tag{2.3}$$

e. Compute the private key d, as equation 2.4:

$$d = e^{-1}\ (mod\ \phi(n)) \tag{2.4}$$

f. The public key can be computed using n and e, as equation 2.5:

$$k_{public} = Public\ key\ (n\ , e) \tag{2.5}$$

❖ **Key distribution**

If sender and receiver use the RSA algorithm to transmit a message, Sender must know the receiver's public key to encrypt the message, and the receiver must use his private key to decrypt the message.

The receiver must send his public key (n, e) to the sender, to enable the sender to send his encrypted message. Sender transmits his public key (n, e) to the sender via a reliable. receiver's private key (*d*) is never distributed [26] [27].

❖ **Encryption**

RSA Encryption process is done using the public key k_{public} and plaintext m, as equation 2.6:

$$c = m^e \, mod \, n \tag{2.6}$$

❖ **Decryption**

RSA decryption process is done using receiver's private key d, and ciphertext c, as equation 2.7:

$$m = c^d \, mod \, n \tag{2.7}$$

2.4.1.2 Diffie –Hellman Algorithm (DH)

The Diffie–Hellman algorithm is used to exchange cryptography keys. It is a key exchange method that allows two parties that do not have any knowledge of each other to create a shared secret key over a communications channel which is insecure. The following steps show how the Diffie-Hellman Algorithm Key exchanges work [29] [21]:

1. Compute global public elements: in this step choose $\boldsymbol{q}$ as prime number and $\boldsymbol{\alpha}$ is **primitive** root of $\boldsymbol{q}$, such that $\alpha < q$**.**
2. Key generation of User **A:**

   This can be done by selecting private key X_A, such that $X_A < \boldsymbol{q}$, then calculate public key Y_A, as equation 2.8:

$$Y_A = \alpha \times X_A \, mode \, q \tag{2.8}$$

3. Key generation of User **B**:

   This can be done by selecting private key X_B, such that $X_B < \boldsymbol{q}$, then calculate public key Y_B, as equation 2.9:

$$Y_B = \alpha \times X_B \, mode \, q \tag{2.9}$$

4. Calculation of secret key by user A: secret key of user **A** can be computed as equation (2.10):

$$K = (Y_B)^{X_A} \, mod \, q \tag{2.10}$$

5. Calculation of secret key by user B: secret key of user **B** can be computed as equation 2.11:

$$K = (Y_A)^{X_A} \, mod \, q \tag{2.11}$$

Notice that, the result is that the two sides have exchanged a secret key value**.**

2.4.1.3 Elliptic Curve Cryptography Algorithm (ECC)

ECC is an alternative mechanism for implementing public-key cryptography. ECC is based on discrete algorithms that are much more difficult to challenge at equivalent key lengths. The security of a public key system using elliptic curves is based on the difficulty of computing discrete algorithms in the group of points on an elliptic curve defined over a finite field. Elliptic curve equation over a finite field F_p can be described by equation 2.12:

$$\boldsymbol{y^2 = x^3 + ax + b} \tag{2.12}$$

Here, y, x, a and b are all within F_p, and p is an integer modulo p. a and b is the coefficients which determine what points will be on the curve. Curve coefficients have to fulfill one condition that is:

$$4a^3 + 27b^2 \neq 0$$

This condition guarantees that the curve will not contain any singularities.

Each value of a and b gives a different elliptic curve. The public key is a point on the curve and the private key is a random number. The public key is obtained by multiplying the private key with a generator point G in the curve [30] [31].

2.4.2 Symmetric Encryption (Private Key Encryption)

Using the same secret key in encrypt and decrypt messages. The problem in this type is transmitting the secret key to the person that needs it [24]. Figure 2.5: A simplified model for symmetric encryption.

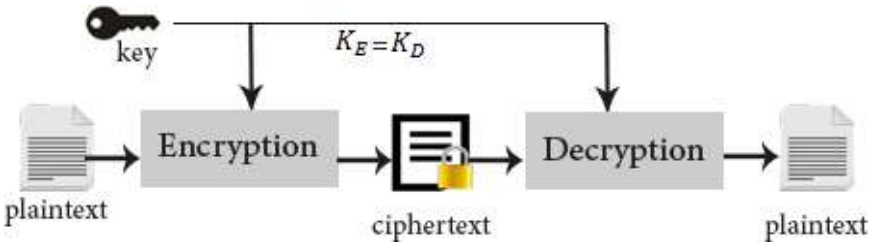


Figure 2.5: A simplified model for symmetric encryption.

2.4.2.1 Advanced Encryption Standard (AES)

AES is based on a substitution permutation network. It is fast in both hardware and software. AES has an affixed block size of 128 bit, and it can specify with key sizes in any multiple of 32 bits. The key size has a maximum of 256 bit. AES algorithms have many characteristics, resistance against all known attacks, speed and design simplicity.

AES operates on a 4 × 4 matrix of bytes, termed a state. The AES cipher is specified as the number of repetitions of transformation rounds that convert the plaintext into ciphertext. Each round consists of several processing steps, including one that depends on the encryption key. A set of reverse rounds are applied to transform ciphertext back into original plaintext using the same encryption key. The encryption algorithm is organized into three rounds. Round 0 is simply an add key round; round 1 is a full round of

four functions; and round 2 contains only 3 functions. Each round includes the add key function, which makes use of 16 bits of key. The initial 16-bit key is expanded to 48 bits, so that each round uses a distinct 16-bit round key. Figure 2.6 represents the AES encryption process, and relationship between number of round and cipher key sizes, where 10 cycles need a key length of 128 bits, 128 cycles support a key length of 192 bit and 14 cycles need a key length of 256 bit [21] [32].

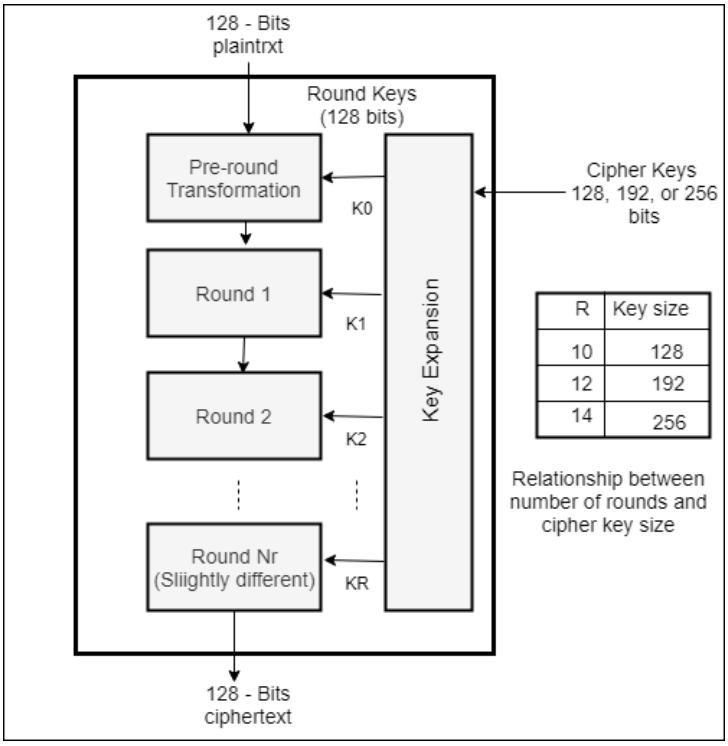


Figure 2.6: AES general encryption process

The encryption algorithm involves the use of four different functions, or transformations: add key, nibble substitution, shift row, and mix column, which are shown on Figure 2.7.

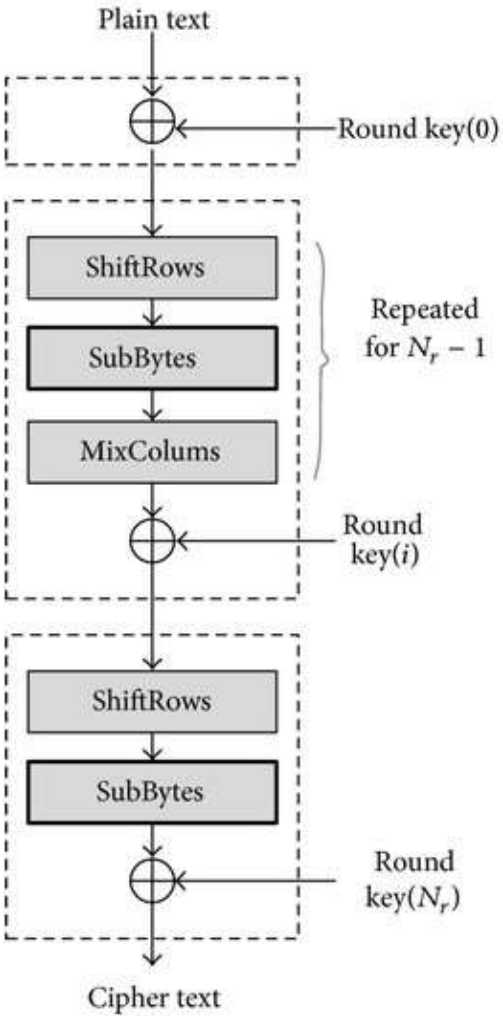


Figure 2.7: AES structure encryption process

2.4.2.2 RC4 Algorithm

RC4 is a simple cryptography XOR process for both encryption and decryption. The main issue is in the key generation which has several stages as follow:

The RC4 has an S-box which contains permutation numbers [0 – 255] as S[0], S[1], … S[255] where this permutation is a function of Key (K), with an effective length. To generate the initial S-box:

First stage: vector initialization **S**:

$$S[i] = i,$$

$$T[i] = K\ [i \bmod key\_length] \tag{2.13}$$

where **i** is an index pointer from 0 to 255 and **T** is a temporary vector.

Second stage: Generate the permutation vector S using the key-scheduled algorithm KSA:

for many iterations of all i values in S and T from 0 to 255:

$$j = (j + s[i] + T[i]) \bmod 256 \tag{2.14}$$

where j is an index pointer of permuted state vectors with an initial value of j = 0. Then, swap (s[i], s[j]) to implement the permutation S vector byte stream.

Third stage: Generate the key byte stream using Pseudo-random generation algorithm (PRGA):

$$\mathbf{i = (i + 1) \bmod 256,} \qquad i = 0 \text{ as initial value} \tag{2.15}$$

$$\mathbf{j = (j + S[i]) \bmod 256}, \qquad j = 0 \text{ as initial value} \tag{2.16}$$

swap values of S[i] and S[j]

$$\mathbf{K = S[(S[i] + S[j]) \bmod 256]} \quad (2.17)$$

K is a byte of the key byte stream K[0], K[1], …

Final stage: The generated key stream is X-or with the plain text for encryption. X-or is used between the generated key and the cipher text for decryption [33] [34].

2.4.2.3 Triple DES (TDES)

Triple DES or 3 DES is a type of symmetric key cryptography. It is simply extending the key size of DES by applying the algorithm three times in succession with three different keys. The combined key size is thus 168 bits (3 times 56) [35] [36]. It is much more secure than DES. The procedure for decrypting something is the same as the procedure for encryption, except it is executed in reverse. Triple DES is advantageous because it has a significantly sized key length, which is longer than most key lengths affiliated with other encryption modes. There are three keying options in data encryption standards: first, all keys being independent, second, Key 1 and key 2 being independent keys, third, all three keys being identical. The third Key option as shown in Figure 2.8 triples DES [35] [36].

The 3DES encryption process as equation 2.17:

$$\boldsymbol{C = E_{K3}\,(D_{K2}\,(E_{K1}\,(m)))} \quad (2.17)$$

The TDES decryption process as equation 2.18:

$$\boldsymbol{m = Dk1\,(Ek2\,((c)))} \quad (2.18)$$

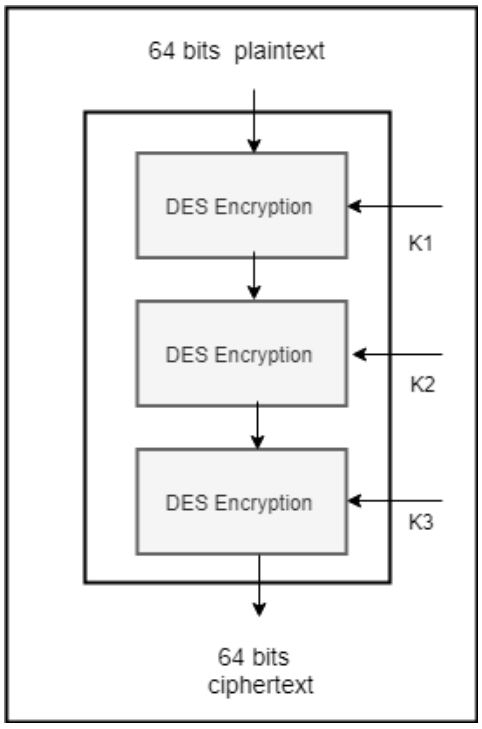


Figure 2.8: Triple DES encryption process [37].

2.5 Performance Comparison of Cryptosystem Model

This section talks about symmetric and asymmetric key cryptography. Two types of cryptography differed from each other through a set of factors as Table 2.1 [38].

Table 2.1: Comparison Between Symmetric and Asymmetric Encryption

Factors	Symmetric encryption			A Symmetric encryption		
	RC4	TDES	AES	RSA	ECC	DH
Key Length	variable	112 to 168 Bits	128,192, or 256 bits	Based on number of bits in $N= p \times q$	135 bits	Key exchange management
Speed	Fast	Fast	Fast	fast	Slow	slow
Security rate	Weak	Adequate	Excellent	Good	Good	Good
Execution time	Slow	Very slow	Faster	slowest	fastest	slow

Each of cryptographic algorithms has weakness points and strength points. We select the cryptographic algorithm based on the demands of the application that will be used. From the comparison shown in Figure 2.9, the AES algorithm is the perfect choice in case of time and memory according to the criteria of prevent attacks, confidentiality and integrity, since it records the shortest time among all algorithms. Also, it consumes the minimum memory storage.

Also, related to the [39], they compared the RSA and Diffie-Hellman key exchange, as there results the Diffie-Hellman more secure than RSA as a key distribution algorithm, see Figure 2.10 they illustrate that Diffie-Hellman needs less time to exchange keys between two parties.

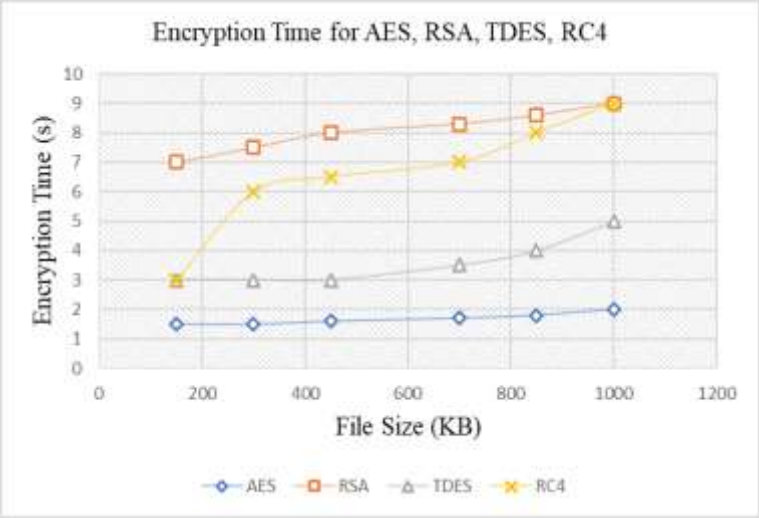


Figure 2.9: Comparison between AES, RSA, TDES, RC4 [62][40][64].

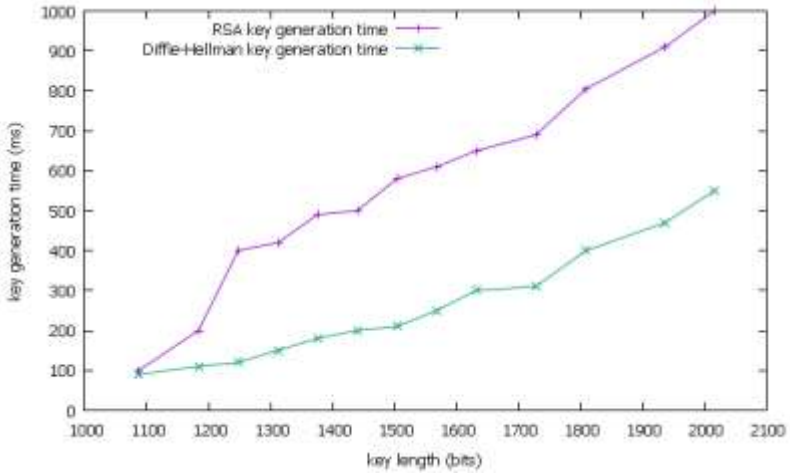


Figure 2.10: Comparison between RSA and Diffie-Hellman Key Generation [39].

2.6 Summary

Based on the AES encryption algorithm; AES is a stronger algorithm against reverse engineering analysis. It provides faster encryption time than another encryption algorithm. AES provides an excellent security rate and better performance than other algorithms. So, depending on researcher results AES algorithm is chosen to be used. The AES algorithm will be used in the encryption process in the DOSP-AES model.

As mentioned in [39], their results confirm that the DH algorithm is more secure and it needs less time to exchange keys between two parties than the RSA algorithm. So, the DH algorithm is chosen to be used for key exchange / distribution in the DOSP-AES model

Chapter 3

Reverse Engineering and Obfuscation Techniques

Chapter 3: Reverse Engineering and Obfuscation Techniques

This chapter discusses reverse engineering, its process and reverse engineering tools. Second part discusses the obfuscation process and its techniques.

3.1 Reverse Engineering

There are many terminologies regarding reverse engineering definition, **Forward Engineering** composed from the normal development process, that is the process of creating a high-level model to form it in a complex form based on specific need, and it's moving from high-level abstractions to the physical implementation of a system. **Reverse engineering**, sometimes called **Back Engineering, is the** process of analyzing a system and drawing out the knowledge about the design, material, structure and surface qualities in order to re-establishing it in another shape based on removed data. Reverse engineering involves the design of new parts, replication of existing parts, destroyed and damaged parts, making the accuracy of the model better and recognition of the digital model. Sometimes, the information is owned by someone who's not willing to share it. Other times, the data has been lost or destroyed. Software Reverse Engineering analyzes the software system to draw out design and implement data and create representations of the system in a different form or at a higher level of abstraction [40] [41] [42] [43].

3.2 Reversing Methodologies

There are different ways for reversing and choosing the right one depending on the intended program, the platform on which it works and on which it was developed, and what

kind of information attackers aim to draw out. There are two primary reversing Methodologies, offline code analysis and live code analysis.

3.2.1 Offline Code Analysis

Means that a binary executable is prepared and uses a disassembler or a decompiler to convert it into a human-readable form. Then reverse engineering analyzes the output of those parts. Offline code required a better understanding of the code than the live analysis because the program data and how data flows can't be seen. So, data type must be guessed.

3.2.2 Live (Online) Code Analysis

The process of converting the code into a human-readable form, then running the code in a debugger and monitoring its behavior on a live system (instead of read converted code). This provides more information because the program's internal data and how it affects the flow of the code can be monitored. Generally, live analysis is better for beginners because it provides a lot more data about the system that can work with [40] [42] [43].

3.3 Reverse Engineering Tools

The tools are an important part of conducting reverse engineering, following some of the reverse engineering tools that the reverse engineering analyzer is used for.

3.3.1 Disassemblers

The disassembler is one of the most important reversing tools. It transfers binary machine code into a readable assembly language text, it translates each instruction and creates a textual representation for it. It aims to make it formatted for human-readability. So, the disassemblers translate the program code back into source code, which humans can understand and analyze.

3.3.2 Debuggers

Debuggers support developers of software with testing, finding and correcting errors in the programs, they also use strong reversing tools as code-tracing phase of program analysis [8][9]. Debugger gives a scattered perspective of the presently working task and lets the user step through the scattered code and look at what the program does at every line. While the code is being stepped through, the debugger mainly depicts the state of the CPU's registers and a memory dump, usually depicting the presently active stack zone [44] [45].

3.3.3 Decompiler

Decompiler generates a high-level language source-code from a program binary (executable file), then the file can assemble eminently. it is never possible to regain the initial code in its exact form because the assembly process always removes some data from the program [40].

3.3.4 System Monitoring

System monitoring is a vital section of the reversing process. Sometimes, the response to user questions can be acquired employing system-monitoring tools and without ever really looking at code. System-monitoring remarked on the different channels of input and output that exist between applications and the operating system. For instance, file monitors tool, monitoring all the system files between program and operating system and displaying every file operation (such as file creation, reading or writing to a file, and so on) made from the application in the system [40].

3.4 Technical Solution to Prevent Reverse Engineering

Many technical solutions have been studied. The most relevant are described as follows.

3.4.1 Code Encryption

Code encryption techniques secure the software against an attacker by transferring the data into another form. Software encrypted using encryption key K_E and decrypted using the decryption key K_D . So, only the user who have the decryption key K_D can decrypt the software and read it [46].

3.4.2 White-Box Cryptography (WBC)

White-box cryptography is a protection software technique that aims to implement a cryptographic algorithm that hide and protect secret keys K_S in the data from attackers (especially Whitebox attackers) who have access to memory, full control of the execution environment, structural algorithm and internal structure of the software. This means that the WBC improves the key extraction security [47] [48].

3.4.3 Obfuscation Technique

Strong technique against reverse engineering analysis. It complicates the code by transforming the code into new difficult code to understand and analyze but with the same characteristic. [4] [49].

3.5 Obfuscation Process

Making things difficult to analyze or understand is called **obfuscation**. Programming code is obfuscated to keep intellectual property or secure data safe, and to prevent an adversary from reverse engineering of the software application. **Deobfuscation** techniques can be utilized to reverse engineer or undo obfuscation [50] [51] [4] [1]. An **obfuscator** is a tool that converts plain source code into an obfuscated code that does the same function and behavior but the obfuscated code is complex to read and understand. The obfuscated version is impossible to follow using the human eye [50] [51] [4].

