

Validation of Encryption Implementation in Software
through Reverse Engineering



By

Muhammad Waqas

A thesis submitted to the faculty of Information Security
Department, Military College of Signals, National
University of Sciences and Technology, Rawalpindi in
partial fulfilment of the requirements for the degree of MS
in Information Security

November 2018

Supervisor Certificate

This is to certify that Muhammad Waqas Student of MSIS-15 Course Reg.No 00000171582 has completed his MS Thesis title "Validation of Encryption Implementation in Software through Reverse Engineering." under my supervision. I have reviewed his final thesis copy and I am satisfied with his work.

Thesis Supervisor
(Dr. Mehreen Afzal)

THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS Thesis written by **Muhammad Waqas** Registration No. **00000171582**, of **Military College of Signals** has been vetted by undersigned, found complete in all respect as per NUST Statutes/Regulations, is free of plagiarism, errors and mistakes and is accepted as partial, fulfillment for award of MS degree. It is further certified that necessary amendments as pointed out by GEC members of the scholar have been also incorporated in the said thesis.

Signature: _____

Name of Supervisor: _____

Date: _____

Signature (HOD): _____

Date: _____

Signature (Dean/Principal): _____

Date: _____

Declaration

I hereby declare that no portion of work presented in this thesis has been submitted in support of another award or qualification either at this institution or elsewhere.

Dedication

This thesis is dedicated to MY FAMILY, TEACHERS AND FRIENDS for their love, endless support and encouragement.

Acknowledgement

First of all, I would like to thank Allah Almighty for His countless blessings. After that I want to express my appreciation to my family, my friends, colleagues and the faculty for providing their enormous support to help me to do this research. Without their relentless support, assistance and prayers, I would not have reached culmination point in a peaceful state of mind.

I would like to convey my gratitude to my supervisor, Dr. Mehreen Afzal, for her supervision and constant support. Her invaluable help of constructive comments and suggestions throughout the experimental and thesis works are major contributions to the success of this research. Also, I would thank my committee members; Dr. Fawad Khan and Asst Prof Waleed Bin Shahid for their support and knowledge regarding this topic.

I am also highly thankful to my parents who always stood by my dreams and aspirations and have been a great source of inspiration for me. I would like to thank them for all their care, love and support through my times of stress and excitement.

Last but not the least, I am grateful and thankful to Military College of Signals and National University of Sciences and Technology for providing me a chance to help achieve excellence by being associated with the prestigious institutions.

Copyright Notice

- Copyright in text of this thesis rests with the student author. Copies (by any process) either in full, or of extracts, may be made only in accordance with instructions given by the author and lodged in the Library of MCS, NUST. Details may be obtained by the Librarian. This page must form part of any such copies made. Further copies (by any process) may not be made without the permission (in writing) of the author.
- The ownership of any intellectual property rights which may be described in this thesis is vested in MCS, NUST, subject to any prior agreement to the contrary, and may not be made available for use by third parties without the written permission of MCS, which will prescribe the terms and conditions of any such agreement.
- Further information on the conditions under which disclosures and exploitation may take place is available from the Library of MCS, NUST, Islamabad.

Abstract

These days data security and communication are major concern for every organization/individual and there are bundle of software available which uses cryptographic primitives to protect internal data and to secure communication. Users want to know that either the software they are using for their organization or data is really the one that the developer has promised or mentioned in the specification because not every software is open source. In case of close source software there need to be some parameters that can guarantee that the software is implemented well as stated in the specification secondly the best implementation practice is in place.

AES (Advance Encryption Standard) is FIPS approved cryptographic algorithm that is basically used for data protection. It is widely used for encryption and decryption of data in software as well as hardware. Reverse engineering is a technique used to disassemble or discover the concept or code used in applications.

In this research, Different reverse engineering techniques were explored to discover standard or otherwise implementation of AES encryption mechanism. This include correct algorithm as defined in this work, key mechanism and modes of encryption. This research also introduces a framework which can be used to achieve the goals using design heuristics and AES signatures which were formulated over the period of time. To automate and speed up the detection process a tool named “AES Crypto Scanner” was developed, which will scan the assembly file against defined parameters.