3.6 Obfuscation Techniques

Obfuscation has many techniques, some of these techniques include the following:

3.6.1 Name Obfuscation

The obfuscation of the source code component is made by changing the program information into another structure with the same characteristics, which makes the code harder to understand, unpredictable, and it affects the presentation of the code. The main goal of it is to make attacking complicated enough against attackers, rather than formally proving the strength of algorithms.

A direct replacement is simply to replace the original name with an unrelated name. The name of the exchange is randomly distributing all the names in the original program among the variables, constants, classes, methods, etc. This method is relatively hidden, and the attacker usually has difficulty detecting it [50] [52]. Removing and renaming is the main method of source code layout obfuscation. Removing means that delete useless debugging information, comments, methods and structures which will not be used in the program.

Renaming includes the transformation of a program variable name, constant name, class name, method name and other identifiers, in order to prevent attackers from understanding the program [1] [50].

3.6.2 Data Transformation

Data obfuscation purpose is to hide data values through changing the statements where variables are defined and utilized. obfuscation is applied to the field of data in order to protect data [50] [53]. One of the techniques of data obfuscation is encryption. Encryption is an obfuscation technique such as in store and transfer data securely, encrypted data can't be analyzed and understood easily [4] [52].

3.6.3 Bytecode Obfuscation

Obfuscation converts clear bytecode into not understood bytecode. The main function of obfuscation is hardening the decompiled program to be understandable, thus attackers will have to give more time and effort on the obfuscated bytecode. Identifiers names of bytecode are replaced with unauthorized obfuscated identifiers applying bytecode obfuscation technique, which produces a syntax error and compilation error when it produces the source code by decompiling [50] [54].

3.6.4 Obfuscation of Assembly Code Instructions

Assembly code or machine code of the software/program can be obfuscated to make the decompiler and debugger more difficult, harder to understand and analysis. To improve and increase the security and piracy of the assembly code, Obfuscation of assembly code

can be applied in different ways, such that using data transformation, using indirect address, combining binary and decimal numbers with assembly code instruction or using the binary and decimal numbers between the assembly code instruction [50] [53].

3.6.5 Anti-Debug

Reverse engineering analyzer using the debug tools to analyze each line in the code. So, the software developer uses the Anti-debug tools to detect if the debugger tool is being used in the software by the attacker and prevent them from analyzing the software [55].

3.7 Characteristic of Success Obfuscation Techniques

- **Complexity:** complexity of the obfuscation code will be increased by using multi levels of obfuscation techniques, the more levels used increase the difficulty and complexity of the obfuscated software.
- **Power and Strength**: Strength of the code determined by how obfuscated code resists deobfuscation attempts. The stronger the code, the more effort, time and resource it takes to analyze.
- **Differentiation**: Differentiation is determining the degree of difference between the obfuscated code and the original code. The higher differentiation means a more complex code.
- **Expense:** The cost-effectiveness of the obfuscation solution is more valuable than one that is pricey especially for large applications [50] [4].

3.8 Summary

Obfuscation is the best solution to prevent Reverse-Engineering analysis, Obfuscation is used in the DOSP-AES model. Three levels of obfuscation are used, name obfuscation, data obfuscation and bytecode obfuscation, as related to the researcher these techniques are powerful and when using multi levels of obfuscation techniques, the complexity, differentiation and strength of the obfuscation code become highest from the original code. So reverse engineering analysis needs more time to de-obfuscated the obfuscated code O(P) especially when using multi levels of obfuscation.

Also, due to the use of the AES – DH algorithms in obfuscation, the strength and strength of AES and DH algorithms was obtained. So, using AES- DH encryption algorithms with multi levels of obfuscation make stronger, robust, and more complex models against reverse engineering analysis.

Chapter Four

Proposed Model Distributed Obfuscation Model for Software Protection (DOSP)

Chapter Four: Proposed Model: Distributed Obfuscation Model for Software Protection (DOSP)

This chapter represents DOSP-AES based on the AES algorithm and DOSP-RSA based on RSA algorithm models which were implemented during this research. Obfuscation and de-obfuscation at all levels of both models were described.

4.1 Introduction to DOSP-AES

The proposed model DOSP-AES is presented in this chapter. DOSP-AES is a novel distributed obfuscation technique for software protection. Obfuscation techniques O(T) transform original code into a revised form that is semantic-equivalent with the original one but more difficult to understand and analyze, to enhance security levels of software against reverse engineering analysis (RE) [1]. We use the Crypto++ library that contains the encryption algorithm, so it's easy to use encryption algorithm (AES) in encryption levels.

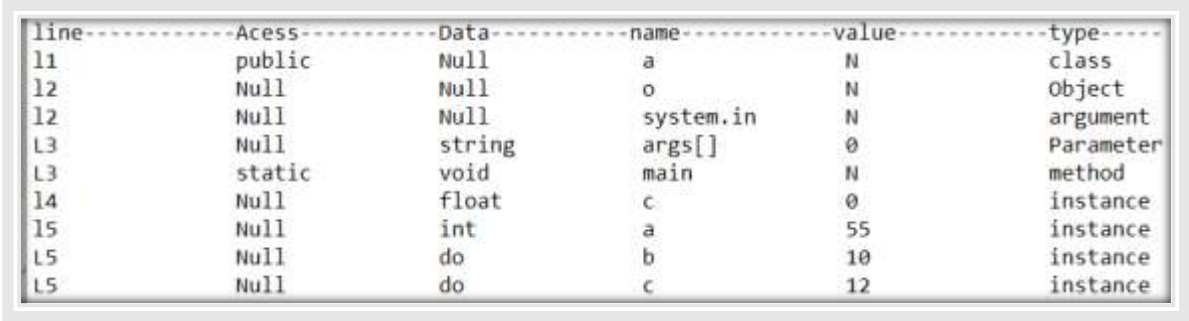
4.1.1 Generate Symbol Table

First of all, the symbol table is generated from the source code that is written in the programming languages C++, Java, and Android. Therefore, the name and value of variables are obfuscated easily.

As mentioned in chapter two, the compiler analyzes the input plain text to produce the output through six phases. Each phase transforms the source program from one representation to another and the symbol table interacts with these phases. The compiler

uses the information about the names in the source code; this information exists and is entered into a symbol table. This information is collected about the string of characters, its type (integer, real, string), its form (variable, structure), and its location. During the lexical and syntactic analysis, the information about the names is entered into the symbol table. Many possible entries symbol tables may contain, name (a string), Attribute, Reserved word, variable name, type name, data type, scope information (where it can be used), storage allocation.

In the model, the symbol table contains Location (the line that the identifier exists on), Scope, data type (integer, double), name, value, type (class, method, instance). The Figure 4.1 show example of the symbol table result.



line	Acess	Data	name	value	type
l1	public	Null	a	N	class
l2	Null	Null	o	N	Object
l2	Null	Null	system.in	N	argument
L3	Null	string	args[]	0	Parameter
L3	static	void	main	N	method
l4	Null	float	c	0	instance
l5	Null	int	a	55	instance
L5	Null	do	b	10	instance
L5	Null	do	c	12	instance

Figure 4.1: Symbol table result

After the Generate symbol table, the 1st and 2nd levels of obfuscation are applied to it, the 3rd level is applied separately. Figure 4.2 illustrates the workflow of DOSP-AES.

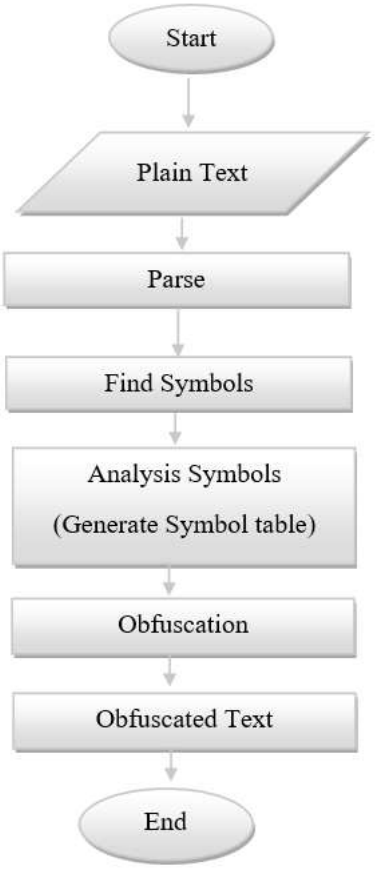


Figure 4.2: workflow of DOSP-AES

DOSP-AES has two inputs **(Plain text, Obfuscation key** K_{Obf}) and one output **(Obfuscation code O(P))** as the result of multi-levels of the obfuscation process that the DOSP-AES technique contains. The plain text represents the source code or bytecode/machine code of the software.

The **Plain Text** is entered into the DOSP-AES model, then all the obfuscation levels of DOSP-AES are applied on the plain text. Plain text obfuscated using Obfuscated key K_{Obf} **,** K_{Obf} generated randomly using a trusted 3rd party key generator (keygen) in Crypto++ simulator. After this process Plain Text becomes Obfuscated text. Obfuscated text

(software) will be available on the internet, so if any user downloads the obfuscated text (software), the user can't use the text (software) until de-obfuscated the text (software) to get the original Plain Text.

When the user downloads the obfuscated text/code **O(P)** from the internet, the user needs to de-obfuscated the text, the user must send request to the server ask for $K_{Obf.}$, After server received the user request, the server send the $K_{Obf.}$, the user can de-obfuscated the text and get the original text (Plain Text). Follow the scenario illustrated in Figure 4.3.

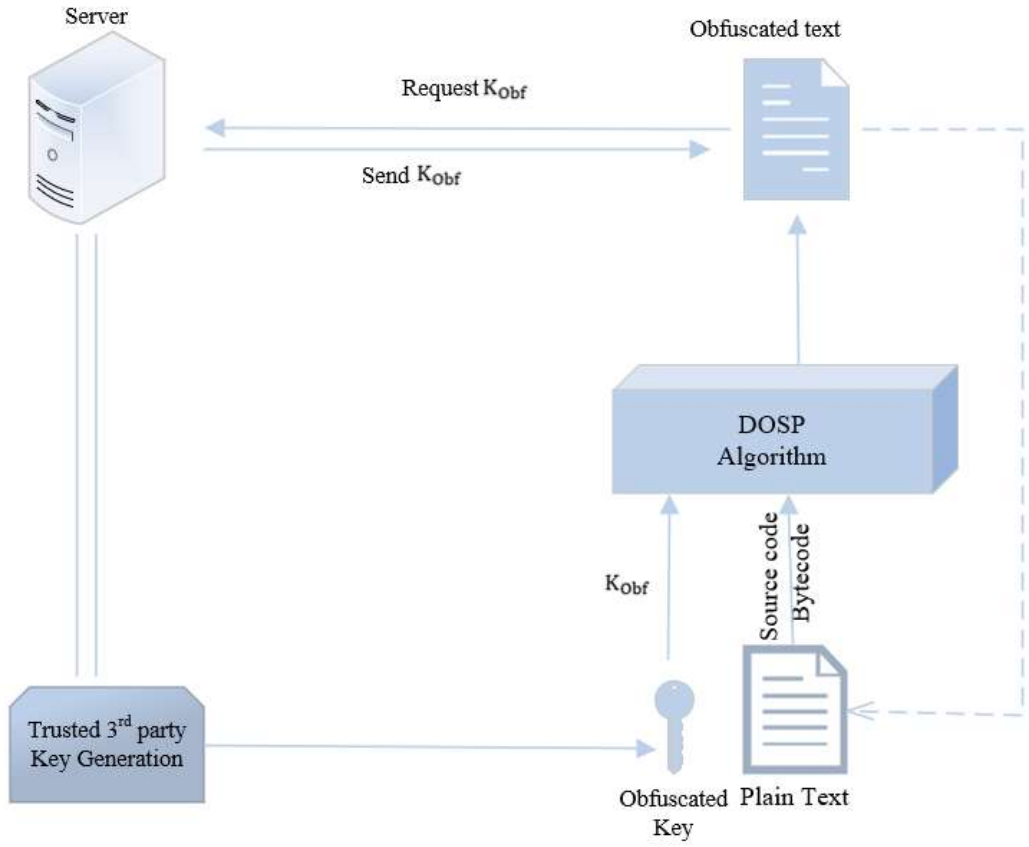


Figure 4.3: DOSP-AES Block Diagram

Information about the obfuscation and de-obfuscation processes of the DOSP-AES model.

Obfuscation and de-obfuscation in the DOSP-AES model are structured by connecting key generation, key exchange, encryption, obfuscation features, and de-obfuscation functions, as shown in Figure 4.4.

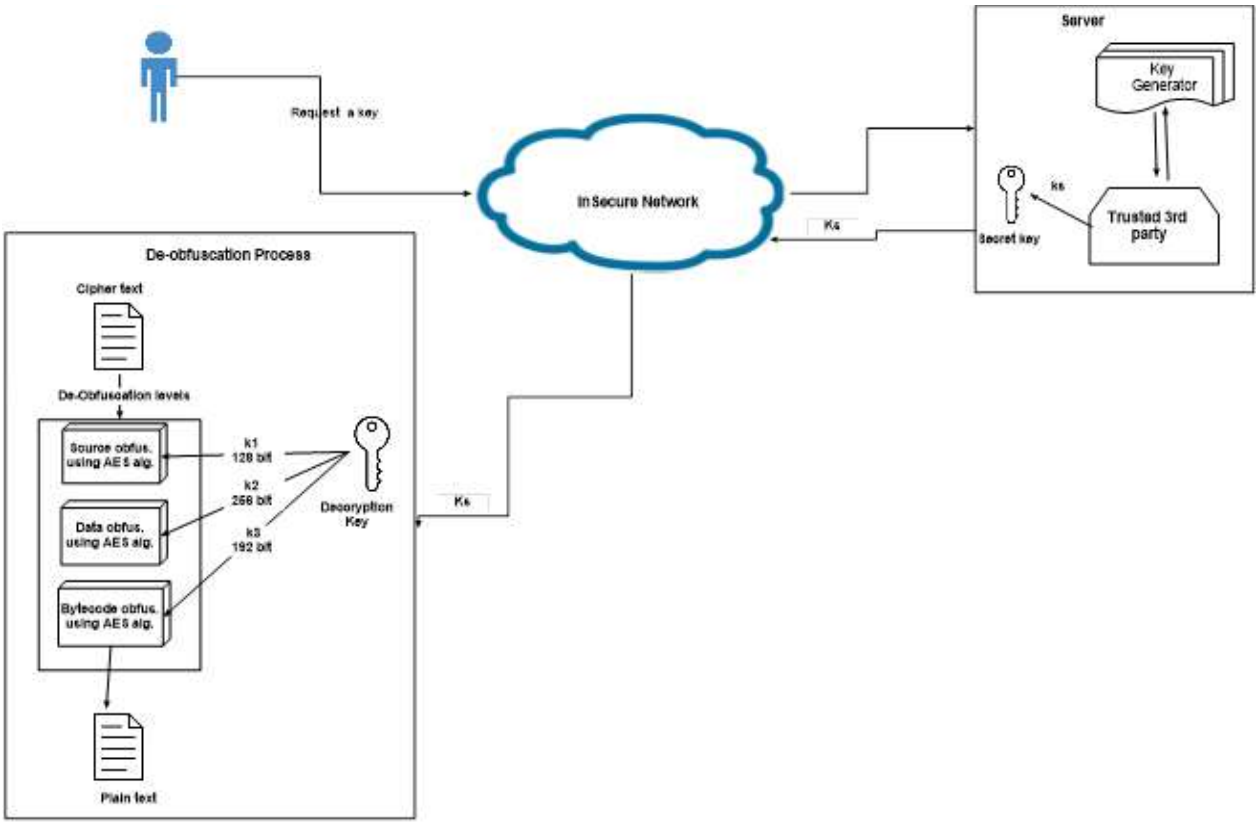


Figure 4.4: General Block diagram of DOSP-AES model (Obfuscation and De-Obfuscation process).

As follows are the obfuscated and de-obfuscated software processes of the proposed techniques.

4.1.2 Obfuscated Process of DOSP-AES

The Figure 4.5 show activity diagram of the obfuscation process of DOSP-AES. After generate symbol table, DOOSP-AES model will apply to the symbol table if the input file is source code, in order to applied 1st and 2nd level of obfuscation. If the input file is bytecode, the 3rd level of obfuscation will apply.

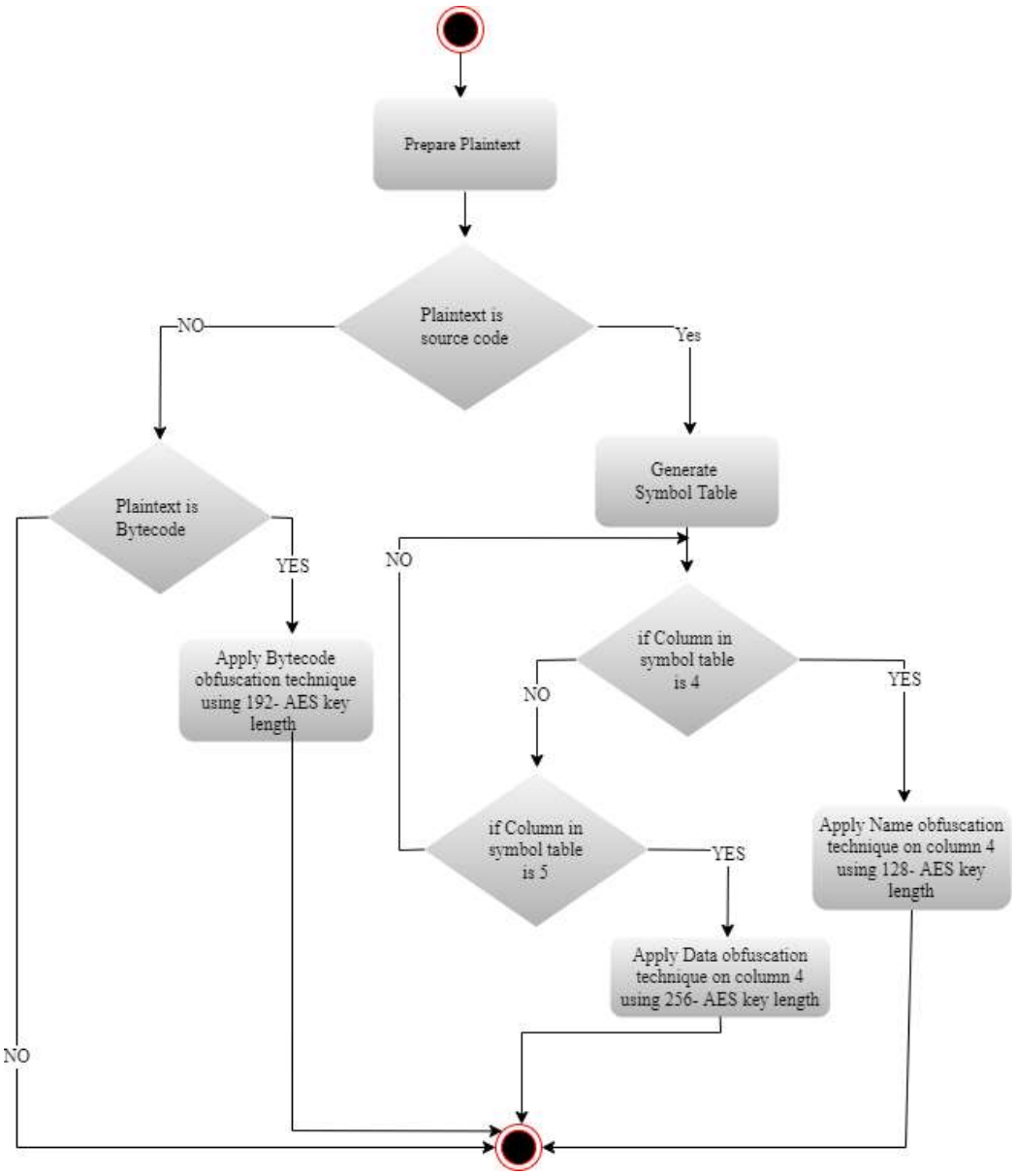


Figure 4.5: Figure: Obfuscation process of DOSP-AES

The three levels of obfuscation techniques, as follows:

1. **First level: Source Code (layout/ Name) Obfuscation**

In this level, the identifiers and variables are given meaningless names by encrypting them with the AES encryption algorithm with a random key length of 128 bits (K128). Code obfuscation involves using **renaming** and **removing** methods. **Removing** means deleting useless debugging information, comments, methods, and structures that will not be used in the program. Deleting not only makes it difficult for an attacker to read, understand, and analyze but also reduces the size of the program. This improves the efficiency of program loading and execution. **Renaming** includes the transformation of a program variable name, constant name, class name, method name, and other identifiers, to prevent attackers from understanding and analyzing the program. Figure 4.6 illustrates an example of the Name obfuscation technique.

```
int foo(int param) {
  int x = 42 + param;
  return x;
}

int main() {
  int y = 2;
  return foo(y);
}
```

```
int i_acbd(int i_eca0) {
  int i_9dd4 = 42 + i_eca0;
  return i_9dd4;
}

int main() {
  int i_4152 = 2;
  return i_acbd(i_4152);
}
```

Figure 4.6: Name obfuscation

2. The Second Level: Data Obfuscation

The data obfuscation technique aims to hide data values by changing the statements where variables are defined and utilized. The main reason for applying obfuscation to a data field is to protect data.

DOSP-AES model technique was used to secure the data. DOSP-AES encrypts the values of constants, local and global program variables to make the reverse engineering process more complex and to secure sensitive data from disclosure. DOSP-AES Encrypt the values of constants, local and global program variables by using an AES encryption algorithm with a key length of 256 bits (K256). The most significant thing is that DOSP-AES obfuscates each variable differently from the other when it is mentioned on more than one site with the same application. Each variable appears in a different form from the other, although it is the same variable. Figure 4.7 illustrates an example of the Data obfuscation technique.

```
int main () {
  int age = 23;
}
```

```
int main () {
  int age = 0x17;
}
```

Figure 4.7: Data obfuscation

3. Third level: Bytecode Obfuscation (BO)

BO modifies bytecode so that the decompiled program of the obfuscated bytecode contains obscure compilation errors while the obfuscated bytecode still functions correctly. A Java program is compiled to bytecode. The names of types, fields, and methods are stored within a bytecode file. These names and the simple machine instructions make de-complication of the bytecode file easy by decompiling.