Contents

1	INTRODUCTION	1
1.1	Overview	1
1.2	Motivation and Problem Statement	2
1.3	Aims and Objectives	2
1.4	Thesis Contribution	3
1.5	Thesis Organization	4
2	LITERATURE REVIEW	5
2.1	Introduction	5
2.2	Advance Encryption Standard (AES) Overview	5
2.3	Reverse Engineering and Code Analysis	7
2.3.1	Static Analysis	8
2.3.2	Dynamic Analysis	8
2.4	Reverse Engineering and Binary Analysis Tools	9
2.5	Importance of Reverse Engineering	10
2.6	Challenges of Reverse Engineering	11
2.7	Related Research	12
2.8	Summary	17

3	Proposed Framework for Cryptographic Algorithm Detection	18
3.1	Introduction	18
3.2	Proposed Framework	18
3.2.1	Define Well Known and Standard Algorithm Implementations of AES	21
3.2.2	Compilation and Signature Extraction	21
3.2.3	Signature Extraction from Open-Source Applications	21
3.2.4	Design Heuristics	22
3.2.5	Database Creation	23
3.2.6	Perform Analysis on Real Applications	24
3.2.7	Development of Tool for Quick Detection	24
3.3	Selected Implementation of AES	25
3.4	Significance of Proposed Framework	27
3.5	Limitation of Proposed Framework	30
3.6	Summary	31
4	Implementation & Results	32
4.1	Introduction	32
4.2	Selected Applications for Analysis	32
4.3	Analysis of Real Applications	33
4.3.1	Analysis Platform	33
4.3.2	Open-Source Applications Analysis	35
4.3.3	Close-Source Applications Analysis	46
4.4	AES Crypto Scanner: An Automated Approach	57
4.4.1	Tool Specification, GUI and Main Functions	57
4.4.2	Benefits of AES Crypto Scanner w.r.t Existent Tools	59

4.5	Tools Comparison	60
4.5.1	List of Tools/Plugins	60
4.5.2	Results	60
4.6	Summary	63
5	Conclusion & Future Directions	64
5.1	Conclusion	64
5.2	Future Directions	65
5.3	Summary	66
	References	68

List of Figures

2.1	AES algorithm encryption structure	6
3.1	Schematic overview of proposed framework	20
3.2	EncFSMP lines of code	25
4.1	VMware Workstation	34
4.2	7-zip S-Boxes	36
4.3	7-zip AES Instruction Set (Encryption)	36
4.4	7-zip AES Instruction Set (Decryption)	37
4.5	7-zip Modes	37
4.6	7-zip compression info	37
4.7	AxCrypt Code view	38
4.8	AxCrypt Modes & padding	39
4.9	VeraCrypt AES Instruction Set (Encryption)	41
4.10	VeraCrypt AES Instruction Set (Decryption)	42
4.11	VeraCrypt AES Instruction Set (Flow Encryption)	42
4.12	EncFSMP S-Box	43
4.13	EncFSMP AES Instruction Set (Pattern1)	43
4.14	EncFSMP AES Instruction Set (Pattern2)	44
4.15	EncFSMP AES Instruction Set (Decryption)	44

4.16	EncFSMP AES Instruction Set (vaesenc)	44
4.17	EncFSMP AES Instruction Set (Key Generation)	45
4.18	EncFSMP Modes	45
4.19	EncFSMP GCM Mode	46
4.20	Privacy Drive library	47
4.21	Privacy Drive Crypto library	47
4.22	LibTomCrypt Library directory	48
4.23	Privacy Drive Mode (xts_decrypt.c function)	48
4.24	Privacy Drive Mode (xts_initt.c function)	48
4.25	Privacy Drive Mode (xts_encrypt.c function)	49
4.26	Privacy Drive Mode (xts_done.c function)	49
4.27	LibTomCrypt XTS Mode Functions	50
4.28	SensiGuard AES_Encrypt and AES_Encrypt_key function call	51
4.29	SensiGuard AES_Decrypt and AES_Decrypt_key function call	51
4.30	SensiGuard IGE Mode	51
4.31	Boxcryptor Code View	52
4.32	Boxcryptor Modes	53
4.33	Rohos Mini Drive S-Box	54
4.34	Rohos Mini Drive Decryption	54
4.35	Rohos Mini Drive Last Round Decryption	54
4.36	Rohos Mini Drive AES Encryption including last round	55
4.37	Rohos Mini Drive Key Generation	55
4.38	Rohos Mini Drive IMC Function	55
4.39	BestCrypt S-Box	56
4.40	BestCrypt AES Signatures	56

4.41 Private Disk S-Box	57
4.42 AES Crypto Scanner	58

List of Tables

4.2	Selected applications for analysis	33
4.3	Open-Source applications result comparison	61
4.4	Close-Source applications result comparison	62

INTRODUCTION

1.1 Overview

At the present time, the key emphases of organizations are on data/information security and the leak of sensitive information can cost them both monetarily and reputation. Although there are bunch of tools and techniques that are used by different organizations depending on their needs. Here detection of correct/standard implementation is an evolving problem in the field of information security because the source code is usually not available and getting information from low level/machine language is cumbersome task. So, without defined perimeters the task of analyst is very difficult and sometime impossible.

To protect the data, the use of cryptography in software is very popular in this insecure world. Cryptography is said to be a science which investigates strategies for ensuring data security i.e. data confidentiality, data integrity and data authenticity [1]. Here data confidentiality means that only the authorized person can view the data, Data integrity means that no unauthorized changes can be made in the data and only changes by authorized person can be made, Data authenticity means that the receiver will be able to correctly identify the sender of the data which means that the data will contain the information of real sender of the data.

If standard cryptographic algorithm is implemented then it increases the security of the tool and it can be verified by analysing source code. The standard or otherwise

implementation of cryptographic algorithms can be seen by analysing source code and can verify that either it confirms to our security requirement or not, but this is not always the case. The issue arises when there is no access to source code either lost or it is close source application, then it gets really tough to identify the cryptography and its parameters in software. The only method left is to reverse engineer the code, get the binary out of it, analyse it and confirm it against the standard implementation but it is not that easy as it sounds, because it requires labour intensive manual binary analysis and skills. Binary code does not hold many comfortable features of high-level code/programming languages like, there is no standard and defined processes or distinction between data and code, variables and functions information's are usually lost during reverse engineering and are represented using memory addresses and registers [2].

1.2 Motivation and Problem Statement

Every developer declares their software to be the best and claims to provide the best implementation that no other developer can provide, however user cannot verify their claim. Similarly, in case of encryption software there are number of software that claim to be the best and user has to believe their claim but cannot technically verify in the absence of source code. For instance, CertainSafe Digital Safety Deposit Box, Folder Lock, AxCrypt Premium are among the best encryption software available according to the recent article [3]. To verify the correctness of cryptographic implementation in a closed source software or in the absence of any clue about the source code, reverse engineering of software remains the only option.

1.3 Aims and Objectives

The prime objectives of thesis are:

1. Exploration of reverse engineering tools and techniques for efficient disassembly of crypto code in software.

2. Formulation of the framework for identification of encryption algorithm and its different parameters including encryption/decryption routines, rounds, key size and modes of encryption.
3. Experimental results on application of feasible open source tools on some open source applications to verify the methodology.
4. Generalization of the obtained results on closed source application.

1.4 Thesis Contribution

To the best of our knowledge there is no work that is specifically done on AES and on verification of AES against any standard implementation. Detection of crypto algorithms are done in earlier work but this kind of specialized work is not stated earlier.

The main contributions of this work are as follows:

- Different reverse engineering tools and techniques for disassembly of applications are explored, analysed and strengths/weakness are reported.
- A framework for the detection of AES and its parameters is formulated which will analyse the applications against standard implementation of AES.
- To check the effectiveness of the proposed framework, it was applied on few open-source applications to verify the methodology.
- To speed up the detection, a tool named “AES Crypto Scanner” is developed that identifies and highlight the AES parameters in software if found.
- The proposed framework is also generalized on close-source applications and few close-source applications were also analyzed.
- Detailed experimental results gained by the proposed framework are compiled which gives a clear comparison of this method versus other methods currently used for the similar purpose.

1.5 Thesis Organization

The thesis is structured as follows:

- **Chapter 1:** This Chapter includes introduction to topic, brief explanation of research area, problem statement, aim and objectives. It also highlighted the main contribution of this research work.
- **Chapter 2:** It contains the literature review conducted during the thesis. This chapter focuses on introduction of AES, different reverse engineering techniques, challenges and already work carried out on crypto code detection and analysis. Identification of cryptographic primitive is previously addressed and studied in different motivation and different authors have presented their ways for identification of cryptographic primitives in software. It also contains the weaknesses and strength of each work.
- **Chapter 3:** This chapter contains the proposed framework to identify the AES in binary code and the procedure of verification against the standard compliance.
- **Chapter 4:** This chapter covers the implementation detail as well as the results acquired using this technique. It also consists of comparison between other similar solutions with proposed framework to show the effectiveness of this method.
- **Chapter 5:** This chapter marks the end of the document. The conclusion and future work areas are revealed in this chapter.