Obfuscation tools are one of the major defenses against decompiles. Obfuscation transforms clear bytecode into more obscure bytecode.

The goal of BO is to make the decompiled program much harder to understand so an attacker has to spend more time and effort on the obfuscated bytecode. The bytecode identifiers are obfuscated by using the bytecode obfuscation technique, which results in a syntax error and compilation error when decompiling.

DOSP-AES obfuscates the identifiers and class names in bytecode files, by employing the AES encryption algorithm with key length 192 bits K_{192}. This level is applied to the machine code when running C++ programs and to the .DEX File when using Android programs.

4.1.3 De- Obfuscated process (DO) of DOSP-AES

The De-obfuscated Process (DO) is based on a client-server model (distributed system) in which clients can download software and applications that have been obfuscated and uploaded on the Internet. A client requests the server to request the obfuscation key K_{Obf} to de-obfuscate software, then the server responds by satisfying the client's request.

The process of transferring data over the network may contain critical and security information. The transferred information must be secured and protected from unauthorized access, use, modification, or destruction. So, the server will send the secure key (the obfuscated key K_{Obf}) as one block, with a key length of 72 bytes K_{72}. The key will be generated randomly by a trusted 3rd party key generator.

The obfuscated key consists of three subkeys. Each subkey will de-obfuscate different levels of obfuscation levels. So, if the attacker knows one subkey, he can't know the other subkeys and can't disclose the secret data.

Code obfuscation will be de-obfuscated using key length 16 bytes K_{16}, data obfuscation will be de-obfuscated with key length 32 bytes K_{32}, and bytecode level will be de-obfuscated with key lengths 24 bytes K_{24}. The proposed encryption algorithm that will be used in our proposed technique is AES (advanced encryption standard) with DH (Diffie Hellman) data exchange. You can follow the scenario of this process illustrated in the description of the algorithms in Figure 4.8. And the algorithm 4.1.

1. Client download obfuscated file and need the obfuscation key from the server to de-obfuscate the file.

   Ciphertext = Obfuscated (plaintext, key)

2. Server create Socket, port =x, for the incoming request.

   WelcomeSocket = Server Socket ()

3. Server Wait for incoming request

   ConnectionSocket = welcomeSocket.accept().

4. Client create a socket, connect to host id, port = x

   Client Socket = Socket ()

5. Client Send request Using client Socket.
6. Server read request from connection socket.
7. Client read the reply of the server.
8. Server sends a confirmation code to the client and checks the software ID for authentication.
9. Server requests the confirmation code from the client for authentication
10. Client sends the received confirmation code to the server.
11. Server de-obfuscated the software using the obfuscation key

    De-obfuscated (Ciphertext, key) = plaintext

12. The client has the option from the menu to end the connection, if he chooses to end the connection program will terminate, else program will continue
13. Client close client socket
14. Server close ConnectionSocket

Figure 4.8: Description of the Algorithm for de-obfuscated process for DOSP-AES.

```
Ciphertext = Obfuscated (plaintext, key);
WelcomeSocket = Server Socket ();
ConnectionSocket = welcomeSocket.accept();
ClientSocket = Socket ();
socket(SendRequest);
socket(ReadRequest);
send(obfuscation_Key);
Varialbe_key = Received_key.substr(0,16);
Value_key = Received_key.substr(16,24);
ByteCode_key = Received_key.substr(24,32);
If(User_Response == yes){
read_obfuscated_message();
} else {
Terminate() }
De-obfuscated (Ciphertext, key) = plaintext
Client_socket.close();
Connection_socket.close();
```

Algorithm 4.1: Algorithm for de-obfuscated process for DOSP-AES.

Figure 4.9 illustrates the sequence diagram of the de-obfuscated process. The figure shows all steps in detail of the de-obfuscated process.

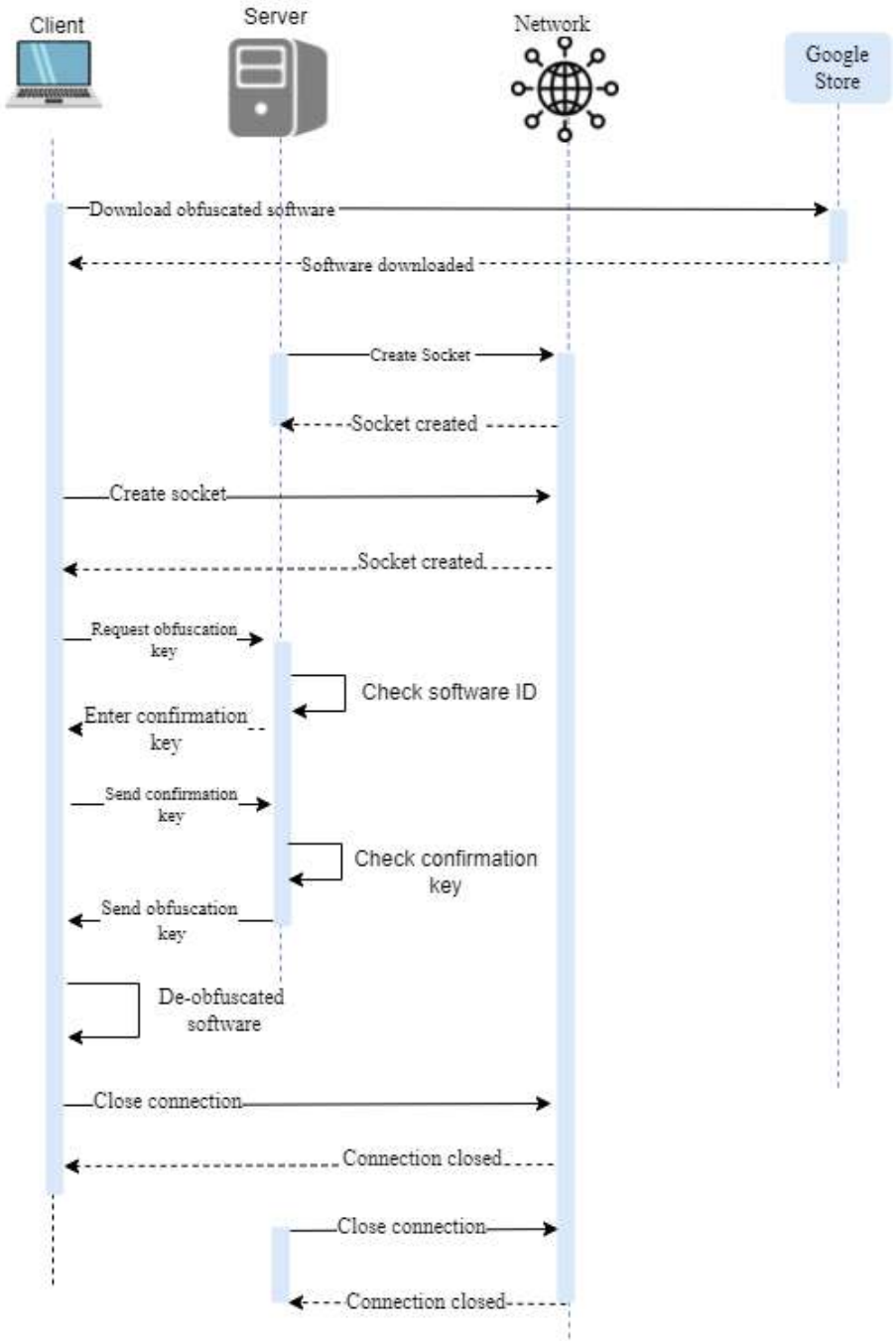


Figure 4.9: Sequence diagram of De-obfuscated process of DOSP-AES

The proposed model focuses on different important points:

1. Solve the problems of distributed systems and transfer data over the network.
2. Make robust software techniques against reverse engineering by using multi-levels of obfuscation techniques.
3. DOSP-AES uses the advantages of AES speed with DH security to produce robust
4. DOSP-AES uses the robust technique of subkeys to protect the key, so if the attacker knows the key, he can't know how the DOSP-AES model uses the key in the obfuscation process; subkeys of the main key are 128,192,256 bits in the AES algorithm.
5. Stored securely obfuscation key, and they must be used only for their intended purpose.
6. DOSP-AES increased the security level of the key by using random numbers (e.g., unpredictable numbers), which are generated on the server-side.

To improve the usability and performance of AES, we compare the model with other previous works.

Related to the [66], in this research, they introduce three levels of obfuscation, source code obfuscation, data obfuscation, and bytecode obfuscation. They use the Permutation Algorithm (PA). DOSP-AES model uses the AES encryption algorithm and its more powerful than Permutation Algorithm (PA).

Also, we introduce another model called DOSP-RSA and use the RSA encryption algorithm instead of using AES as in the first model which we called DOSP-AES.

The AES represents symmetric cryptography techniques that are fast and can be

implemented easily. DH is used for key exchange.

4.2 Introduction to DOSP-RSA

As part of the obfuscated process of the levels, DOSP-RSA used RSA. This section shows how encrypted and de-obfuscated processes work in the DOSP-RSA model. The DOSP-RSA model works the same as the DOSP-AES model but DOSP-RSA uses RSA as the encryption algorithm instead of the AES encryption algorithm at all obfuscation levels.

4.2.1 Obfuscation process of DOSP-RSA

Following sections describes the three levels of DOSP-RSA obfuscation process.

4.2.1.1 First level: Source code (Name, layout) obfuscation

Obfuscation of names involves renaming identifiers and variables with meaningless names. This is done by using the RSA algorithm with a key length of 128 bits K128 to obfuscate all the names in the source code.

4.2.1.2 Second level: Data obfuscation

The DOSP-RSA model uses RSA encryption algorithm to secure the data. DOSP-RSA encrypts the values of constants, local and global program variables to make the reverse engineering process more complex and to protect sensitive data from disclosure. Encrypt the values of constants, local and global program variables, by using the RSA algorithm with a key length of 256 bits K256 to secure the Data. The de-compilation process of reverse engineering becomes more complex.

4.2.1.3 Third level: Bytecode obfuscation

DOSP-RSA modifies bytecode so that the decompiled program of the obfuscated

bytecode contains obscure compilation errors while the obfuscated bytecode still functions correctly. The bytecode identifier names are replaced with an illegal encrypted identifier using the bytecode obfuscation technique, which leads to a syntax error and compilation error when decompiling. Encrypt the identifiers and class names in bytecode files, by using the RSA algorithm with key length 192 bits K192 to obfuscate the bytecode.

4.2.2 De-obfuscation process of DOSP-RSA

The de-obfuscated process uses a client-server model (distributed system), where clients download applications and software from encrypted websites. A client requests the server to request the encryption key to de-obfuscate software, then the server responds by satisfying the client's request. So, the server then sends the obfuscated key KObf as one block, key length is 72 bytes K72, the key will be generated randomly.

De-obfuscating code obfuscation with a key length of 16 bytes, de-obfuscating data obfuscation with 32 bytes, and de-obfuscating bytecode level with 24 bytes. Furthermore, the proposed encryption algorithm that will be used in our proposed technique is the RSA algorithm.

4.3 Summary

This chapter presents in detail the obfuscated process and the three levels of obfuscation of the DOSP-AES model (Name, Data, and bytecode). And the de-obfuscation of the DOSP-AES model at all three levels. Also, represent the DOSP-RSA model, the encrypted and deobfuscated process of DOSP-RSA with all levels of obfuscation (Name, Data, and bytecode).

Chapter Five

Results and Analysis

Chapter Five: Results and Analysis

The results of the DOSP-AES model are presented in this chapter. In all experiments, the following specifications are used: Intel® CoreTM i7-165G7 CPU @ 1.30GHz 1.50GHz, 16 GB RAM/ Microsoft Windows 10 Pro/Visual Studio 2019.

A symbol table is constructed from different file sizes (KBs) in order to apply the different data and name obfuscation levels of the DOSP-AES model, using different key sizes at each level. As a first step, Name obfuscation has been implemented using 128-bit keys. Second, the Data obfuscation is implemented using 256-bit key sizes. The third level uses the 192-bit key length to obfuscate bytecode. These levels in C++, Java and Android software are taken into account when calculating the time. Both during obfuscation and deobfuscation. In order for a user to deobfuscate each file with its own key, the files need to be uploaded to the Internet in an orderly manner and the keys must be sent in the same order as the uploaded files.

All three levels of obfuscation can be accomplished with either the AES encryption algorithm, represented by the first technique (DOSP-AES), or the RSA encryption algorithm, represented by the second technique (DOSP-RSA).

5.1 Results for Generate Symbol Table

An important data structure is the symbol table, which is generated in the DOSP-AES model to extract identifier names and data values from software source code. So it is easy to obfuscate the first two levels of obfuscation (Name, Data). When the bytecode is extracted during runtime, the third level of obfuscation, which is bytecode obfuscation, is

applied. A Symbol table can be generated from the source code written in C++, Java, and Android programming languages using the DOSP-AES model.

5.2 All the Trials during prepare our model

5.2.1 First Trial using normal implementation of Encryption Algorithm

During the first trial, the programming code was finished and prepared for obfuscation, as well as the general implementation of the encryption algorithm. After the programming code and key issue was resolved, the following problem occurred:

- A maximum of 14 KB of input files can be used and applied to the DOSP-AES code. If the input file size exceeds 14 kb, for example 20 kb, 30 kb. An error message appears as **"out of range at memory location", or "Access violation reading location".**
- **The following solutions were applied to solve the problem:**
  1. Different solutions were applied to allocate more memory for the visual studio program from the windows of the device.
  2. All indexes of the defined arrays in the code were modified so that the number of lines for the project analysis was increased. Also, the arrays were defined as text.
  3. Another version of the visual studio program was used.
  4. Hard disk device was replaced with another SSD hard disk.
  5. Another laptop device was used with higher memory and stronger specifications.

After applying all the above attempts to solve the problem. The problem still existed. So deep analysis was made in the code, an error in the block size in AES algorithm was found, and there is an error in key expansion. There was only 176-bit expansion in the code of

AES implementation. The Key Expansion function only expands to 176 bytes (note that AES128 bit key is expanded to 176-bytes), AES 192-bit key is expanded to 208-bytes. An AES 256-bit key is expanded to 240-bytes.

An attempt was made to separate the **structure.h** file to implement and modify the key expansion, however, this took a long time and had another error. Thus, another implementation and another library of encryption algorithms were used (note that the key expands is calculated from the following equation):

$$Expanded\ Key\ Size\ =\ (number\ of\ Rounds + 1)\ *\ Block\ Size \qquad (5.1)$$

Table 5.1 Shows some results that were made in this trial. Figure 5.1 shows the relation between the program size and obfuscation time in the first level of obfuscation (Source code obfuscation), (note that the time increases with the increase in the program size).

Table 5.1: Results of Obfuscation and De-Obfuscation process of the First Trial

First Trial							
Num.	File size (KB)	AES time obfuscation (s)			AES time de-obfuscation (s)		
		Key 256 bits Data	Key 192 bits Machine code	Key 128 bits Name	Key 256 bits Data value	Key 192 bits Machine code	Key 128 bits Name
1	3	3	4	2	4	5	3
2	6	3	6	2	6	7	5
3	9	4	6	5	4	7	5
4	12	5	8	7	8	9	7

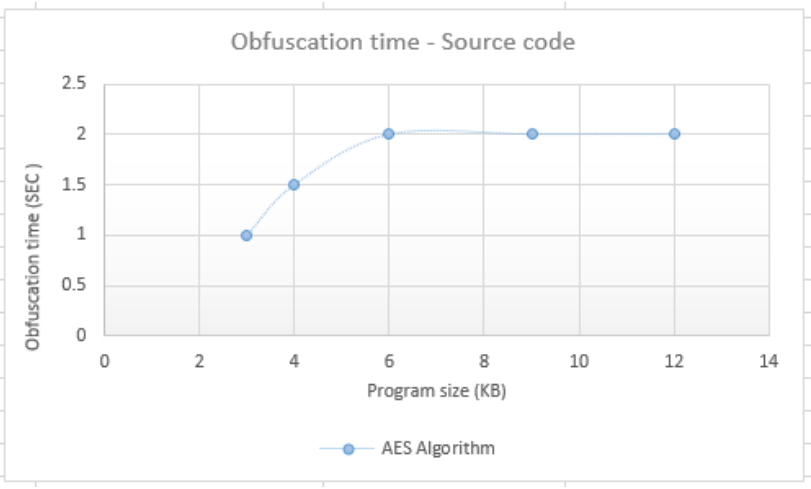


Figure 5.1: Obfuscation time in Source code level using AES algorithm

Figure 5.2 shows the relation between the program size and De-obfuscation time in the Source code, (note that the time increases with the increase in the program size). But the relation is not clear correctly because the max program sizes up to 14KB! Only.

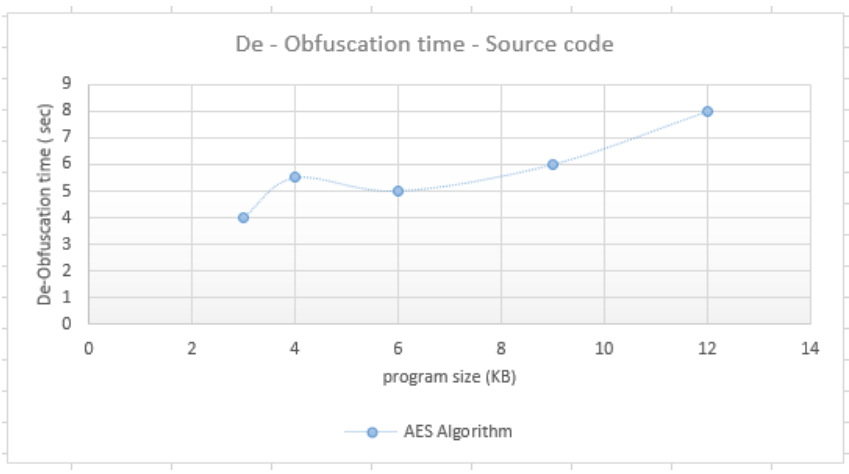


Figure 5.2: De- Obfuscation time in Source code level using AES algorithm

Figure 5.3 shows the relation between the program size and obfuscation time in the 3rd level of obfuscation (machine code obfuscation), (note that the time increases with the increase in the program size).

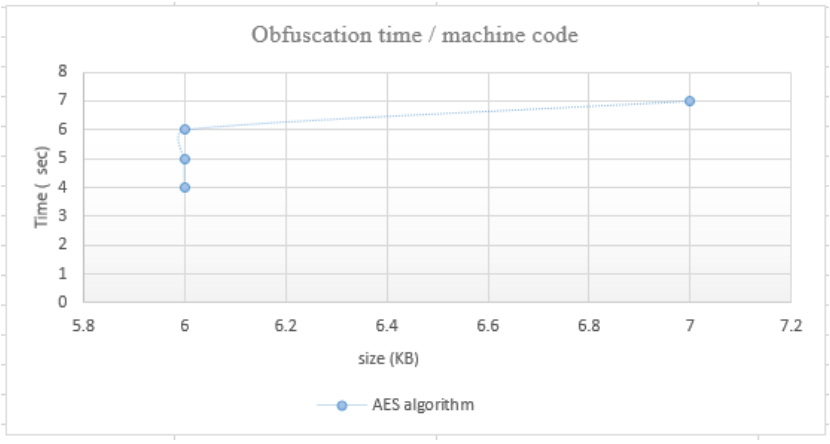


Figure 5.3: Obfuscation time in machine code level using AES algorithm

Figure 5.4 shows the relation between the program size and De-obfuscation time in the Source code De-obfuscation,(note that the time increases with the increase in the program size). But the relation is not clear correctly because it doesn't allow a file size of more than 14 KB!

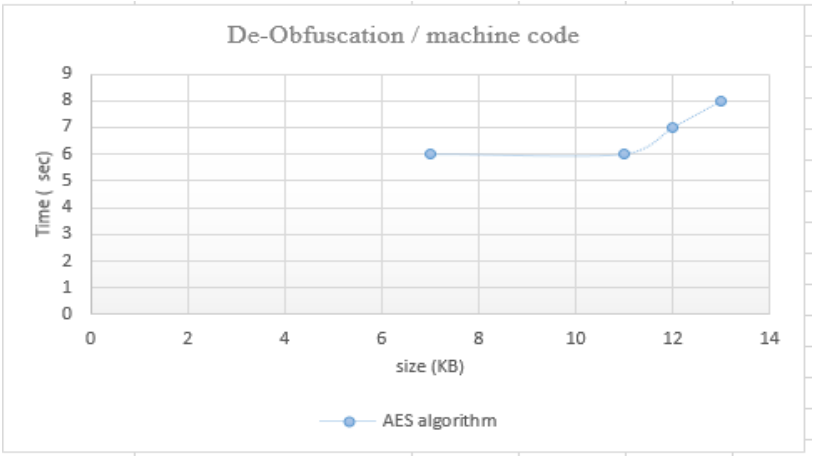


Figure 5.4: De-Obfuscation time in machine code level using AES algorithm

5.2.2 The Second Trial using Rajindi library and TPF_Math_Library

In the second trial, a newly developed implementation of AES using a newly developed library called the Rajindi **library** was implemented to build the DOSP-AES model. The library of **TPF_Math_Library** was used to implement the RSA encryption algorithm to build the DOSP-RSA model. This method is done successfully. However, the results of the comparison between DOSP-AES and DOSP-RSA are not accurate, and it is preferable to employ the same library of encryption algorithms.

So the two models were re-written using the same library. Therefore, if the results of the two models are compared to each other, the results will be more accurate. The **Chilkat encryption library** that implements and contains all encryption algorithms inside it is implemented in the revised implementation. So the Chilkat **encryption library** is used to

implement both AES, RSA, and DH algorithms, to make a comparison better and more accurate. The programs are written in the C++, Java, and Android programming languages.

5.2.3 The Third Trial Using Chilkat Library

The Chilkat library was used in this trial to implement both algorithms AES and RSA. The programming code was completed and the obfuscation model was developed. AES has been implemented successfully (random key generator, DH exchange key algorithms, and dividing the key into subkeys).