LITERATURE REVIEW

2.1 Introduction

This chapter contains the AES introduction, different methods and techniques for reverse engineering and the detail analysis of the relevant work that is already been carried out over the time. In this section different techniques, tools and methods used to detect crypto code, its effectiveness and the weaknesses of each solution are highlighted.

2.2 Advance Encryption Standard (AES) Overview

Advance Encryption Standard (AES) is FIPS approved cryptographic algorithm that can be used for electronic data protection [4]. It is widely used for encryption and decryption of data in software as well as hardware. The algorithm of AES is symmetric block cipher and is capable of using block size of 128 bits, key size of 128,192, 256 bits and rounds 10, 12 and 14 respectively.

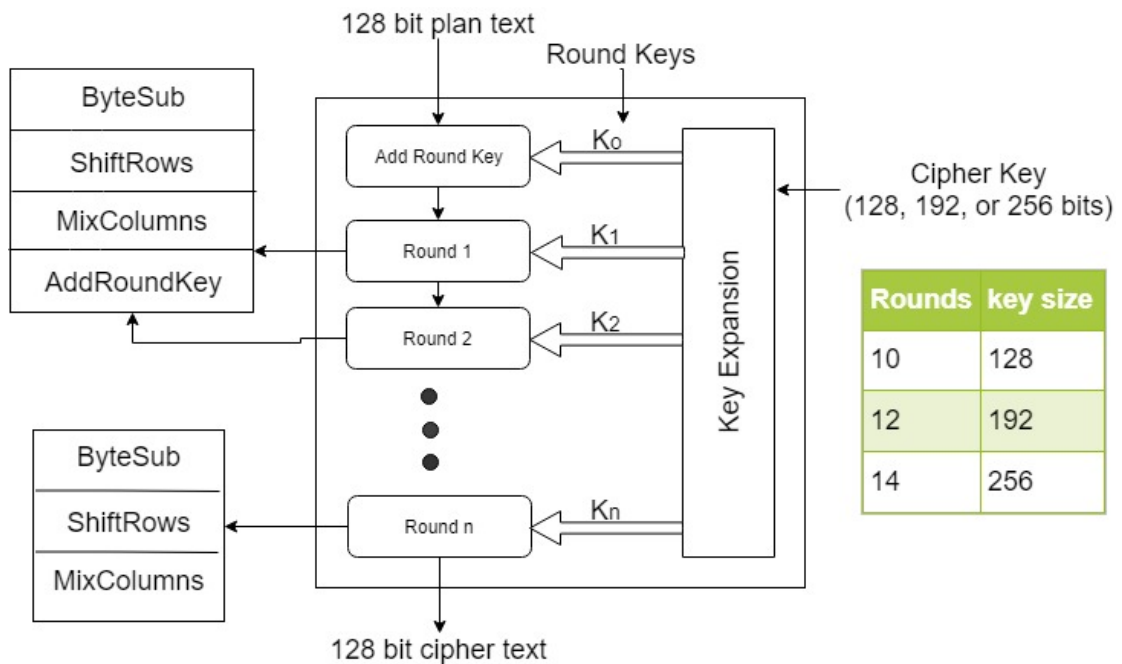


Figure 2.1: AES algorithm encryption structure

Figure 2.1 describes the encryption structure of AES algorithm. It contains rounds (10, 12 or 14), plain text, cipher text and keys. Each round except last (as shown in the figure 2.1) is consist of 4 elements i.e. ByteSub, ShiftRows, MixColumns and AddRoundKey. Initially during encryption, the AddRoundKey is added at the start of the process and that's why there is no AddRoundKey at the end/last round.

Key Expansion: The round keys like shown in the Figure 2.1 are derived from the main cipher key which uses Rijindael's key scheduler.

AddRoundKey: It is the key for every round of AES algorithm that is combined with each byte of the state using bitwise xor.

ByteSub: In ByteSub stage, each byte is replaced with another using a lookup table in a non-linear way.

ShiftRows: In this stage, each row of a state gets a certain number of steps episodically shifts.

MixColumns: At this step, the four bytes of each column are multiplied with a fixed matrix and form a new column. It is similar to matrix multiplication of a columns.

The decryption process is also the same but in reverse order.

It is an easy task to check the software source code and verify it against the standard compliance like FIPS 197 for AES. Secondly the tests are available to check the conformance to AES standard algorithm as started in FIPS 197 [5] but the problem arises when the access to source code is restricted either lost or the application is close source then there is no way to find that the code/application is as per the standard or not except reverse engineering.

2.3 Reverse Engineering and Code Analysis

Reverse engineering is a technique used to disassemble or discover the concept or code used in applications [6]. It is the method used when the access to the source code is restricted either lost or the application source code is not public. So, in the absence of source code, the only way to check the internal architecture or algorithm of code is to reverse engineer it. Reverse engineering of code is not a tough job as there are few tools that can easily give the low-level code of any application called binary code. The problem is binary code is not familiar to most of the analyst/programmers and is not easily understandable like the high-level code is, so looking for the desire functionality in binary is tedious job and without parameters sometimes not possible to understand the functionality of code.

Once the application is disassembled using any tool then the next step is the code analysis of that low-level code. There are two main methods of code analysis:

1. Static analysis
2. Dynamic analysis

So, the low-level code can then be analysed using any or all of the above-mentioned methods.

2.3.1 Static Analysis

It is widely used analysis method for low-level and high-level programming code. In Static analysis, the main focus is to locate the code sequence that possess some properties [7]. For this work, the target of static binary analysis was to locate specific parameters that can provide us desired information about the application under analysis.

There are few static analysis tools that use signatures for detection of the desired parameters [8]. That signatures are called magic constants and they can be anything like s-boxes, any variable or any function call and using those tools, the signature is then matched with x86 assembly code and if that signature is identified in that code it displays the message that the particular parameter is found in the targeted code.

2.3.2 Dynamic Analysis

Dynamic analysis is the type of analysis where the specific algorithm is revealed during runtime, means the examination of code is performed after executing it. Then after execution the different states are monitored and analysed against the reason that how the application behaves. Unless static analysis, the dynamic analysis enables analyst to see the true functionality of the application and secondly, the analyst can observe the state changes by putting his values as input. It will enable the analyst to determine the true functionality of application and will show the response, behavior and output of that application [9]. This approach is very useful in case of malware analysis because it will point out the kind of malware that hides in the code like keylogger etc., and will allow to locate the location where it stores the data/information. In addition to that it is also beneficial to analyse packed applications [8].

It is highly recommended to create a safe environment like sandbox or virtual machine before selecting this approach because if the application is infected with malware then it will damage the operating system and data. So, there should be different operating system for dynamic analysis of application.

2.4 Reverse Engineering and Binary Analysis Tools

The following tools are used to reverse engineer applications, analysis of applications against known parameters and perform binary analysis to identify cryptographic parameters. The following tools are used for reverse engineering and analysis of applications during the course of our thesis.

IDA Pro: The Interactive DisAssembler (IDA) is very rich and effective tool to disassemble, debug and decompile the software programs. It is written in C++ language and mainly operates on Microsoft Windows, Linux and Mac Operating System platforms [21].

Given software application, it can disassemble it into low level code called assembly or binary code. In addition to disassembly, it is very powerful debugger and disassembler which gives a high-level code from the binary code but it does not give the actual high-level code, it can only resolve the well-known and recognized procedures, API calls, functions, loops and switches which make code easy to understand.

OllyDbg: OllyDbg is another great tool which is used for debugging the application and it is usually used when the source code is not available. It is a 32-bit debugger named after Oleh Yuschuk, who is the developer of this tool and it is available for Microsoft Windows [22]. The main benefit of OllyDbg is that it works best for dynamic analysis means that the running state of application can be seen using this tool. Usually it is not possible to analyse the program using static analysis as it will not show the complete functionality, so it should be analysed dynamically to check the behavior.

WinDbg: WinDbg is similar tool like OllyDbg published by Microsoft but the main difference is that WinDbg is for windows and used both for kernel mode debugging as well as user mode debugging. This make it more powerful tool than OllyDbg in terms of level of access [9].

FindCrypt: FindCrypt is a plugin designed for IDA Pro and coded in python language. It provides static analysis of application and find common parameters, constants in applications [23]. The Find Crypt2 is the second version of FindCrypt which is stable and works better as compare to FindCrypt. FindCrypt2 has a large database of constants that is matched for any application under analysis. It contains signatures of almost all the well-known hash functions and crypto algorithms like: Rijndael, DES, RawDES, CAST, CAST256, Camellia, Blowfish, GOST, MARS, HAVAL, MD2, MD4, MD5, RC2, RC5, RC6, SHA1, SHA256, SHA512, Whirlpool, Zlib.