The problem that occurred in this experiment when using RSA algorithms in the DOSP-RSA. The same key lengths can't be used as in DOSP-AES (256, 192, 128 bits) because the Chilkat library does not support these key lengths. Chilkat library Supports data sizes ranging from 512 bits to 4096 bits [68].

5.2.4 The Fourth Trial using crypto++ library

Simulation results of the DOSP-AES model can be measured when using the crypto++ library, for both techniques DOSP-AES and DOSP-RSA. DOSP-AES is the obfuscation technique that employs the AES encryption algorithm and DOSP-RSA is the obfuscation technique that uses the RSA encryption algorithm.

The DOSP-AES model was applied to different programming languages, such as C++, Java, and Android. The first and second levels were applied to C++, Java, or Android source code, and the third level was added to C++ machine code, Java bytecode, and Android bytecode (DEX file).

5.3 Analysis of DOSP-AES model

Analysis of DOSP-AES model entails description of the DOSP-AES analysis for C++, Java and android programs.

5.3.1 DOSP-AES model for C++ programs

Time of the obfuscation and the de-obfuscation process of the DOSP-AES model is calculated in the 1st, 2nd and 3rd levels using different file of C++ programs. See Table 5.2.

Table 5.2: Obfuscation and De- Obfuscation time for DOSP-AES using different file sizes of C++ programs

DOSP-AES / C++ programs							
Num.	File size (KB)	AES time obfuscation (ms)			AES time de-obfuscation (ms)		
		1st level	2nd level	3rd level	1st level	2nd level	3rd level
1	200	1157	1180	860	1166	833	1407
2	400	1231	1011	1809	856	1125	2632
3	600	1315	1268	3043	1353	1225	3434
4	800	1864	1871	4323	1827	1802	5383
5	1000	2206	2245	5499	2271	2238	7090
6	1200	2590	2842	6452	2609	2702	7562
7	1400	3060	3141	7726	3481	4634	7819
8	1600	3482	3751	8434	4301	5027	9187
9	1800	4022	4111	9257	3773	6493	12589
10	2000	5455	5436	10564	5159	6846	13587

Note that, each value of file size is calculated in kilobyte (KB), and the time of obfuscation and De-Obfuscation process is calculated on (ms).

Obfuscation and De- Obfuscation time for DOSP-AES using different file sizes of C++ programs illustrated in Figure 5.5. The Figure shows that when increasing the file size, the time in both obfuscation and de-obfuscation will increase. The Figure represents the curve of the obfuscation process in all levels (Name, Data and machine code obfuscation). Note that, the obfuscation of Data using K_{256} bits take more time than the Name obfuscation using K_{128} bits. Also, the 3rd level (machine code) takes a longer time than the 1st and 2nd levels (Name and Data) because the file contains more complex information and instructions than the first two levels.

As shown in Figure 5.5. The K_{128} In obfuscation and de-obfuscation processes, they take approximately the same time but, in some cases, the de-obfuscated processes require more The K256 In file size up to 1200 KB takes the same time but when file size increases then the obfuscation process takes a longer time than the obfuscation process and this also depends on the data size that is obfuscated-obfuscating the K192 takes more time than obfuscating it.

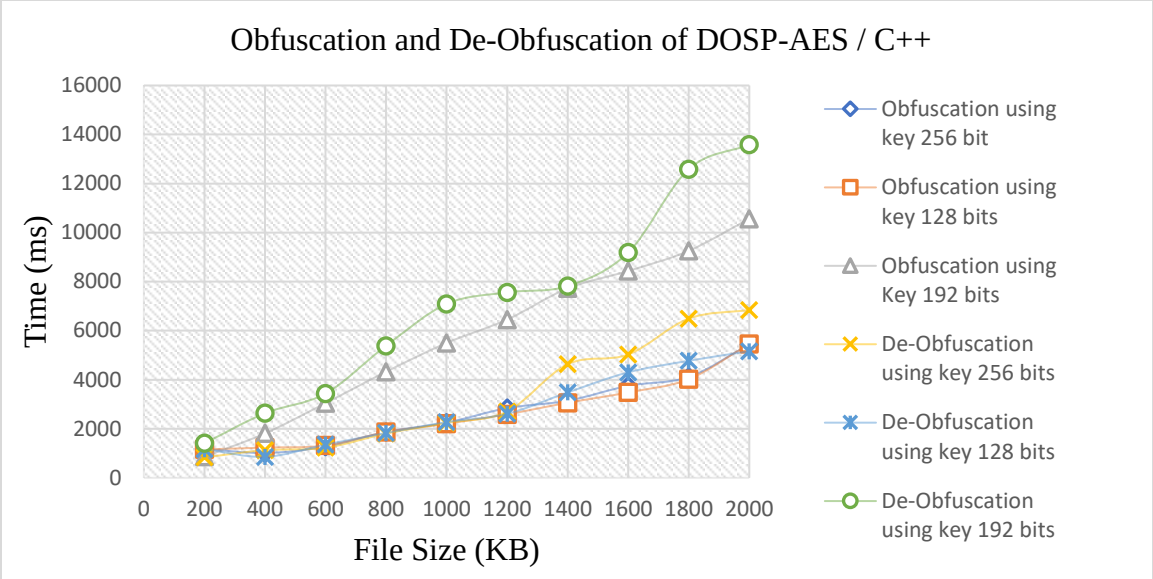


Figure 5.5: Obfuscation and Deobfuscation of DOSP-AES for C++ programs

5.3.2 DOSP-AES model for Java programs

The Time of obfuscation and de-obfuscation process of the DOSP-AES model is calculated in the 1st ,2nd and 3rd levels when different file sizes of Java programs are used. See Table 5.3.

Note that, each value of the file size is calculated in kilobyte (KB), and the time of obfuscation and De-Obfuscation process is calculated on (ms).

Obfuscation and De- Obfuscation time for DOSP-AES using different file sizes of Java programs illustrated in Figure 5.6. The Figure shows that when increasing the file size, the time in both obfuscation and de-obfuscation will increase. The Figure represents the curve of the obfuscation process in all levels (Name, Data and machine code obfuscation). Note that, the obfuscation of Data using K_{256} bits takes more time than the Name obfuscation using K_{128} bits.

Table 5.3: Obfuscation and De- Obfuscation time for DOSP-AES using different file sizes of Java programs

DOSP-AES / Java programs							
Num.	File size (KB)	AES time Obfuscation (ms)			AES time De-obfuscation (ms)		
		1st level	2nd level	3rd level	1st level	2nd level	3rd level
1	200	288	286	901	281	257	265
2	400	1184	1214	1527	885	875	1221
3	600	1802	1789	2858	2488	2240	3720
4	800	3219	3190	4801	3357	3165	5107
5	1000	4137	4010	8216	4403	3957	6288
6	1200	4957	4734	10261	5469	4642	7458
7	1400	6528	5306	10938	5491	4933	8793
8	1600	8390	8439	11546	9894	8332	15046
9	1800	9160	9079	12749	7137	6694	13684
10	2000	11484	13022	17629	13704	14195	14803

Also, the 3rd level (Bytecode obfuscation) using key K_{192} bits take longer time than 1st and 2nd level (Name and Data) because the file contains complex information and instruction than the first two levels.

As shown in Figure 5.6. The K_{128} take more time in the de-obfuscation process than obfuscation process. The K_{256} takes approximately the same time in the obfuscation and de-obfuscation process. The K_{192} take more time in a de-obfuscated process than obfuscated process.

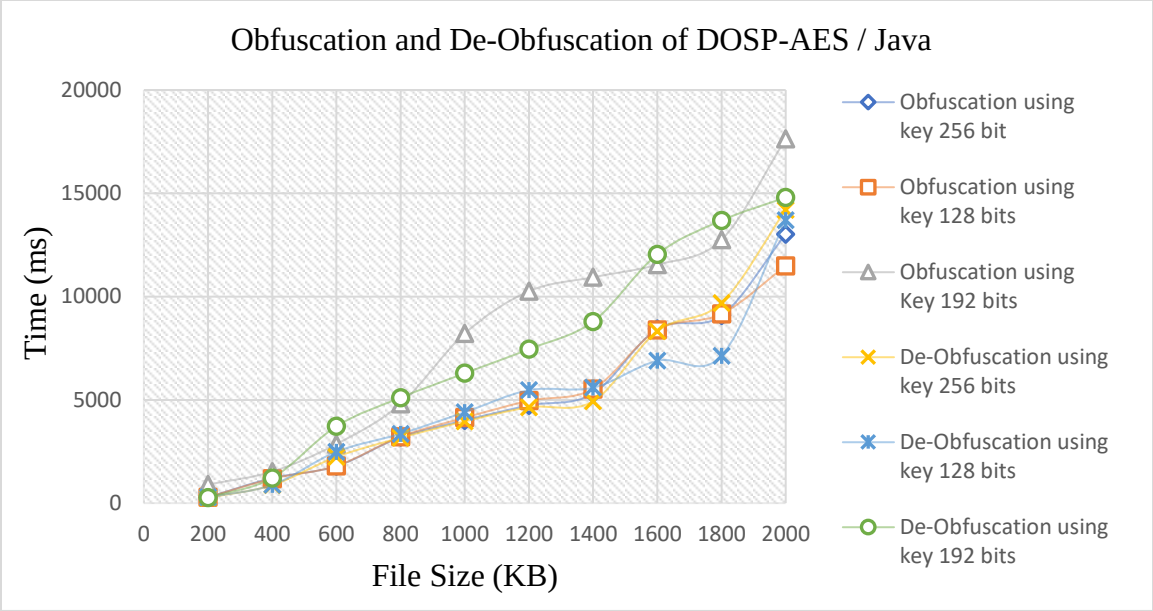


Figure 5.6 : Obfuscation and Deobfuscation of DOSP-AES for Java programs.

5.3.3 DOSP-AES model for Android programs

The Time of obfuscation and de-obfuscation process of DOSP-AES model is calculated in the 1st ,2nd and 3rd levels when different file sizes of Android programs are used. See Table 5.4.

Note that, each value of file size is calculated in kilobyte (KB), and the time of obfuscation and De-Obfuscation process is calculated on (ms).

Obfuscation and Deobfuscation time for DOSP-AES using different file sizes of Android programs illustrated in Figure 5.7. The Figure shows that when increasing the file size, the time in both obfuscation and de-obfuscation will increase. The Figure represents the curve of the obfuscation process in all levels (Name, Data and machine code obfuscation).

Table 5.4: Obfuscation and De- Obfuscation time for DOSP-AES using different file sizes of Android programs

DOSP-AES / Android programs							
Num.	File size (KB)	AES time Obfuscation (ms)			AES time De-obfuscation (ms)		
		1st level	2nd level	3rd level	1st level	2nd level	3rd level
1	200	333	574	458	914	994	832
2	400	743	851	612	749	815	675
3	600	1105	1162	900	1115	1252	1019
4	800	1437	1517	1235	1467	1646	1374
5	1000	1912	1959	1556	1841	2064	1729
6	1200	2150	2303	1816	2202	2467	2046
7	1400	2526	2681	2190	2557	2876	2383
8	1600	2966	3128	2513	2919	3301	2787
9	1800	3274	3493	2724	3358	3785	3055
10	2000	3585	3887	3095	3645	4108	3447

Note that, the obfuscation of Data using K_{256} bits takes more time than the Name obfuscation using K_{128} bits and K_{192} bits. Also, The K_{128} , K_{256} and K_{192} take more time in a de-obfuscated process than obfuscated process.

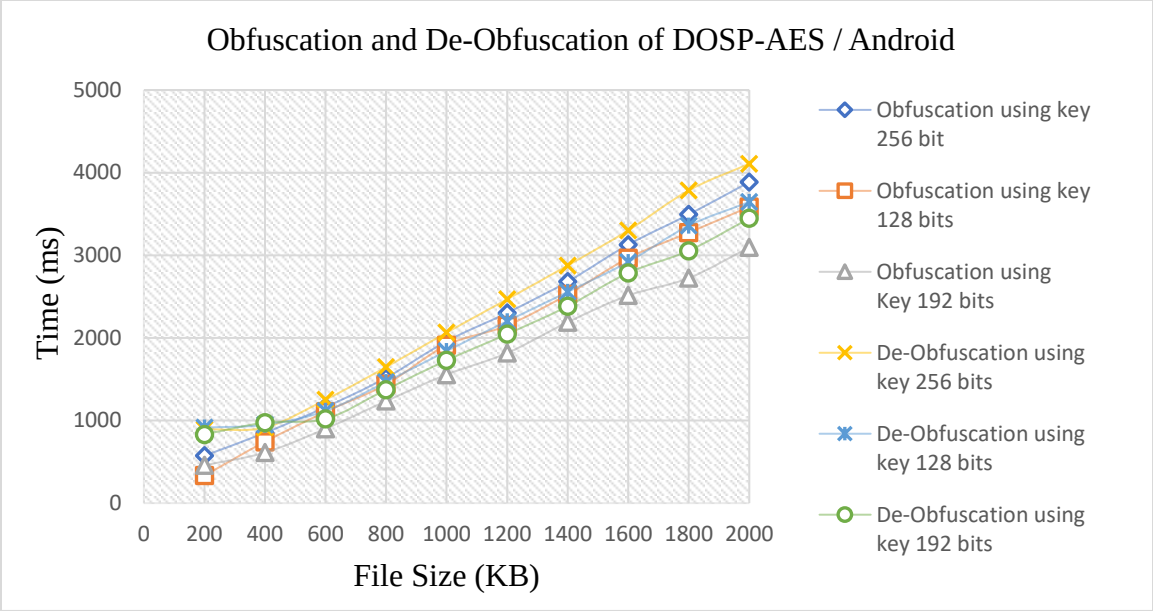


Figure 5.7: Obfuscation and De-Obfuscation of DOSP-AES for Android programs.

5.4 Analysis of DOSP-RSA model

Analysis of DOSP-RSA model contains description of the DOSP-RSA analysis for C++, Java and android programs.

5.4.1 DOSP-RSA model for C++ programs

The time of obfuscation and deobfuscation of the DOSP-RSA model is calculated in the 1st ,2nd and 3rd levels when different file sizes of C++ programs are used. See Table 5.5. Note that, each value of file size is calculated in kilobyte (KB), and the time of obfuscation and De-Obfuscation process is calculated on (ms).

Obfuscation and De-Obfuscation time for DOSP-RSA using different file sizes of C++ programs illustrated in Figure 5.8. The Figure shows that when increasing the file size, the time in both obfuscation and de-obfuscation will increase. The Figure represents the curve of the obfuscation process in all levels (Name, Data and machine code obfuscation).

Table 5.5: Obfuscation and De- Obfuscation time for DOSP-RSA using different file sizes of C++ programs

DOSP-RSA / C++ programs							
Num.	File size (KB)	RSA time Obfuscation (ms)			RSA time De-obfuscation (ms)		
		1st level	2nd level	3rd level	1st level	2nd level	3rd level
1	200	1271	1589	982	1233	1695	1173
2	400	1989	2245	2725	3020	4170	2907
3	600	3720	4198	4113	4753	6455	4408
4	800	4710	5034	5590	6163	8561	6716
5	1000	5760	5274	6486	10044	10517	7296
6	1200	6088	6119	7852	12537	16817	11643
7	1400	7075	5758	8298	5290	7157	5119
8	1600	8495	7222	9387	5890	8021	5587
9	1800	9144	8978	10072	6849	9184	6435
10	2000	9315	9894	11450	12221	18139	12735

Note that, the obfuscation of Data using K_{256} bits takes more time than the Name obfuscation using K_{128} bits and K_{192} bits. Also, the K_{128}, K_{256} and K_{192} take more time in a de-obfuscated process than obfuscated process. Note that is some programs with large sizes, K_{192} takes more time in obfuscation than de-obfuscation, this is related to the data type that is obfuscated.

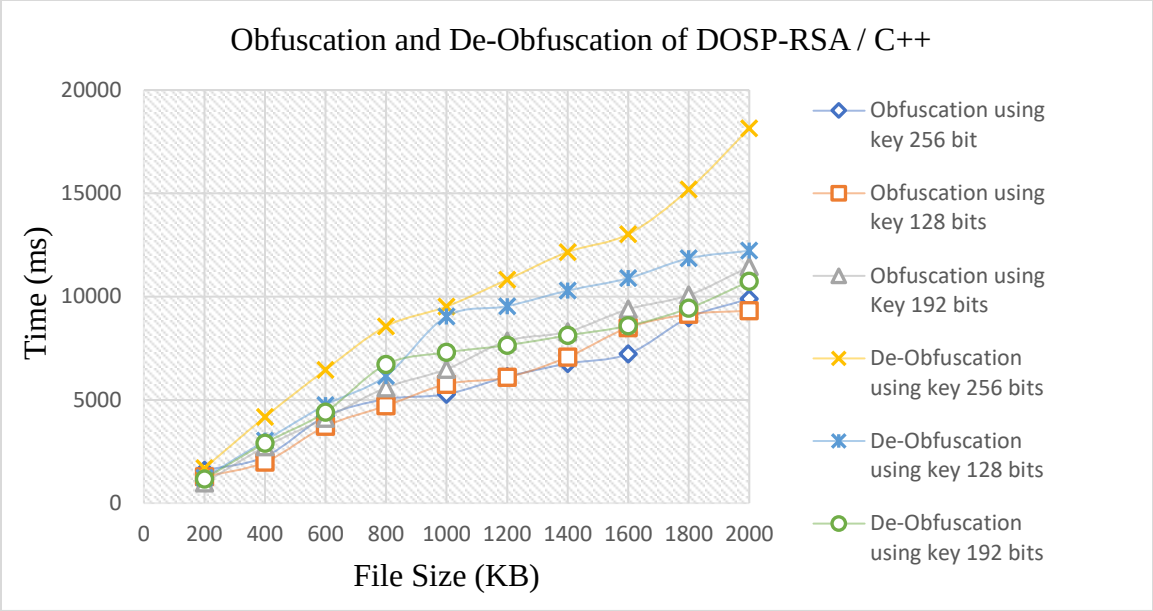


Figure 5.8 : Obfuscation and Deobfuscation of DOSP-AES for C++ programs

5.4.2 DOSP-RSA model for Java programs

The time of obfuscation and deobfuscation of the DOSP-RSA model in the 1st ,2nd and 3rd levels is taken when different file sizes of Java programs are used. See Table 5.6.

Note that, each value of the file size is calculated in kilobyte (KB), and the time of obfuscation and De-Obfuscation process is calculated on (ms).

Obfuscation and De-Obfuscation time for DOSP-RSA using different file sizes of Java programs illustrated in Figure 5.9. The Figure shows that when increasing the file size, the time in both obfuscation and de-obfuscation will increase. The Figure represents the curve of the obfuscation process in all levels (Name, Data and machine code obfuscation).

Table 5.6: Obfuscation and De- Obfuscation time for DOSP-RSA using different file sizes of Java programs

DOSP-RSA / Java programs							
Num.	File size (KB)	RSA time Obfuscation (ms)			RSA time De-obfuscation (ms)		
		1st level	2nd level	3rd level	1st level	2nd level	3rd level
1	200	690	726	1067	831	1007	1595
2	400	1487	1580	2293	1611	2122	3067
3	600	2916	2997	4759	3361	4205	7517
4	800	5084	5155	8052	6050	7554	10522
5	1000	6197	6574	9747	7241	9143	14056
6	1200	7543	7912	12333	8842	11270	16321
7	1400	8657	9169	13905	10126	12615	18071
8	1600	9774	11581	15799	11599	14696	21065
9	1800	10457	13075	17231	12874	15784	22511
10	2000	11857	15570	19510	13987	16782	25784

Note that, the obfuscation of Data using K_{256} bits takes more time than the Name obfuscation using K_{128} bits and K_{192} bits. Also, the K_{128} , K_{256}, K_{192} and, K_{192} take more time in a de-obfuscated process than obfuscated process.

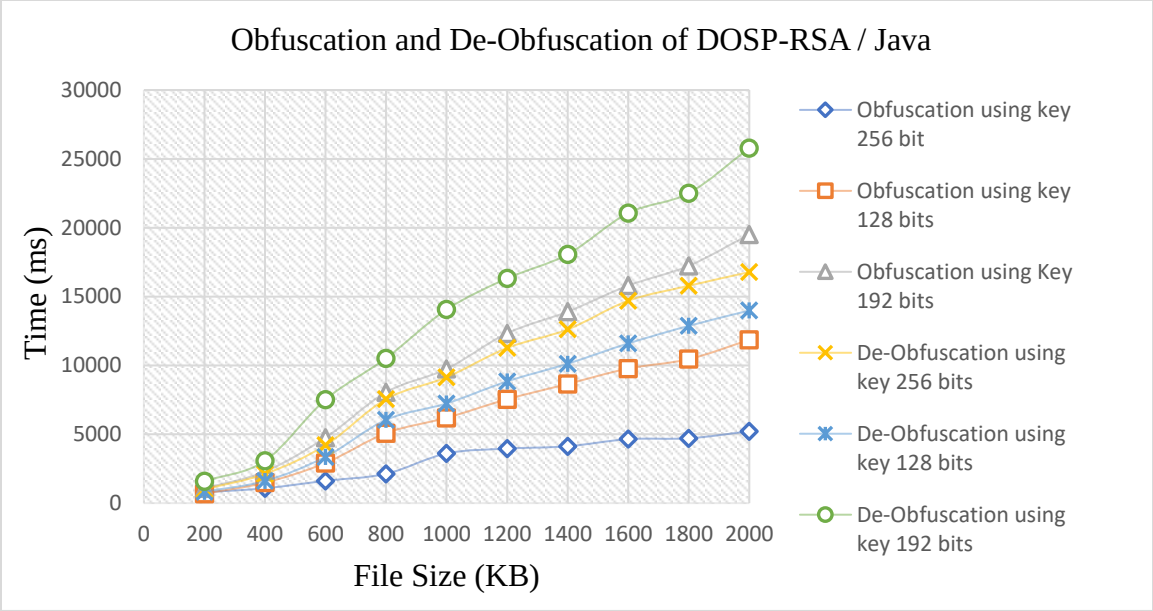


Figure 5.9 : Obfuscation and Deobfuscation of DOSP-AES for Java programs

5.4.3 DOSP-RSA model for Android programs

The Time of obfuscation and de-obfuscation process of DOSP-RSA model is calculated in the 1st ,2nd and 3rd levels when different file sizes of Android programs are used. See Table 5.7. Note that, each value of the file size is calculated in kilobyte (KB), and the time of obfuscation and De-Obfuscation process is calculated on (ms).