FindCrypt2 does not comes with IDA Pro but it has to be added into the plugins directory to use it for analysis. So, whenever an application is set for analysis, it searches for all the known constants from the database and match it against the one found in the application binary code.

Signsrch: Signsrch is another very useful tool to scan for the cryptographic algorithms in an application, program or process. This tool depends on cryptographic constants/signatures to identify encryption parameters/algorithms [24]. The cryptographic parameters are situated in a text file named, signsrch.sig. It can identify a huge number of compressions & encryption algorithms as well as a lot of strings & anti-debugging code.

SnD Crypto Scanner: SnD Crypto Scanner is also binary analysis plugin that is used with OllyDbg. It also contains a list of well-known cryptographic signatures, constants and hash functions. It is very powerful tool and its results are much better than FindCrypt2.

2.5 Importance of Reverse Engineering

Reverse engineering is very useful technique specially in software development and information security field. It is a way to verify specification, quality of product and known bugs.

Aside from providing a means to assess software quality, reverse engineering has additional benefits [10][11]:

Extending the life of older procedures/method: Making an improved copy of something: Involves taking an existing part and extracting the design data or other information and then creating an improved copy of it.

Malware Analysis: These days reverse engineering is frequently used to find malware in applications. Viruses and malicious code can be found using identifying and recognizing patterns or signatures in binary code.

Flaws Discovery: Reverse Engineering can be used to discover flaws and loopholes in applications. Sometimes even well-implemented and well-designed systems have loopholes that can be discovered after analysing binary code.

Flow Discovery: Using reverse engineering it can be seen that the program is behaving normally or it has some other flow which is suspicious. For example, it is possible that some application use keylogger and send the keystroke to some external server which is a critical vulnerability in application.

Open Source Code detection: If the software is intended to be used for proprietary use or for security then it is a point of concern that the software is using an open source or shareware code. Reverse Engineering make it possible to find and detect replicated code.

Improvement in Code: Reverse engineering similar apps can help in improving the existing code by adopting the advance methodology and techniques used in the similar apps.

2.6 Challenges of Reverse Engineering

Reverse engineering can be very beneficial in many cases as described in section 2.5 but it also has some challenges which make it hard to use. These challenges are the barrier for effective reverse engineering. The key challenges are:

Skill set: Reverse engineering requires a high-level of skill set in low-level language, machine code and compiler architecture. It requires a clear understanding of different fundamentals of reversing, code construct and flow analysis. When an application is reversed, it loses the high-level representation, constants and functions information. It tends to become very hard to understand and analyze component level design in deep.

Mining relevant data: In binary the execution traces become unmanageable and large which make it very challenging to trace and mine relevant data [11].

Effectiveness of single method: No single method gives the complete and accurate result i.e. static and dynamic analysis. For example, if the application is obfuscated, packed or compressed then it is usually not possible to extract all the information using static analysis of binary code so here dynamic analysis will come to aid. So, both the methods should be used in parallel to get the complete overview of the source code and to get the information of constants, libraries, loops, functions and any other coding parameter.

Programming Language difference: The change of programming language has effects on the reversed engineered code. For instance, if an application is compiled using java code than reversing that application will give us the java byte code which is way better understandable than the low-level assembly language. Similarly, if .Net is used to compile an application than the source code that is produced after disassembly is in .Net assembly language which is also understandable. But if the application is compiled using C++ or C language, it produces assembly code after being dissembled which is complex and tough to understand.

2.7 Related Research

While analysing the software security, Special attention needs to be placed on the cryptographic algorithm choice because the software security relies on it. Secondly, the algorithm should be implemented using standard values and practices. For example, MD5 function is not secure to use because practical collision can be found [25] but still it is being used in software which is not a good choice. Another example is of GnuPG

that used a smaller nonce value for the faster implementation of DSA signature scheme which is a major algorithm implementation flaw [12]. There is another purpose as well for binary program analysis and that is to check that your program is not leaking your data because it will be a serious threat to data security. It is possible that the algorithm is implemented in a way which may leak your confidential data so the program should be very carefully analysed to check for this kind of vulnerabilities because side channel attack will be applicable in this scenario.

A work on this direction was carried out by Felix Grobert, Throsten Holz and Carsten Willems [8]. The authors proposed few methods for the identification of crypto code/primitive such as algorithm identification or just keys from the binary program. They used dynamic binary analysis approach to for the detection of crypto code and extraction of keys from a malware binary program under analysis. In their work, the authors did not only rely on signatures for the identification purpose but have presented heuristics which are based on both generic characteristics of crypto code and the signatures. They used dynamic binary instrumentation framework to generate an execution trace. The system then identifies the cryptographic primitives via several heuristics and summarizes the results of the different identification methods. They evaluated six tools that were publicly available and noted that no tool was able to detect all the cryptographic primitives. They also demonstrate that our system can be used to uncover cryptographic primitives and their usage in off the-shelf and packed applications, and that it is able to extract cryptographic keys from a real-world malware sample.

Another work is proposed by Leonard et al., on the detection of crypto algorithms and for detection they used grap which is a YARA like tool which allow analyst to describe rules on binaries or textual pattern so that it can be checked on binary programs [13]. Grap is open source tool which help analyst to define detection pattern that are based on CFG (Control Flow Graph) to detect the algorithm by focusing on instructions and flow in the executable program. They created rules and pattern for AES and ChaCha20 that are based on parts of the assembly code produced by compiling popular implementa-

tions available in LibreSSL and libsodium. The requirement of this work is one should have accurately defined pattern which will be used for the detection of crypto algorithm because their approach do not rely on constants. Secondly, their technique also depends on the disassemble code quality since detection is done at assembly level that's why the pattern is sensitive to the choice of compilation option like compiler choice or optimizations etc.

The latest work in the similar direction is done by Giegory et al., on classification of cryptographic primitive keeping focus on cryptovirology [14]. Cryptovirology is described as the offensive nature of cryptography for extortion-based security threats. The authors presented a novel approach for the classification of cryptographic code in a compiled binary executable using deep learning.

Diane Duros Hosfelt present did his thesis on “Automated detection and classification of cryptographic algorithms in binary programs through machine learning”[15]. The motivation of author was to automate the process of identification so that the process can be speedy and more efficiently combat malware. The goal of his thesis was to utilize machine learning technique to detect and classify crypto code in small and single purpose program. The authors also focus on the importance of basic block detection for successful detection, where a basic block is a sequence of instructions in a given order that has a single entry and exit point. These are generated from the dynamic trace. The author elected to use Pin which is Intel's dynamic binary instrumentation (DBI) framework. It enables an analyst to examine the behavior of binary program at runtime by injecting instrumentation code. As the code executes, DBI tools analyze what actually occurs, instead of considering what might occur (as in static binary analysis). As this work only focus on small and single purpose application that's why it has few limitations. First, this method relies on dynamic analysis using Pin for feature extraction. If the code of interest is not executed during instrumentation, then it will not be analyzed and extracted. Therefore, it must be assumed that the cryptographic code is always executed. Secondly, it will not work efficiently on real world examples because they

involve multiple crypto libraries and not small or single purpose.

A work in similar direction is proposed by Felix Matenaar, Andre Wichmann, Felix Leder and Elmar Gerhards-Padilla [16]. They present the architecture of CIS, the crypto intelligent system that provides a framework which is complemented with the selection of suitable heuristics to detect crypto functions in malwares. The authors distinguished between symmetric, asymmetric, and hash algorithms because each class has their own set of properties that must be met in order to be secure.

Another contribution on the topic cryptographic primitive identification was made by Pierre et al., using Data Flow Graph Isomorphism [17]. The purpose of their research was to evaluate the security of the binary programs that involve cryptography So, the first step is to locate the point and choice of algorithm used in the binary program. For this purpose, they device a method to automatically identify cryptographic choice used in the binary program because manual analysis requires a lot of expertise and it is a cumbersome task to perform. The method consists of static analysis of binary program using Data Flow Graph Isomorphism and it targets symmetric cryptographic algorithm. The limitation of their work is that it does not address the problem of code obfuscation because the purpose is to analyse the general software not the malware. In their paper, they also present few results on sample programs and cryptographic algorithms, libraries and their implementations using several compilers.

Another work on a similar direction was presented by Joan Calvet, Jose Fernandez, Jean-Yves Marion [18]. Their work focused on obfuscated binaries which provides a solid clue that the program can be a malware. The tools usually do not work on obfuscated binaries/code because the actual implementation of the program is hidden in this case so the tools are not able to analyse the binaries. The authors have presented a tool that solves problem of obfuscated code by retrieving the input-output parameters of programs and comparing them to standard functions. They successfully identified few cryptographic functions using this tool including AES, RC4, MD5, TEA and basic

operations of RSA.