Obfuscation and De-Obfuscation time for DOSP-RSA using different file sizes of Android programs illustrated in Figure 5.10. The Figure shows that when increasing the file size, the time in both obfuscation and de-obfuscation will increase. The Figure represents the curve of the obfuscation process in all levels (Name, Data and machine code obfuscation). Note that, the obfuscation of Data using K_{256} bits takes more time than the Name obfuscation using K_{128} bits and K_{192} bits. Also, the K_{128} , K_{256}, K_{192} and, K_{192} take more time in a de-obfuscated process than obfuscated process.

Table 5.7: Obfuscation and De- Obfuscation time for DOSP-RSA using different file sizes of Android programs

DOSP-RSA /Android programs							
Num.	File size (KB)	RSA time Obfuscation (ms)			RSA time De-obfuscation (ms)		
		1st level	2nd level	3rd level	1st level	2nd level	3rd level
1	200	578	787	532	578	730	404
2	400	1175	1081	662	1175	1514	811
3	600	1763	1621	1010	1763	2267	1203
4	800	2356	2131	1316	2356	3033	1595
5	1000	3563	3610	1990	3563	4810	2372
6	1200	3912	3960	2267	3412	4550	2370
7	1400	4752	4130	2706	3952	5162	2794
8	1600	5048	4648	2914	5048	6100	3171
9	1800	5513	4709	3078	5213	6686	3527
10	2000	5701	5218	3398	5701	7507	3899

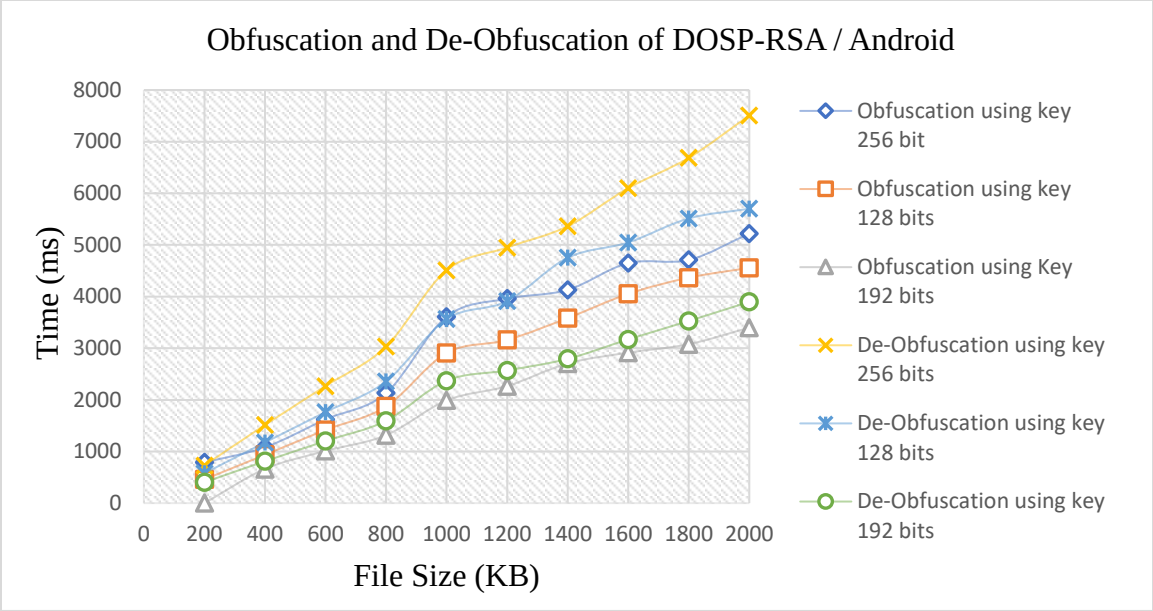


Figure 5.10 : Obfuscation and De-Obfuscation of DOSP-AES for Android programs

5.5 Comparison between DOSP-AES and DOSP-RSA Depend on the Time in all levels

A comparison is shown in this section between the time that different keys length takes in DOSP-AES and DOSP-RSA using C++, Java, Android, and the obfuscation process.

5.5.1 Comparison in the 1st level (Name Obfuscation)

A comparison between the DOSP-AES and DOSP-RSA techniques in 1st level obfuscation (Name obfuscation) which used 256-bit key length (32 byte). In C++ and Java and Android programs. See Table 5.8.

1. **1st level Using C++ program**

   The DOSP-RSA model takes more time than DOSP-AES models in the 1st level Name obfuscation when using C++ programs. So, the DOSP-AES model is faster than DOSP-RSA in Name obfuscation level in C++ programs. Figure 5.11 shows a

comparison between DOSP-AES and DOSP-RSA in name obfuscation level using C++ programs. We take the average time of 10 trials for each point.

Table 5.8: Name obfuscation (Source code) technique

1st level of obfuscation (Name obfuscation) using 16-byte key length						
File size (KB)	C++		Java		Android	
	DOSP-AES	DOSP-RSA	DOSP-AES	DOSP-RSA	DOSP-AES	DOSP-RSA
200	1157	1271	288	690	333	460
400	1231	1989	1184	1487	743	936
600	1315	3720	1802	2916	1105	1418
800	1864	4710	3219	5084	1437	1871
1000	2206	5960	4137	6197	1912	3908
1200	2590	8088	4957	7543	2150	3160
1400	3060	4075	5528	8657	2526	3184
1600	3482	4495	8390	9774	2966	4053
1800	4022	5144	7160	10457	3274	4066
2000	5455	9315	11484	11857	3585	4557

2. 1st level using Java programs

The DOSP-RSA model takes more time than DOSP-AES models in the 1st level Name obfuscation in Java programs. So, the DOSP-AES model is faster than DOSP-RSA in Name obfuscation level in Java programs. Figure 5.12 shows a comparison between DOSP-AES and DOSP-RSA in name obfuscation level using Java programs.

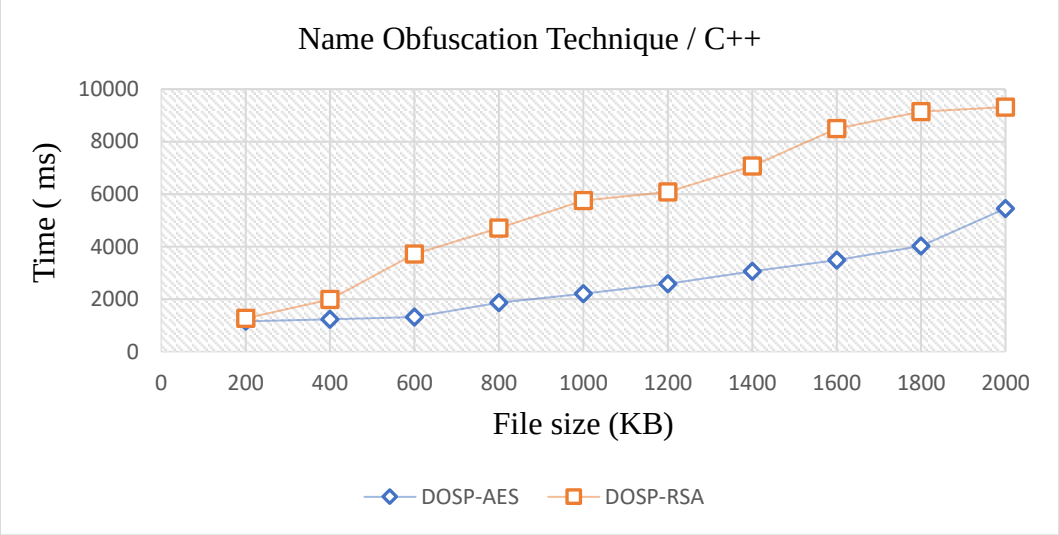


Figure 5.11: Name obfuscation technique of C++ programs.

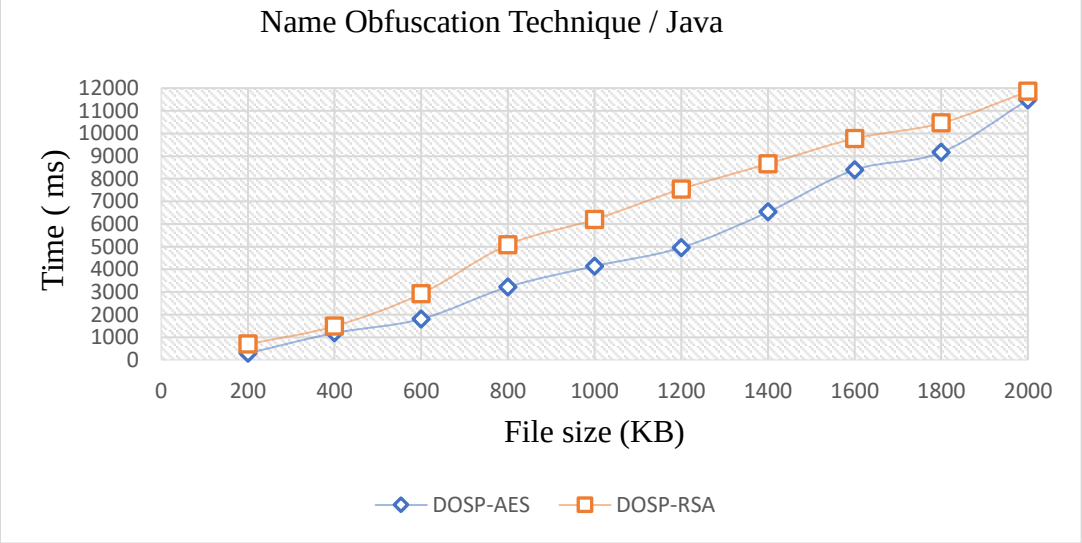


Figure 5.12: Name obfuscation technique of Java programs.

3. 1st level using Android programs

DOSP-RSA models take more time than DOSP-AES models in the 3rd level Name obfuscation in Android programs. So, the DOSP-AES model is faster than DOSP-RSA in Name obfuscation level using Android programs. Figure 5.13 shows a comparison between DOSP-AES and DOSP-RSA in name obfuscation level using Android programs.

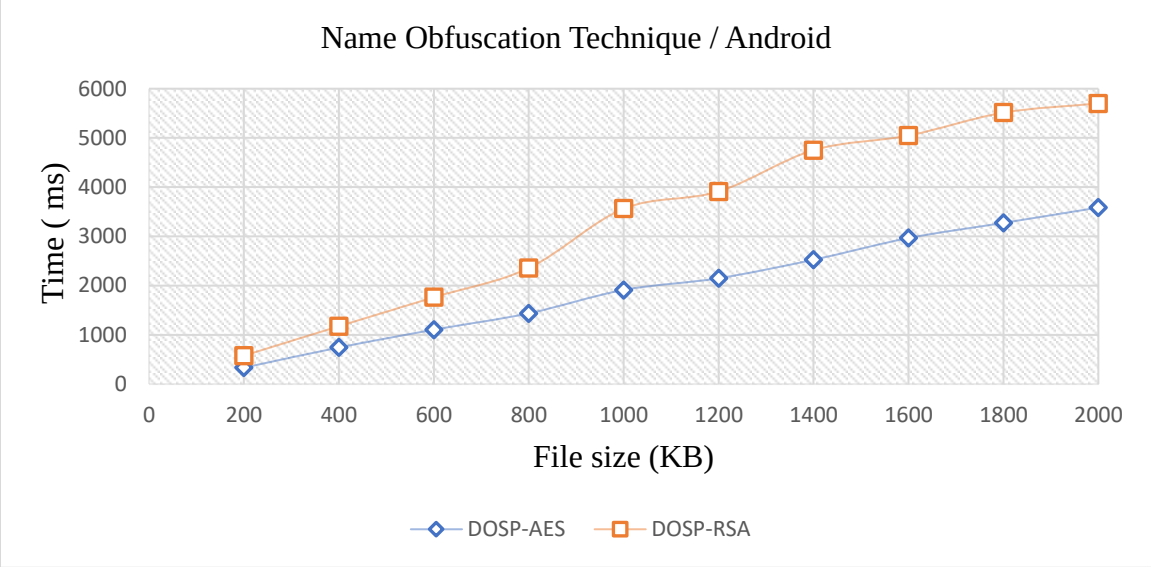


Figure 5.13: Name obfuscation technique of Android programs.

5.5.2 Comparison in the 2nd level (Data Obfuscation)

A comparison is shown in this section between the DOSP-AES and DOSP-RSA techniques in 2nd level obfuscation (Data obfuscation) in which 256-bit key length (32 byte) was used. In C++ and Java and Android programs. See Table 5.9.

1. 2nd level using C++ programs

The DOSP-RSA model takes more time than DOSP-AES models in the 2nd level Data obfuscation in C++ programs. So, the DOSP-AES model is faster than DOSP-

RSA in Data obfuscation level using C++ programs. Figure 5.14 shows a comparison between DOSP-AES and DOSP-RSA in Data obfuscation level using C++ programs.

Table 5.9: Data obfuscation technique

2nd level of obfuscation (Data Obfuscation) using 32-byte key length						
File size (KB)	C++		Java		Android	
	DOSP-AES	DOSP-RSA	DOSP-AES	DOSP-RSA	DOSP-AES	DOSP-RSA
200	1180	1089	286	726	574	522
400	1011	2245	1214	1580	851	1081
600	1268	4198	1789	2997	1162	1621
800	1871	5534	3190	5155	1517	2131
1000	2245	7274	4010	6574	1959	3610
1200	2842	8119	4734	7912	2303	3160
1400	3141	4758	5306	9169	2681	3630
1600	3751	5222	8439	11581	3128	4648
1800	4111	5978	7079	13075	3493	4709
2000	5436	10894	13022	15570	3887	5218

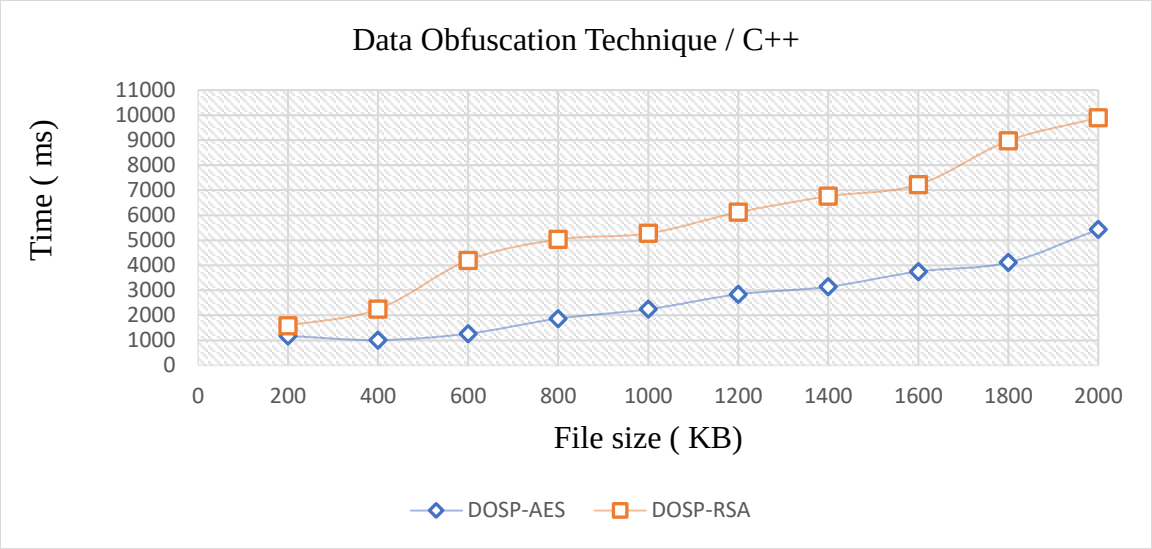


Figure 5.14: Data obfuscation technique of C++ programs.

2. 2nd level using Java programs

The DOSP-RSA model takes more time than DOSP-AES models in the 2nd level Data obfuscation in Java programs. So, the DOSP-AES model is faster than DOSP-RSA in Data obfuscation level using Java programs. Figure 5.15 shows a comparison between DOSP-AES and DOSP-RSA in Data obfuscation level using Java programs.

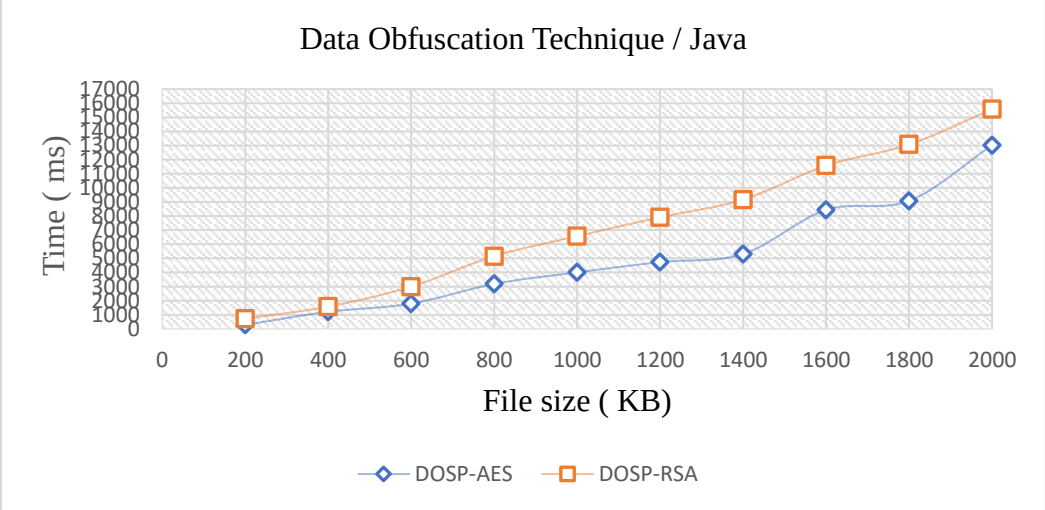


Figure 5.15: Data obfuscation technique of Java programs.

3. 2nd levels using Android programs

The DOSP-RSA model takes more time than DOSP-AES models in the 2nd level Data obfuscation in Android programs. So, the DOSP-AES model is faster than DOSP-RSA in Data obfuscation level using Android programs. Figure 5.16 shows a comparison between DOSP-AES and DOSP-RSA in Data obfuscation level using Android programs.

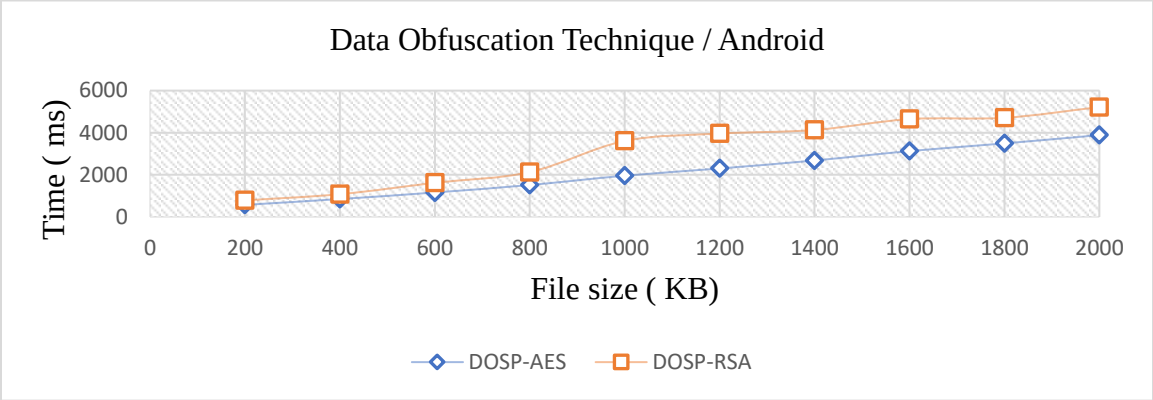


Figure 5.16: Data obfuscation technique of Android programs

5.5.3 Comparison in the 3rd level (Bytecode/machine code Obfuscation)

A comparison between the DOSP-AES and DOSP-RSA techniques in 3rd level obfuscation (Bytecode/machine code obfuscation) which used 256-bit key length (32 byte). In C++ and Java and Android programs. See Table 5.10.

1. 3rd level using C++ programs

The DOSP-RSA model takes more time than DOSP-AES models in the 3rd level machine code obfuscation in C++ programs. So, the DOSP-AES model is faster than DOSP-RSA in machine code obfuscation level using C++ programs. Figure

5.17 shows a comparison between DOSP-AES and DOSP-RSA in machine code obfuscation level using C++ program

Table 5.10: Bytecode / machine obfuscation technique

3rd level of obfuscation (Bytecode) using 24-byte key length						
File size (KB)	C++		Java		Android	
	DOSP-AES	DOSP-RSA	DOSP-AES	DOSP-RSA	DOSP-AES	DOSP-RSA
200	860	982	401	1067	458	423
400	1809	2725	1527	2293	612	662
600	3043	3113	2858	4759	900	1010
800	4323	5590	4801	8052	1235	1316
1000	5499	7486	8216	9747	1556	2767
1200	6452	6552	10261	12333	1816	1990
1400	3726	7298	7338	13905	2190	2206
1600	4134	11387	15046	15799	2513	2914
1800	4957	10072	9749	17231	2724	2878
2000	8564	10450	17629	19510	3095	3198

1. **3rd level using Java programs**

The DOSP-RSA model takes more time than DOSP-AES models in the 3rd level Bytecode obfuscation in C++ programs. So, the DOSP-AES model is faster than DOSP-RSA in Bytecode obfuscation level using Java programs. Figure 5.18 shows a comparison between DOSP-AES and DOSP-RSA in machine code obfuscation level using Java programs.

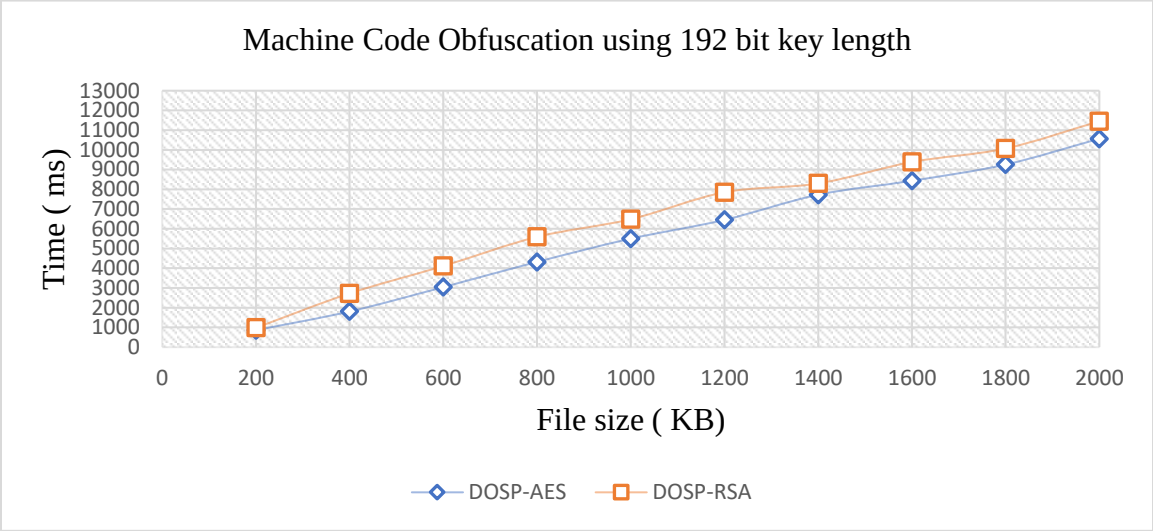


Figure 5.17: C++ Machine code obfuscation technique using 192-bit key length.

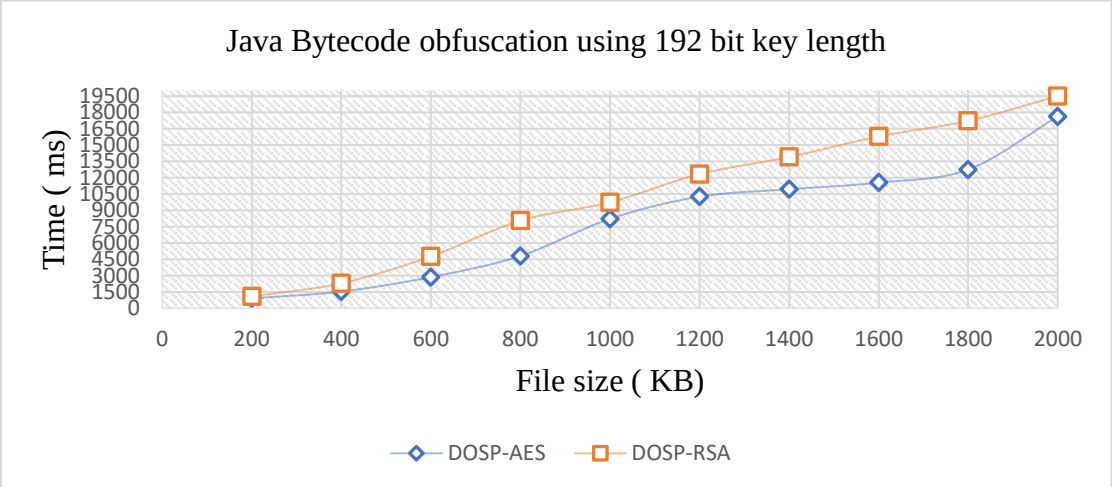


Figure 5.18: Java bytecode obfuscation technique using 192-bit key length

2. 3rd level using Java programs

The DOSP-RSA model takes more time than DOSP-AES models in the 3rd level Bytecode obfuscation in Android programs. So, the DOSP-AES model is faster than DOSP-RSA in Bytecode obfuscation level using Android programs. Figure 5.19 shows a comparison between DOSP-AES and DOSP-RSA in machine code obfuscation level using Android programs. DEX. File of Android programs is obfuscated.

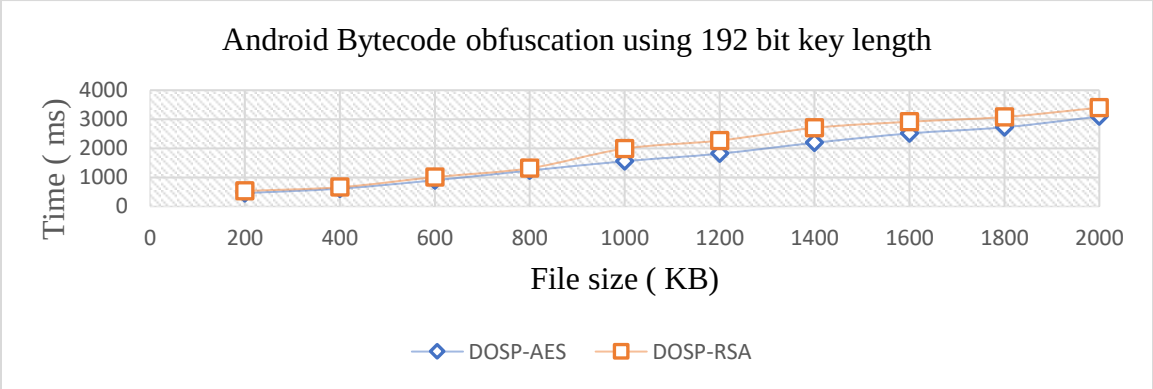


Figure 5.19: Android bytecode obfuscation technique using 192-bit key lengths

- As a result of the comparison between DOSP-AES and DOSP-RSA models. The DOSP-AES is faster than the DOSP-RSA in all the levels of obfuscation (Name, Data and bytecode), Which is implemented in DOSP-AES models. So, the DOSP-AES is faster and more feasible than DOSP-RSA.

5.6 Comparison between DOSP-AES and DOSP-RSA Depend on the key's length

A comparison is shown in this section between the time that different key lengths take in DOSP-AES and DOSP-RSA using C++, Java, Android, In obfuscation and De-obfuscation process.

5.6.1 Comparison between DOSP-AES and DOSP-RSA using different key lengths in C++ programs in the obfuscation process.

The K_{128} bits takes less time than K_{192} bits and K_{256} bits. Because K_{128} uses 10 rounds, K_{192} uses 12 rounds, and K_{256} uses 14 rounds. DOSP-AES takes less time in all key lengths K_{128}, K_{192}, and K_{256} than DOSP-RSA in obfuscated C++ programs. The result of this comparison is illustrated in Figure 5.20.

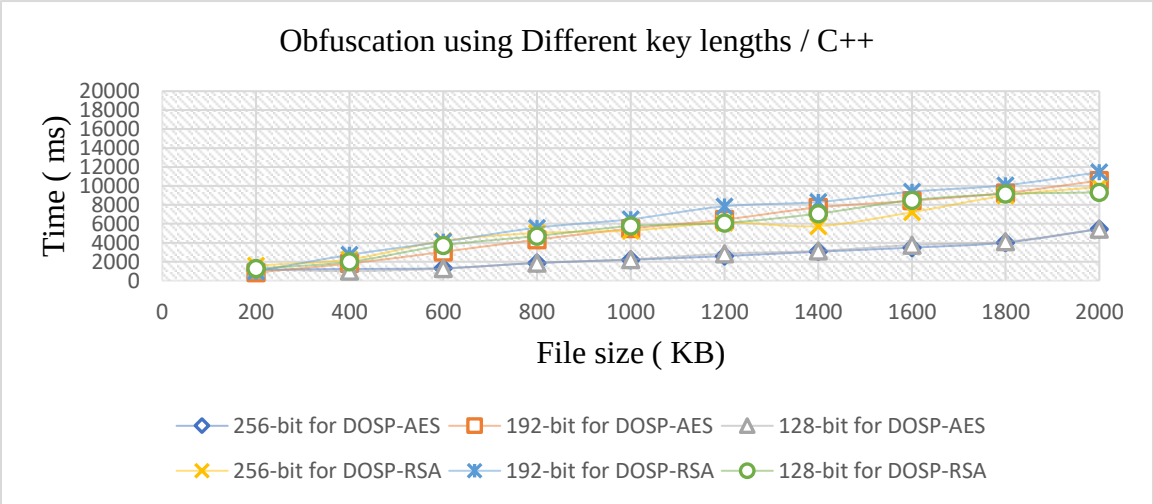


Figure 5.20: Obfuscation of DOSP-AES and DOSP-RSA using different key lengths in C++ programs

5.6.2 Comparison between DOSP-AES and DOSP-RSA using different key lengths in C++ programs in the De-obfuscation process.

- The K_{128} bits require the largest time in de-obfuscated processes in C++ programs in DOSP-RSA than DOSP-AES.
- The K_{192} bits take the longest time in the deobfuscated process in C++ programs in DOSP-AES than DOSP-RSA.
- and K_{256} bits consume the longest time in the deobfuscated process in C++ programs in DOSP-RSA than DOSP-AES.

These results appear because in DOSP-AES there is a client- server model so it will take time when the user requests the key and the server responds to the user. DOSP-RSA, however, uses the RSA encryption algorithm to generate and distribute keys without using third parties. . Therefore, the results show that DOSP-AES takes less time than DOSP-RSA in de-obfuscated time when utilizing 128 and K256. This comparison is illustrated in Figure 5.21.

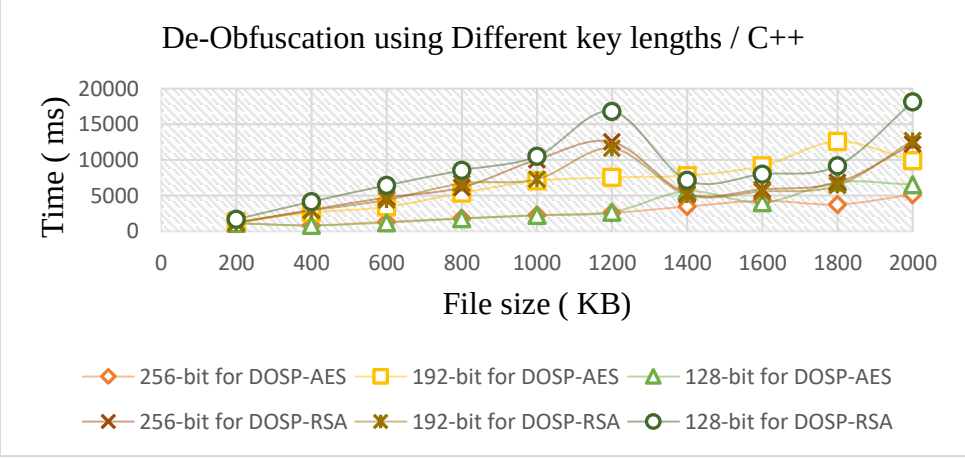


Figure 5.21: Deobfuscation of DOSP-AES and DOSP-RSA using different key lengths in C++ programs

5.6.3 Comparison between DOSP-AES and DOSP-RSA using different key lengths in Java programs in the obfuscation process.

- The K_{128} bits take the longest time in obfuscated processes in C++ programs in DOSP-RSA than DOSP-AES.
- The K_{192} bits require the longest time in an obfuscated process in C++ program in DOSP-RSA than DOSP-AES.
- and K_{256} bits consume the longest time in an obfuscated process in C++ program in DOSP-RSA than DOSP-AES.

These results show that the DOSP-AES takes less time in obfuscation in all key lengths K_{128}, K_{192} and K_{256} bits. This comparison is illustrated in Figure 5.22.

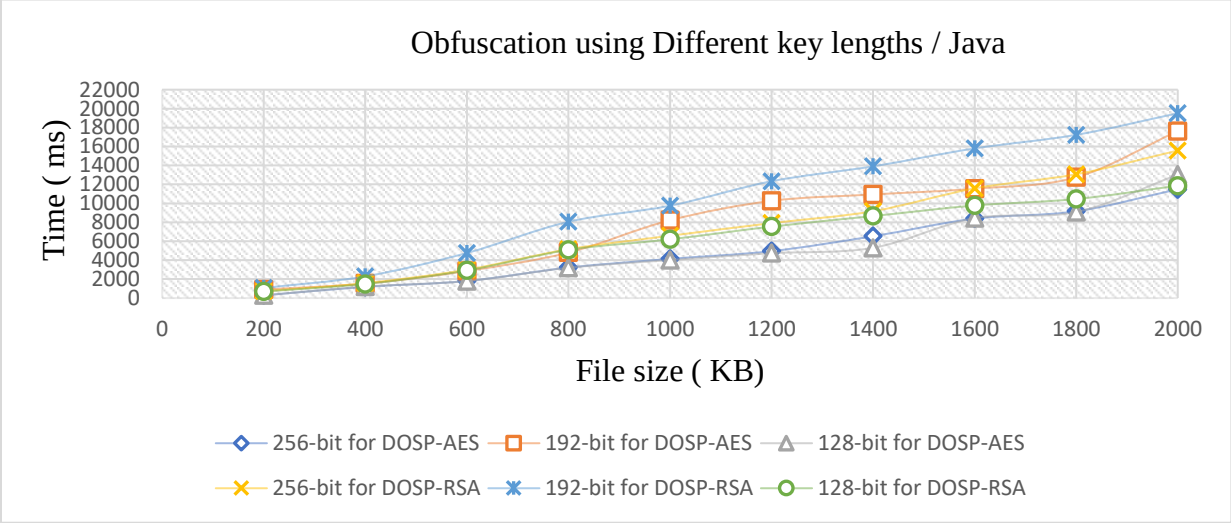


Figure 5.22: Obfuscation of DOSP-AES and DOSP-RSA using different key lengths in Java programs

5.6.4 Comparison between DOSP-AES and DOSP-RSA using different key lengths in Java programs in the De-obfuscation process.

- The K_{128} bits take the largest time in de-obfuscated processes in Java programs in DOSP-RSA than DOSP-AES.
- The K_{192} bits take the longest time in the deobfuscated process in Java programs in DOSP-RSA than DOSP-AES.
- and K_{256} bits take the largest time in de-obfuscated processes in Java programs in DOSP-AES.

These results show that the DOSP-AES requires less time in de-obfuscation for all key lengths, K128, K192, and K256 bits. These results appear because in DOSP-AES there is a client- > server model so it will take time when the user requests the key and the server responds to the user. But DOSP-RSA uses the RSA encryption algorithm to generate and distribute keys without involving a third party. So, the results show that DOSP-AES takes less time than DOSP-RSA in de-obfuscated time when using K128, K192, and K256 bits. This comparison is illustrated in Figure 5.23.

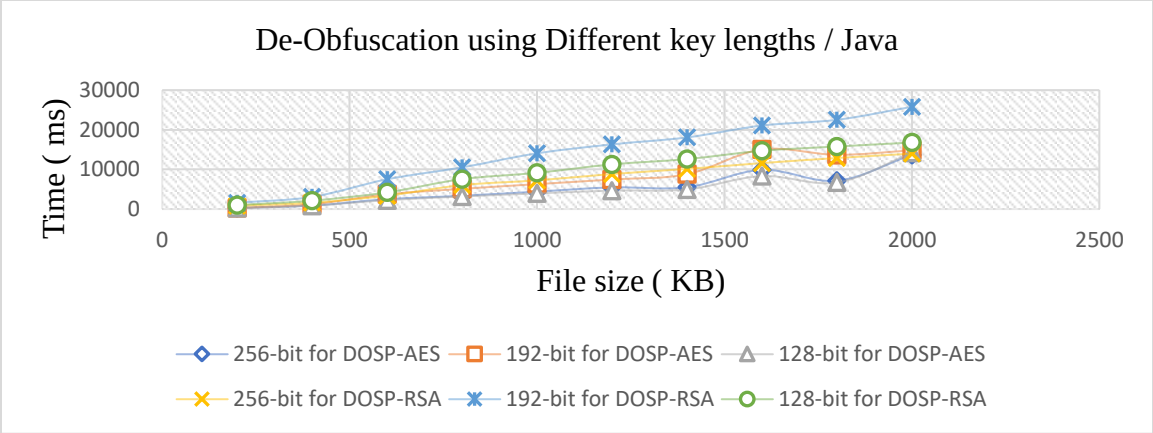


Figure 5.23: Deobfuscation of DOSP-AES and DOSP-RSA using different key lengths in Java programs

5.6.5 Comparison between DOSP-AES and DOSP-RSA using different key lengths in Android programs in the obfuscation process.

- The K_{128} bits takes the longest time in obfuscated processes in Android programs in DOSP-RSA than DOSP-AES.
- The K_{192} bits consumes the longest time in the obfuscated process in Android programs in DOSP-RSA than DOSP-AES.
- and K_{256} bits requires the longest time in the obfuscated process in Android programs in DOSP-RSA than DOSP-AES.

These results show that the DOSP-AES requires less time in obfuscation in all key lengths K_{128}, K_{192} and K_{256} bits. This comparison is illustrated in Figure 5.24.

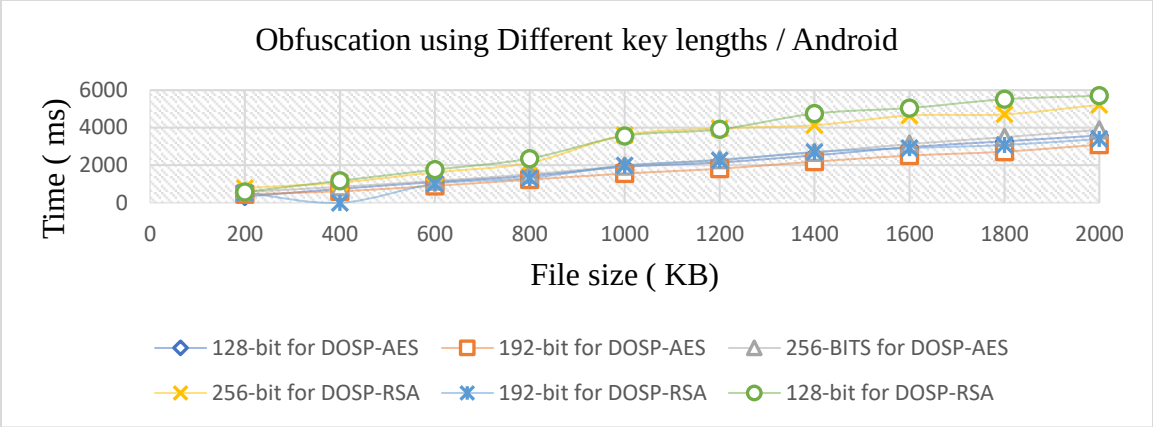


Figure 5.24: Obfuscation of DOSP-AES and DOSP-RSA using different key lengths in Android programs

5.6.6 Comparison between DOSP-AES and DOSP-RSA using different key lengths in Android programs in the De-obfuscation process.

- The K_{128} bits requires the longest time in de-obfuscated processes in Java programs in DOSP-RSA than DOSP-AES.
- The K_{192} bits takes the longest time in the deobfuscated process in Java programs in DOSP-RSA than DOSP-AES.
- and K_{256} bits consumes the longest time in de-obfuscated processes in Java programs in DOSP-AES.

These results show that the DOSP-AES requires less time in de-obfuscation for all key lengths, K128, K192, and K256 bits. These results appear because in DOSP-AES there is a client- > server model so it will take time when the user requests the key and the server responds to the user. In contrast, DOSP-RSA uses an RSA encryption algorithm for key generation and distribution without involving a third party. So, the results show

that DOSP-AES takes less time than DOSP-RSA in de-obfuscated time when using K128, K192, and K256 bits. This comparison is illustrated in Figure 5.25.

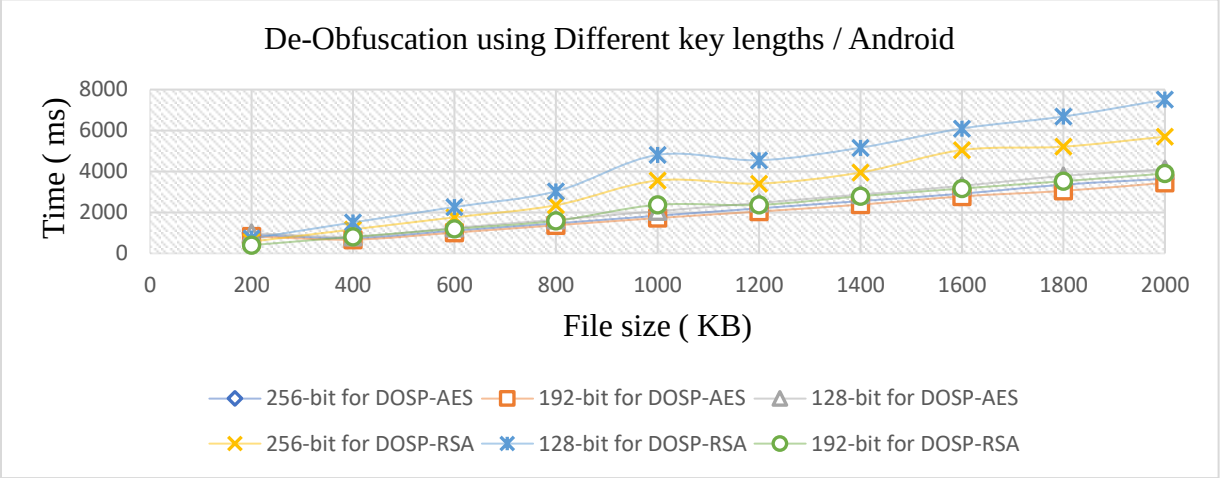


Figure 5.25: Deobfuscation of DOSP-AES and DOSP-RSA using different key lengths in Android programs

5.7 Comparison between DOSP-AES and DOSP-RSA Depend on the Programming languages C++, Java, Android

This section describe comparison between the programming languages C++, Java, Android. When using DOSP-AES and DOSP-RSA

5.7.1 1st level: Name Obfuscation

- C++, Java, Android programs using DOSP-AES talk shortest time than using it using DOSP-RSA.

- Android programs talk shortest time in Name Obfuscation technique than C++ and Java programs. Java programs talk largest time than C++ and Android programs.

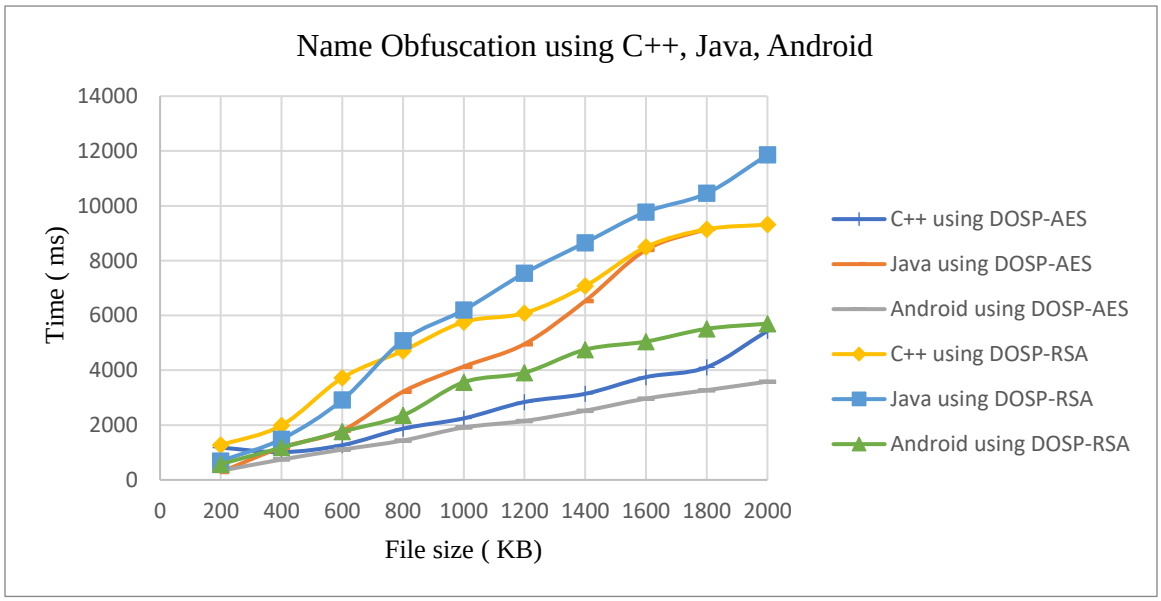


Figure 5.26: Name obfuscation using different key lengths in C++, Java, Android programs

5.7.2 2nd level: Data Obfuscation

- C++, Java, Android programs using DOSP-AES talk shortest time than using it using DOSP-RSA.

- Android programs talk shortest time in Name Obfuscation technique than C++ and Java programs. C++ programs talk largest time than Java and Android programs.

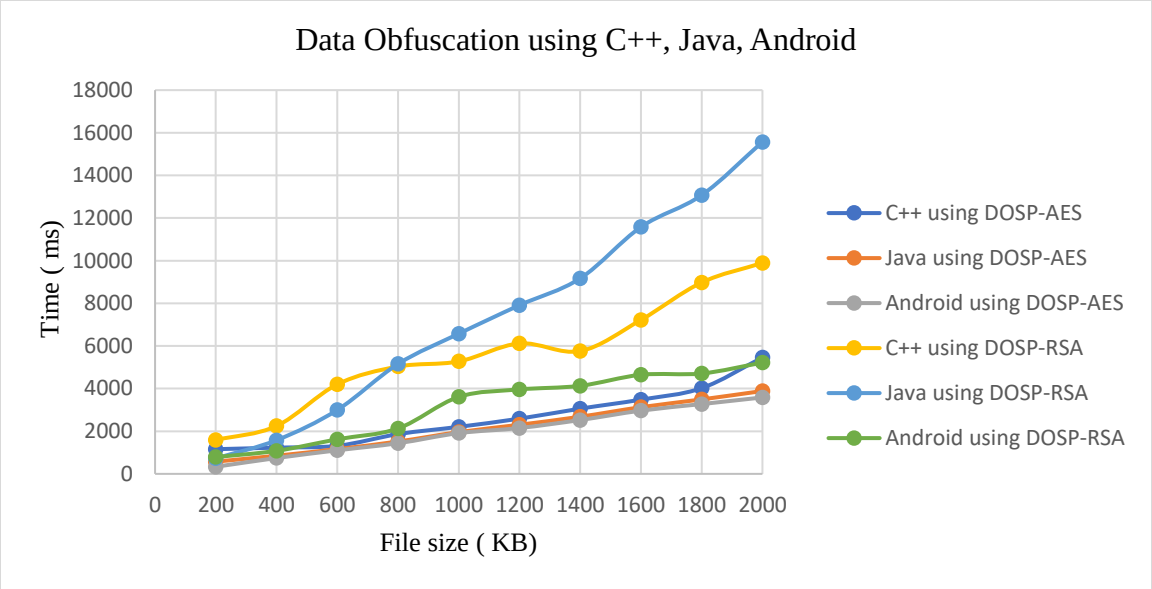


Figure 5.27: Data obfuscation using different key lengths in C++, Java, Android programs

5.7.3 3rd level: Bytecode Obfuscation

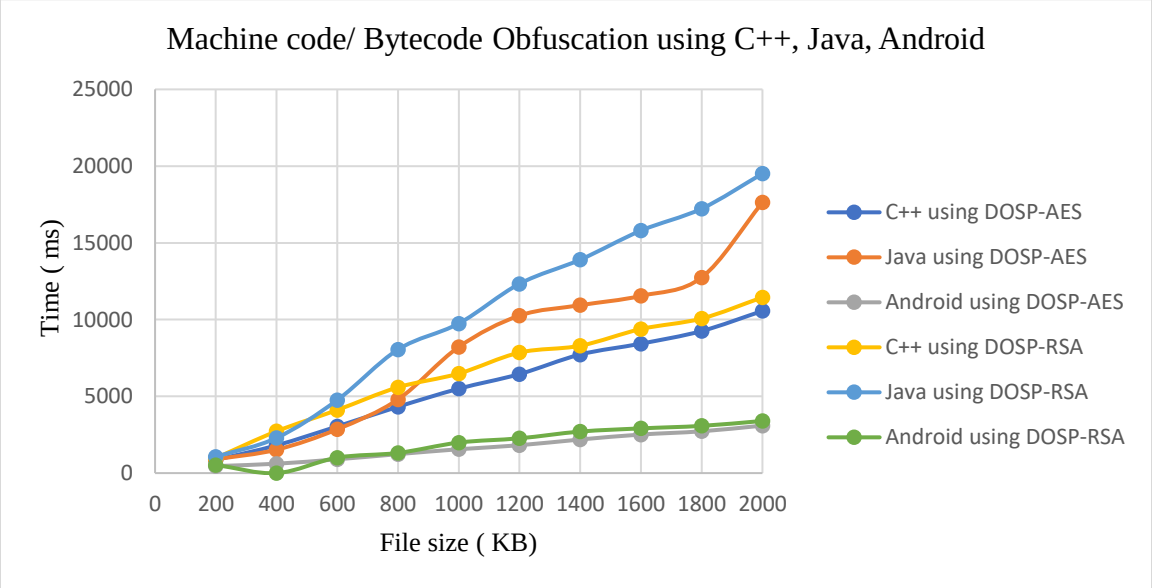


Figure 5.28: Machine code/bytecode obfuscation using different key lengths in C++, Java, Android programs

- C++, Java, Android programs using DOSP-AES talk shortest time than using it using DOSP-RSA.
- Android programs talk shortest time in Name Obfuscation technique than C++ and Java programs. Java programs talk largest time than C++ and Android programs.

5.8 Brute force attack

A brute-force attack involves systematically checking all possible key combinations until the correct key is found and is one method of attack when other weaknesses in an encryption system cannot be exploited. The length of the key used in encryption determines the practical feasibility of a brute-force attack, with longer keys being exponentially more difficult to crack than shorter ones.

Keyspace analysis is used to prevent an adversary from using a brute-force attack to find the key used to encrypt a message. The keyspace is usually designed to be large enough to make such a search infeasible. On average, half the keyspace must be searched to find the solution.

Advanced Encryption Standard (AES) can use a symmetric key of 128, 192, and 256 bits, resulting in a keyspace containing:

- $2^{256} = 1.1579 \times 10^{77}$ possible keys
- $2^{192} = 6.2771017 * 10^{57}$ possible keys
- $2^{128} = 3.4028237 * 10^{38}$ possible keys

Table 5.10: Possible combination keys of AES algorithm

Key size	Possible combination
AES 256	1.1579×10^{77}
AES 192	$6.2771017 * 10^{57}$
AES 128	$3.4028237 * 10^{38}$

In computing, floating-point operations per second (FLOPS, flops, or flop/s) is a measure of computer performance, useful in fields of scientific computations that require floating-point calculations. For such cases, it is a more accurate measure than measuring instructions per second.

FLOPS per cycle for Core processor x64 bit = 4

GFlops = (CPU speed in GHz) x (number of CPU cores) x (CPU instruction per cycle) x (number of CPUs per node) (5.2)

Number of instructions that executed in a millisecond on a cold CPU (turbo frequency times number of cores is taken).

For a minute, take the sustained rate with all cores running, not the turbo rate.

GFlops = 2.0Ghz * 2 * 2 * 2 = 16 GFlops / per second (5.3)

GFlops = 3.1Ghz * 2 * 2 * 2 = 24.1 GFlops / per millisecond (5.4)

No. of Flops required per combination check: 1000 (very optimistic but just assume for now)

No. of combination checks per second = $(4 \times 10^{15}) / 1000 = 4 \times 10^{12}$ // processor core x64bit – flops = 4 per second (5.5)

No. of seconds in one Year = 365 x 24 x 60 x 60 = 31536000 sec (5.6)

No. of Years to crack AES with 128-bit Key = $(3.4 \times 10^{38}) / [(4 \times 10^{12}) \times 31536000]$ (5.7)

$= (0.85 \times 10^{26})/31536000$

$= 2.695 \times 10^{18}$

= 2 billion billion years to crack 128bit AES

Cracking the 128-bit AES key using brute force attack takes 2 billion years. This is more than the age of the universe (13.75 billion years)

No. of Years to crack AES with 192-bit Key = $(6.28 * 10^{57}) / [(4 \times 10^{12}) \times 31536000]$

$= (1.57 \times 10^{45})/31536000$ (5.8)

$= 4.98 \times 10^{37}$

$= 49.8 * 10^{36}$

= 49 (billion)^4 years to crack AES-192 bit

- No. of Years to crack AES with 256-bit Key = (1.158×10^{77}) / $[(4 \times 10^{12}) \times 31536000)$ (5.9)

$$= (0.29 \times 10^{65})/31536000$$

$$= 9.18 \times 10^{56}$$

$$= 918 * 10^{54}$$

= 918 (billion)^6 years to crack 256 bit AES

Table 5.11: Time to crack AES various key using Brute force analysis

Key size	Time to crack (years)
AES – 256 bit	$918 * 10^{54}$
AES – 192 bit	$49.8 * 10^{36}$
AES – 128 bit	2.695×10^{18}

Despite belief and arguments, AES has never been cracked and remains safe against brute force attacks. However, the key size used for encryption should always be large enough that it could not be cracked by modern computers. This is despite considering advancements in processor speeds based on Moore's law.

5.9 Attack model

The proposed model DOSP-AES solved the threat models that were mentioned chapter one as follows:

DOSP-AES solved the Radare2, Ghidra, and IDA software problems by making the assembly and machine code more difficult to disassemble, debug, and decompile. DOSP-AES employs multiple levels of obfuscation, making it more difficult to analyze and comprehend the code. DOSP-AES prevent man in the middle attack by using DH key exchange algorithm. DOSP-AES prevents attempts at reverse engineering or debugging a target process. DOSP-AES employs name obfuscation, data obfuscation, and bytecode/machine code obfuscation, making decompiler, debugger, and disassembly more difficult to implement in the code.

5.10 Summary

When compared to DOSP-RSA, the values of time in DOSP-AES moved down and decreased, according to the results of our model. This indicates that the DOSP-AES obfuscation procedure takes less time than the DOSP-RSA. On every level. Because RSA takes a lengthy time, employing the AES algorithm in the levels is preferable to using the RSA method.

Furthermore, a comparison is made between all cases that include different file sizes at the same time for our first technique DOSP-AES and second technique DOSP-RSA, and obfuscation time is calculated in all levels of the two techniques, with DOSP-AES using AES encryption algorithm having the better obfuscation time in all levels. RSA is both more computationally intensive and much slower than AES. RSA is typically only used to encrypt small amounts of data. Because of the large number calculations, RSA is extremely slow.

Chapter Six

Conclusion and future work

Chapter six: Conclusion and future work

6.1 Conclusion

As data exchange on the internet gains importance and value over time, solutions to protect data from attackers become increasingly important. As a result, high-level protection techniques and measures are regarded as the most critical issue concerning the security of our communities.

Obfuscation is an important software protection technique that can solve more problems with software protection and intellectual property security than other software techniques such as cryptography. Obfuscation techniques benefit the entire computer science community. To analyze software, attackers employ a variety of tools for reverse engineering analysis.

The DOSP-AES model presents effective and promising software protection techniques for software intellectual property and distributed systems; DOSP-AES enhances the obfuscation intensity and enhances the difficulty of the reverse engineering process. The above mentioned model makes attacking complicated enough to repulse attackers rather than proving the strength of algorithms. Obfuscation solves many problems that cryptography has not yet addressed.

The early theoretical work showed that it is impossible to find a general-purpose obfuscator that can efficiently obfuscate programs, and secure programs according to the virtual black-box security model. These challenges have been addressed in this thesis.

In the DOSP-AES model, the obfuscation model is designed to protect and secure software, especially a distributed system from reverse engineering attacks. The idea of this thesis is to apply multi-levels of obfuscation techniques to support software security and to provide strong protection for software. The third and last level of obfuscation is the bytecode transformation level. This involves altering bytecode so that it is a more difficult order to read and understand for a hacker but remains functional. The first level uses code obfuscation, with the definition that an obfuscated program is more difficult to perceive and read than the original program. The second level uses data transformation methods by replacing sensitive information with data that looks like real data and making it useless to attack actors.

By combining these levels, a robust and high level of security for software against reverse engineering analysis has been reached. Second, to enhance the security level of software and make the model stronger and more difficult to analyze by reverse engineering, and an obfuscation model integrating the encryption algorithm and using the DH key-exchange algorithm to exchange the key.

Two models are built: DOSP-AES which uses AES encryption algorithm and DOSP-RSA which uses RSA encryption algorithm. Files from the GitHub site with a large size up to 2000 KB (2MB) were used for testing. Note that Files with a larger size than 2 MB can be used with the DOSP-AES model.

The results show that the DOSP-AES model increases the security level of the software protection when using AES combine with DH. So, it is of great importance that the security of the model is increased to maintain a high level of confidentiality for the users by combining the two algorithms in one process AES and DH algorithms; due to use the AES and DH algorithms in obfuscation, the speed and strength of AES and DH algorithms was obtained. Also using the technique of sending the key as one block to the client and but in the code using it as subkeys, so in the DOSP-AES model the key block is divided into three subkeys blocks, each subkey is Obfuscate / Deobfuscate a specific level of obfuscation, and each level can only be de-obfuscated with its key that was obfuscated. The evolutions of this technique confirm that employing the subkeys technique and using multi-levels of obfuscation with integrated AES encryption algorithms increase the difficulties of reverse engineering analysis or brute force attack.

The outcomes of this experiment show that the proposed technique protects and secures the software and application against hackers and reverse engineering analysis. The performance evaluation confirmed that the DOSP-AES model saves software with acceptable implementation time and memory utilization. The experimental results prove

that the proposed technique DOSP-AES provides stronger security and protection for software and applications against attackers and reverse engineering analysis. The performance evaluation confirms that our model protects software with acceptable execution time and memory usage.

6.2 Future Work

The proposed DOSP-AES model enhances software protection and security and enhances the difficulty of reverse engineering analysis. In no way does this claim to be the best achievable outcome. But DOSP-AES solves current problems in software and enhances obfuscation intensity. DOSP-AES builds robust obfuscation security. Towards achieving this significant goal, the following research directions are proposed.

1. To improve the DOSP-AES proposed model, use other obfuscation techniques like flow control as an additional level to make the DOSP-AES model more complex and more secure against reverse engineering attacks.

2. Use the Whitebox technique in the DOSP-AES model, which is the combination of obfuscation and encryption security techniques. In spite of the fact that attackers have full access and control over the code, Whitebox remains one of the most robust techniques for keeping the key and internal structure of algorithms secure.
3. Other security techniques can be produced to provide self-protection techniques, so if an attacker is trying to access the connection or steal software, the software is tampered with.

Finally, the goal is to improve the DOSP-AES model to achieve a higher and better level of security, protection, and robustness against all attackers and all their techniques.

Bibliography

[1] Krishan Kumar Kumar, Prabhpreet Kaur. A Thorough Investigation of Code Obfuscation Techniques for Software Protection. International Journal of Computer Sciences and Engineering, pp. 158-164, 2015.

[2] Marius Iulian Mihailescu, Stefania Loredana NITA, Marian Dorin PirloagaI. Software security techniques: risks and challenges. Mircea cel Batran Naval Academy Press. pp. DOI: 10.21279/1454-864X-16-I1-007, 2015.

[3] Wadha Al Nafjan and Murtaza Ali Khan. Software Copyright Infringement: Causes, Forms and Effects. 4th Conference on e-Learning Excellence. pp. 153-159, 2011.

[4] Savio Antony Sebastian, Saurabh Malgaonkar, Paulami Shah, Mudit Kapoor, and Tanay Parekhji. A Study and Review on Code Obfuscation. IEEE. 2016.

[5] Jan Cappaert. Code Obfuscation Techniques. Katholieke Universiteit Leuven, PHD thesis, Heverlee (Belgium),2012.

[6] https://techbeacon.com/security/reverse-engineering-attacks-6-tools-your-team-needs-know, 20,Nov, 2021.

[7] https://www.apriorit.com/ ,7,Jan, 2021.

[8] Moritz Schloegel et al. LOKI: Hardening Code Obfuscation Against Automated Attacks. IEEE.Germany, June 2021.

[9] Zhifeng Hu Hu, Serhii Havrylov, Ivan Titov. Obfuscation for Privacy-preserving Syntactic Parsing. International Conference on Parsing Technologies and the IWPT , pp. 62–72, 2020.

[10] Philippe Dugerdil and Roland Sako. Dynamic Analysis Techniques to Reverse. Springer International Publishing Switzerland. vol. DOI: 10.1007/978-3-319-30142-6_14, pp. 250–268, 2016.

[11] Zheheng Liang, Wenlin Li, Jing Guo, Deyu Qi, Jijun Zing. A parameterized flattening control flow based obfuscation algorithm with opaque predicate for reduplicate obfuscation. IEEE. Dec, 2017.

[12] Yanru Peng, Yuting Chen, Beijun Shen. An Adaptive Approach to Recommending. IEEE, 2019.

[13] Dr. Prerna Mahajan and Abhishek Sachdeva.A Study of Encryption Algorithms AES, DES and RSA. Global Journal of Computer Science and Technology, 2013.

[14] Bahare Hashemzade and Ali Maroosi. Hybrid Obfuscation Using Signals and Encryption. Hindawi Journal of Computer Networks and Communications, 2018.

[15] Prajakta Pahade and Mahesh Dawale. Introduction to compiler and IT phases. International Research Journal of Engineering and Technology (IRJET), pp. 1318- 1322, 2019.

[16] Steven P. Reiss. Software Tools and Environments. ACM Computing Surveys, vol. 28, pp. 281-284, March 1996.

[17] Rajendra Kumar. Introduction to Compiler. Published versionA New Compiler for Space-Time Scheduling of ILP Processors, pp. DOI: 10.13140/RG.2.2.31394.89289, 2019.

[18] Jad Matta. Paper on Symbol Table Implementation in Compiler Design. American University of Beirut, pp. DOI:10.13140/RG.2.2.14217.60005, 2019.

[19] Fan Wu, Hira Narang, Miguel Cabral. Design and Implementation of an Interpreter Using Software Engineering Concepts. International Journal of Advanced Computer Science and Applications, pp. 170-177, 2014.

[20] Prof. Mukund R. Joshi R. Joshi and Renuka Avinash Karkade. Network Security with Cryptography. International Journal of Computer Science and Mobile Computing, pp. 201-204, 2015.

[21] William Stallings. Cryptography and Network Security Principles and Practices, Fourth Edition.: Prentice Hall, 2005.

[22] Sarita Kumari. A Research Paper on Cryptography Encryption and Compression. International Journal Of Engineering And Computer Science ISSN:2319-7242, pp. 20915-20919, 2017.

[23] Omar Riyad. Cryptography and Data Security: An Introduction. Sohag University, Sohag, Egypt, pp. DOI:10.13140/RG.2.2.30280.16646, 2018.

[24] Barranca Parkway. Encryption and Its Importance to Device Networking, 2018.

[25] Sirajuddin Asjad. The RSA Algorithm. University of South-Eastern Norway, Dec 2019.

[26] Dr. M. Gobi, R. Sridevi , and R. Rahini priyadharshini. A Comparative Study on the Performance and the Security of RSA and ECC Algorithm. in Proceedings of the UGC Sponsored National Conference on Advanced Networking and Applications, 2015, pp. 168-171.

[27] Mashrufee Alam, Israt Jahan, Liton Jude Rozario, and Israt Jerin. A Comparative Study of RSA and ECC and Implementation of ECC on Embedded Systems. International Journal of Innovative Research in Advanced Engineering (IJIRAE), vol. 3, no. 3, March 2016.

[28] Rasha Samir Abdeldaym, Hatem Mohamed Abd Elkader, and Rida Hussein. Modified RSA Algorithm Using Two Public Key and Chinese Remainder Theorem. Electronics and Information Engineering, vol. 10, no. (DOI: 10.6636/IJEIE.201903 10(1).06), pp. 51-64, March 2019.

[29] Keith Palmgren. Diffie-Hellman Key Exchange: A Non-mathematician's explanation. ISSA Journal, Oct 2006.

[30] Neha Grg and Partibha Yadav. Comparison of Asymmetric Algorithms in Cryptography. International Journal of Computer Science and Mobile Computing, vol. 3, no. 4, pp. 1190-1196, April 2014.

[31] Aqeel Khalique, Singh Kuldip, and Sandeep Kumar Sood. Implementation of Elliptic Curve Digital Signature Algorithm. International Journal of Computer Applications, vol. 2, pp. 21-27, May 2010.

[32] Ako Muhamad Abdullah. Advanced Encryption Standard (AES) Algorithm to Encrypt and Decrypt Data. Cryptography and Network Security , June 2017.

[33] Isnar Sumartono, Andysah Putera Siahaan, and Nova Mayasari. An Overview of the RC4 Algorithm. IOSR Journal of Computer Engineering, vol. 18, no. 6, pp. 67-73, 2016 Dec.

[34] Sheetal Charbathia and Sandeep Sharma. A Comparative Study of Rivest Cipher Algorithms. International Journal of Information and Computation Technology, vol. 4, pp. 1831-1838, 2014.

[35] Prashant Kumar Dey and Tarun Kumar Dey, ANALYSIS OF THE SECURITY OF AES, DES, 3DES AND IDEA NXT ALGORITHM. INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY,Oct 2015.

[36] Karthik.S and Muruganandam.A, Data Encryption and Decryption by Using Triple DES and Performance Analysis of Crypto System. International Journal of Scientific Engineering and Research (IJSER), vol. 2, pp. 24-31, Nov 2014.

[37] Mozhgan Mokhtari. Analysis and Design of Affine and Hill Cipher. Journal of Mathematics Research, vol. 4, Feb 2012.

[38] Ritu Tripathi and Sanjay Agrawal. Comparative Study of Symmetric and Asymmetric Cryptography Techniques. International Journal of Advance Foundation and Research in Computer (IJAFRC), vol. 1, June 2014.

[39] Rushdi A. Hamamreh, Israa M. Al-Qatow, and Mohammed A. Jamoos. SSCC: Cryptosystem Model for Cloud Data Sharing. in The 19th International Conference on Microarchitecture and Multiprocessor (ICMMT), Barcilona, 2017, pp. 3094-3098.

[40] Eldad Eilam. Reversing: Secrets of Reverse Engineering. CRYPTOLOGIC, vol. 29, no. 3, pp. 282-283, May 2005.

[41] A. Bryant, R Mills, M Grimaila, and G Peterson. Top-Level Goals in Reverse Engineering Executable Software. Journal of Information Warfare, vol. 1, pp. 32-43, May 2013.

[42] G. Sreeram Reddy, Manzoor Hussian, and K. Srinivasa Rao. Latest Research on Reverse Engineering Technology: Review. in Proceedings of the International conference on Paradigms in Engineering & Technology (ICPET), 2016, pp. 945-948.

[43] N. Rathore and P. Jane. REVERSE ENGINEERING APPLICATIONS IN MANUFACTURING INDUSTRIES: AN OVERVIEW. in DAAAM INTERNATIONAL SCIENTIFIC BOOK. Vienna, Austria: DAAAM International, 2014, ch. 45, pp. 567-576.

[44] www.preemptive.com. 20,NOV.2021.

[45] Sebastian Schrittwieser and Stefan Katzenbeisser. Code Obfuscation against Static and Dynamic Reverse Engineering.2012.

[46] Pierre-Louis Cayrel, Mohamed ElYousf, Gerhard Hoffmann, Mohammed Meziani, and Robert Niebuhr. Recent progress in code-based cryptography. CASED – Center for Advanced Security Research Darmstadt, 2012.

[47] Seungkwang Lee and Myungchul Kim. Improvement on a Masked White-box Cryptographic Implementation. Information Security Research Division, ETRI, May 2020.

[48] Okan Seker, Thomas Eisenbarth, and Maciej Liśkiewicz. A White-Box Masking Scheme Resisting Computational and Algebraic Attacks. IACR Transactions on Cryptographic Hardware and Embedded Systems, vol. 2021, no. 2, pp. 61-105, Feb 2021.

[49] Boaz Barak. On the (Im)possibility of Obfuscating Programs. July 2010.

[50] Muhammad Rizwan Asghar, Steven D. Galbraith, Andrea Lanzi, Giovanni Russello, and Lukas Zobernig. Towards a Theory of Special-purpose Program Obfuscation. Research Gate, Nov 2020.

[51] Hui Xu, Yangfan Zhou, Jiang Ming, and Michael Lyu. Layered obfuscation: a taxonomy of software obfuscation techniques for layered security. Xu et al. Cybersecurity, pp. doi.org/10.1186/s42400-020-00049-3, April 2020.

[52] Yakobu Dasari, Hemanth Kumar Kalluri, and Venkatesulu Dondeti. A Crypto Scheme Using Data Obfuscation of Entity Detection and Replacement for Private Cloud. International Journal of Safety and Security Engineering, vol. 10, pp. 417-422, April 2020.

[53] Chandan Kumar Behera and D. Lalitha Bhaskari. Different Obfuscation Techniques for Code Protection. in 4thInternational Conference on Eco-friendly Computing and Communication Systems, 2015, pp. 757 – 763.

[54] Jien-Tsai Chan and Wuu Yang. Advanced obfuscation techniques for Java bytecode. The Journal of Systems and Software, Aug 2004.

[55] Jong-Wouk Kim, Jiwon Bang, and Mi-Jung Choi. Defeating Anti-Debugging Techniques for Malware Analysis Using a Debugger. Advances in Science, Technology and Engineering Systems Journal, vol. 5, pp. 1178-1189, Nov 2020.

[56] Erwin Adi, Zubair Baig, and Philip Hingston. Stealthy Denial of Service (DoS) Attack Modelling and Detection for HTTP/2 Services. Journal of Network and Computer Applications 91, pp. DOI:10.1016/j.jnca.2017.04.015, April 2017.

[60] Gurpreet Singh and Supriya Kinger.Integrating AES, DES, and 3-DES Encryption Algorithms for Enhanced Data Security. International Journal of Scientific & Engineering Research, vol. 4, no. 7, July 2013.

[61] Subhi R. M. Zeebaree. DES encryption and decryption algorithm implementation. Indonesian Journal of Electrical Engineering and Computer Science, vol. 18, pp. 774-781, May 2020.

[62] Akshita Bhandari. A framework for data security and storage in Cloud Computing. IEEE 2016 International Conference on Computational Techniques in Information and Communication Technologies (ICCTICT), New-Delhi, 2016.

[63] https://www.chilkatsoft.com/rsa-c++.asp, 10,Jan, 2021.

[64] Dindayal Mahto, Dilip Kumar Yadav, RSA and ECC: A Comparative Analysis,. International Journal of Applied Engineering Research, 2017

[65] Avijit Mallika, Abid Ahsanb, Mhia Md. Zaglul Shahadata and Jia-Chi Tsou. Man-in-the-middle-attack. International Journal of Data and Network Science.2019.

[66] Adwan Yasin, Ihab Nasra , Dynamic Multi Levels Java Code Obfuscation Technique (DMLJCOT), IEEE, Palestine,2016

Appendix A

The following is Pseudo code of DOSP-AES model and each function in the code that present each instruction:

1.Server creates a socket with a specific port number for incoming connection requisitions.

//Create a socket.

```
ListenSocket = socket(result->ai_family, result->ai_socktype,
 result->ai_protocol);
```

//Bind socket with TCP.

```
iResult = bind(ListenSocket, result->ai_addr, (int)result->ai_addrlen);
```

//Listen to the client.

```
iResult = listen(ListenSocket, SOMAXCONN);
```

6 Server wait for incoming connection

```
ClientSocket = accept(ListenSocket, NULL, NULL);
```

7 Client creates a socket and connects to the server with host id and port number.

```
ConnectSocket = socket(ptr->ai_family, ptr->ai_socktype, ptr->ai_protocol);
iResult = connect(ConnectSocket, ptr->ai_addr, (int)ptr->ai_addrlen);
```

8 Client sends a requestion for downloading a specific software with a command "soft:<name>".

```
sendbuf = "soft:" + softname
iResult = send(ConnectSocket, sendbuf, (int)strlen(sendbuf), 0);
```

9 Server receives the requestion and answers with “ok”.

iResult = recv(ClientSocket, recvbuf, recvbuflen, 0);

10 Client starts DH algorithm to make secret key.

Creates an instance of CkDh class of chilkat library.

CkDh clientDh; //creat client DH

clientDh.UnlockComponent("abc123");

clientDh.UseKnownPrime(1);

const char* p = 0;

int g;

p = clientDh.p(); //get prime

g = clientDh.get_G(); //get generator : 2 or 5

const char* eClient = 0;

eClient = clientDh.createE(96);

•Sends the three values to the server so that it can exchange key and secret key.

iResult = send(ConnectSocket, p, (int)strlen(p), 0); //send prime

iResult = send(ConnectSocket,gen,(int)strlen(gen), 0);//send generator

iResult = send(ConnectSocket,eClient,(int)strlen(eClient),0); //send exchange value

11 Server starts DH algorithm to make secret key.

Creates instance of DH object

CkDh serverDh; //create server DH

serverDh.UnlockComponent("abc123");

•Receives prime number, generator and client exchange key.

```
iResult = recv(ClientSocket, recvbuf, recvbuflen, 0); //receive p
string prime(recvbuf);
p = prime.c_str();

iResult = recv(ClientSocket, recvbuf, recvbuflen, 0); // receive g
g = atoi(recvbuf);

//receive exchange value of client
iResult = recv(ClientSocket, recvbuf, recvbuflen, 0);
string ekey(recvbuf);
eClient = ekey.c_str();
```

•Makes server exchange key and sends to client.

```
const char* eServer = 0;
eServer = serverDh.createE(96); //make exchange value
iResult = send(ClientSocket, eServer, strlen(eServer), 0); //send exchange value to client
```

•Make a secret key using a client exchange key.

```
const char* kServer = 0;
kServer = serverDh.findK(eClient);
```

8.Client receives a server exchange key and makes a secret key using it.

```
kServer = serverDh.findK(eClient);
string strE(recvbuf);
eServer = strE.c_str();

//make secret key using exchange key of server
```

```
const char* kClient = 0;
kClient = clientDh.findK(eServer);
```

9. Part one: Explanation of AES Obfuscation and de-obfuscation code

1) Obfuscation AES (3 levels)
   The Obfuscation is processed as following:
   - Obfuscation of variable names

```
//First, get the file name to obfuscate the variable names from the user

string filename;
cout << "Please enter the name of a file for variable name: ";
cin >> filename;

//Set the obfuscation key and initialization vector to encryptor
encryptor.SetKeyWithIV(nameKey, sizeof(nameKey), iv);

//the path of original file
string filepath = "../Data/name&value/" + filename + ".txt";
//the path of encrypted file
string nameencpath = "../Data/name&value/" + filename + "_nameencrypted.txt";

string line; //used to store one line string to encrypt name
vector<string> tokens; //used to store every word separated by tab
int len;

//input stream and output stream to read and save
fstream instrm, outstrm;

//open the file to encrypt variable names and do encryption per line
instrm.open(filepath, ios::in);
if (instrm.is_open()) {
        //encrypt variable names
        outstrm.open(nameencpath, ios::out);

        while (getline(instrm, line)) { //read one line
        plain_text = "";
        tokens.clear();
        splitString(line, '\t', tokens);

        if (tokens.size() >= 6) { //if dataline

        //variable name to encrypt
        plain_text = tokens.at(6);

        /// Encryption
        string encryptedStr;
        CryptoPP::StreamTransformationFilter encFilter(encryptor,
                new CryptoPP::StringSink(encryptedStr));

        encFilter.Put(reinterpret_cast<const
```

```
                byte*>(plain_text.c_str()), plain_text.size());
                encFilter.MessageEnd();

                /// convert encrypted name to hex representation
                string hexenstr;
                StringSource ss(encryptedStr, true, new HexEncoder(new
                StringSink(hexenstr)));


                //save encrypted string to the output file specified by      //nameencpath
                tokens.at(6) = hexenstr;
                for (string token : tokens) {
                        outstrm << token << '\t';
                }
                outstrm << endl;
                }
                else {
                        outstrm << line << endl;
                }
                }
                outstrm.close();
}
instrm.close()
```

- Obfuscation of variable value and byte code
  This step is so similar to the case of variable name encryption step

4 De-obfuscation process

The de-obfuscation process is done on the client side.

```
// get the file name to decrypt
string filename;
cout << "Please enter the name of a file for variable name: ";
cin >> filename;

//set the key and iv to decryptor
decryptor.SetKeyWithIV(nameKey, sizeof(nameKey), iv);

string nameencpath = "../Data/name&value/" + filename +          "_nameencrypted.txt";
string namedecpath = "../Data/name&value/" + filename +          "_namedecrypted.txt";

string line;
vector<string> tokens;
fstream instrm, outstrm;
instrm.open(nameencpath, ios::in);

//open the file and read per one line
if (instrm.is_open()) {
//decrypt  names
outstrm.open(namedecpath, ios::out);
while (getline(instrm, line)) { //get one line

        tokens.clear();

        splitString(line, '\t', tokens);
        if (tokens.size() >= 6) { //if dataline
```

```
        string hexenstr = tokens.at(6).c_str();

        string encryptedStr;
        StringSource ss(hexenstr, true, new HexDecoder(new
        StringSink(encryptedStr)));

        /// Decryption
        string decryptedStr;
        CryptoPP::StreamTransformationFilter decFilter(decryptor, new
        CryptoPP::StringSink(decryptedStr));
        decFilter.Put(reinterpret_cast<const byte*>(encryptedStr.c_str()),
        encryptedStr.size());
        decFilter.MessageEnd();
        tokens.at(6) = string(decryptedStr);

        //save to output file
        for (string token : tokens) {
                outstrm << token << '\t';
        }
        outstrm << endl;
        }
        else {
                outstrm << line << endl;
        }
}
outstrm.close();
```

- Value and bytecode decryption is so similar to the case of name decryption

5 Explanation of RSA Obfuscation and De-Obfuscation / code

Obfuscation RSA (3 levels)

- Obfuscation of the names

```
//get the name of file to encrypt
string filename;
cout << "Please enter the name of a file for variable name: ";
cin >> filename;


string filepath = "../Data/name&value/" + filename + ".txt";
string nameencpath = "../Data/name&value/" + filename + "_nameencrypted.txt";

string line;
vector<string> tokens; //store words separated by tab
int len;
fstream instrm, outstrm;
instrm.open(filepath, ios::in);

//open the file to encrypt and read per one line
if (instrm.is_open()) {
        //encrypt  names
        outstrm.open(nameencpath, ios::out);
        while (getline(instrm, line)) {
```

```
                flat_text = "";
                tokens.clear();
                splitString(line, '\t', tokens);

                if (tokens.size() >= 6) { //if dataline

                flat_text = tokens.at(6); //the word representing name

                // Treat the message as a big endian byte array
                Integer iFlat_text = Integer((const byte*)flat_text.data(),
                flat_text.size());
                Integer iEncStr = namePubKey.ApplyFunction(iFlat_text);
                stringstream ss;
                ss << hex << iEncStr << endl;
                ss >> encryptedStr;

                tokens.at(6) = encryptedStr;
                //save the encrypted word to output file
                for (string token : tokens) {
                        outstrm << token << '\t';
                }
                outstrm << endl;
                }
                else {
                outstrm << line << endl;
                }
        }
        outstrm.close();
```

- Obfuscation of value and bytecode is so similar to the last name encryption process
-

4.3.2.1 How to Generate the random Key in the code

```
AutoSeededRandomPool rng;
InvertibleRSAFunction params;
params.GenerateRandomWithKeySize(rng, 128);
RSA::PrivateKey namePrivKey(params);
RSA::PublicKey namePubKey(params);

params.GenerateRandomWithKeySize(rng, 256);
RSA::PrivateKey valuePrivKey(params);
RSA::PublicKey valuePubKey(params);

params.GenerateRandomWithKeySize(rng, 192);
RSA::PrivateKey bcdPrivKey(params);
RSA::PublicKey bcdPubKey(params);
```

4.3.2.2 De- Obfuscation RSA

```
//get the name for decryption
cout << "Please enter the name of a file for variable name: ";
cin >> filename;

nameencpath = "../Data/name&value/" + filename + "_nameencrypted.txt";
string namedecpath = "../Data/name&value/" + filename +
```

```
"_namedecrypted.txt";

//open the file and read per one line
instrm.open(nameencpath, ios::in);
if (instrm.is_open()) {
	//decrypt  names
	outstrm.open(namedecpath, ios::out);
	clock_t tic = clock();
	while (getline(instrm, line)) {

	tokens.clear();

	splitString(line, '\t', tokens);

	if (tokens.size() >= 6) { //if dataline
	encryptedStr = tokens.at(6).c_str();

	/// Decryption
	Integer iEnc(encryptedStr.c_str());
	Integer iDec = namePrivKey.CalculateInverse(rng, iEnc);
	size_t req = iDec.MinEncodedSize();
	decryptedStr.resize(req);
	iDec.Encode((byte*)decryptedStr.data(), decryptedStr.size());

	tokens.at(6) = decryptedStr;
	//save the decrypted word to the output file
	for (string token : tokens) {
		outstrm << token << '\t';
	}
	outstrm << endl;
	}
	else {
		outstrm << line << endl;
	}
	}
	outstrm.close();
```

نموذج التشفير الموزع لحماية البرمجيات

اعداد: مي كامل عاطف عمرو

اشراف: الدكتور رشدي حمامره

ملخص:

في هذا البحث، تم اقتراح نموذج تشفير جديد وهو تقنية التشفير الموزعة لحماية البرمجيات من عمليات الهندسة العكسية والتغيير. تم تطبيق هذه التقنية على برامج C++, java, android.

يتكون النموذج المقترح من نظام لتشفير البرمجيات. ويتكون من عملية تشفير على ثلاثة مستويات:

المستوى الأول، هو تشفير الاسم البرمجي (name code obfuscation)، ويتضمن هذا المستوى إعادة تسمية المعرفات function, والمتغيرات variable بأسماء غير مفهومة ولا معنى لها، باستخدام خوارزمية معيار التشفير المتقدم AES (Advance Encryption Standard) باستخدام مفتاح عشوائي random key طوله 128 بت لتشفيرها. يتضمن تشفير الكود البرمجي استخدام طريقة إعادة التسمية والإزالة. الإزالة تعني حذف المعلومات غير المفيدة، والتعليقات comments التي لن يتم استخدامها في البرنامج. يجعل الحذف من الصعب على المهاجم attacker القراءة والفهم والتحليل. تتضمن إعادة التسمية تحويل أسماء المتغيرات والمعرفات الأخرى من أجل منع القرصنة او فهم البرنامج وتحليله.

المستوى الثاني، هو تشفير البيانات(data obfuscation) ، يهدف هذا المستوى إلى إخفاء قيم البيانات وحمايتها. حيث يقوم النموذج المقترح بتشفير قيم الثوابت constant ومتغيرات البرامج المحلية والعالمية global /local variables لجعل عملية الهندسة العكسية أكثر تعقيدًا، باستخدام خوارزمية AESبمفتاح طوله 256 بت. ويشفر نفس المتغير كل مره في نفس البرنامج بطريقة مختلفة يظهر فيها خلال الكود.

المستوى الثالث، هو تشفير البايت كود (bytecode obfuscation)، هذا المستوى يقوم بتعديل البايت كود. يقوم DOSP-AES بتشفير المعرفات (Identifier) في ملف البايت كود، باستخدام خوارزمية AES بمفتاح طوله 192

بت. وإذا تم محاوله تشغيل البرنامج وفك تشفيره بدون مفتاح فك التشفير، ف ان البرنامج يعطي أخطاء ولن يتم تشغيله. يهدف تشفير البايت كود لجعل البايت كود أكثر غموضاً، وجعل الملف المشفر أكثر صعوبة بالفهم، لذلك يجب على المهاجم أن يقضي المزيد من الوقت والجهد على محاوله فك تشفير البايت كود.

يؤدي استخدام مستويات متعددة من التشفير والتعتيم (Obfuscation and encryption) إلى زيادة صعوبة وتعقيد الكود. لذلك، سيستغرق المهاجم وقتًا أطول لتحليل الكود ويصعب تحليله وفهمه.

في عملية إلغاء التشفير، يقوم المستخدم بتنزيل التطبيق او البرنامج الذي تم تشفيره ورفعه على الانترنت، لن يستطيع المستخدم القيام بتشغيل التطبيق المشفر أو استخدامه بدون مفتاح فك التشفير، لذلك يجب على المستخدم ان يقوم بطلب مفتاح فك التشفير من الخادم (server). بعد قيام السيرفر بالتأكد من المستخدم (authentication process) من خلال رمز التأكيد الذي يتم ارساله للمستخدم، ومن خلال رقم التطبيق (software ID)، يقوم السيرفر بفك تشفير التطبيق عن طريق انشاء مفتاح بشكل عشوائي وله 72 بايت، بعد ذلك يتمكن المستخدم من استخدام التطبيق.

يتكون مفتاح فك التشفير من ثلاثة أجزاء، كل جزء يختص ب فك تشفير مستوى معين من مستويات التشفير، المستوى الأول وهو تشفير الكود البرمجي (name obfuscation)، يتم فك تشفيره بمفتاح طوله 16 بايت، المستوى الثاني وهو تشفير البيانات(data obfuscation) ، يتم فك تشفيره بمفتاح طوله 32بايت، تشفير البايت كود (bytecode obfuscation)، يتم فك تشفيره بمفتاح طوله 24 بايت.

من خلال مقارنه نتائج هذا النموذج (DOSP-AES)، مع نتائج برامج التشفير الأخرى، ومع نموذج (DOSP-RSA)، أثبت هذا النموذج فاعليته وقوته ضد هندسه التحليل العكسية (reverse engineering) وغيرها من أدوات تحليل البرامج والتطبيقات. باستخدام هذا النموذج فانه من الصعب على المهاجم (attacker) القيام بتحليل البرامج والتطبيقات.