A work in a similar direction was proposed by Dongpeng et al.,. They proposed a novel approach for recognition of cryptographic function in an obfuscated binary program using the technique of bit-precise loop mapping [19]. Their approach catches the semantics of conceivable cryptographic algorithms with bit-exact representative execution in a circle. Then they performed guided fuzzing to productively coordinate Boolean equations with known reference executions. After their successful results they built a model like prototype called CryptoHunt and assessed it with an arrangement of obfuscated binary test cases, famous and best-known cryptographic libraries, and malware. Contrasted to current famous tools for this purpose, CryptoHunt is a general way to deal with distinguishing generally used cryptographic algorithms like, RC4, AES, TEA, MD5, and RSA under different types of controls and schemes for data obfuscation.

Another work was proposed by Ruoxu Zhao, Dawu Gu, Juanru Li, & Ran Yu [20]. Their paper proposes a novel automatic cryptographic data detection and analysis approach. Their approach is based on execution tracing and data pattern extraction techniques, searching the data pattern of cryptographic algorithms, and automatically extracting detected Cryptographic algorithms and input-output data. They implemented and evaluate their approach, and the result shows that their approach can detect and extract common symmetric ciphers and hash functions in most kinds of programs with accuracy, effectiveness and universality.

It can be seen that the detection and binary code analysis is not just a favorite area of research for malware analyst but also for the those who care about data security. For this purpose, different cryptographic parameters identifications techniques and solutions are proposed by different authors. For instance, the work [8] uses dynamic binary analysis of binary code using both signature and heuristics analysis method for packed applications while [14] [15] uses deep learning and machine learning to detect crypto code.

The motivation of the authors was also to provide a tool which can add in the detection process as mentioned in [13] and many others has targeted obfuscated binaries [18] [19] to retrieve the required data. The pattern analysis [20] or control flow graph analysis [13] is useful for small programs but it can not be generalized well on real applications while the signature plus heuristics [8] or even simple heuristics [16] method can be generalized on other applications and their success ratio is also better than other signature based or pattern based analysis.

2.8 Summary

This chapter has briefly described AES and reverse engineering methodology including the list of primary tools used for reverse engineer the applications and analysis. It also explained the benefits and challenges of reverse engineering. The primary purpose of this chapter was throughly research work done in similar direction, which is briefly explained in the last section.

Proposed Framework for Cryptographic Algorithm Detection

3.1 Introduction

This is the most vital chapter as it describes the proposed framework which has been recommended in this research. It will contain a detail proposed methodology, selected algorithms of AES, significance of our approach as compare to other techniques used for crypto algorithm detection and limitation of the proposed framework.

3.2 Proposed Framework

To identify cryptographic signatures, constants and known parameters one should either have the database of the signatures and constants or one should have the flow information. Both techniques have their advantages and disadvantages.

If static analysis is applied which means there should be a repository/database of constants and using that repository the analysis of the application will be performed. The advantage is this that the process of analysis became simple, easy to deploy, frequently update the database and less time consuming. But the disadvantage is this that it might not detect the valid implementation that is developed by developer himself rather than

using well known cryptographic libraries so, the analysis can fail in this case and this is the limitation of this approach.

The flow information of program can be seen using dynamic analysis means one should have to run the application on the binary analysis tool to see the flow operations, and even constants of the code. The advantage of this approach is this that it gives a high level of accuracy because it is possible that some information cannot be retrieved using static analysis as the application might be packed or obfuscated but using dynamic analysis it will show true functionality, detailed understanding of the parameters and flow of program. The disadvantage of this technique is this that getting flow information is very tough and cumbersome task secondly the flow changes with the changes in source code or even change of compiler.

This research approach relies on Static analysis of application where the focus will not be only on the detection of AES algorithm in binary code but also its parameters like S-Box, encryption & decryption routines, key length and mode of operation. To effectively deploy the proposed method, a framework is developed which will help in obtaining the desired results. The proposed framework consists of 2 main phases:

1. Planning
2. Analysis

The planning phase is the initial phase where analyst will prepare, define and refine our approach. It mainly contains the following steps:

- Define Well Known and Standard Algorithm implementations for AES
- Compilation and signature extraction
- Signature extraction from open-source applications
- Design heuristic
- Database creation

The planning phase is the prerequisite for the analysis phase. The analysis phase will use the output of phase 1 i.e. planning as an input and will analyse it against the real applications. This phase will output our required results i.e. identification of AES, Identification of standard algorithm, key length, S-Box and mode of operation. This phase contains 2 main steps:

- Perform analysis on real applications
- Development of tool for quick analysis

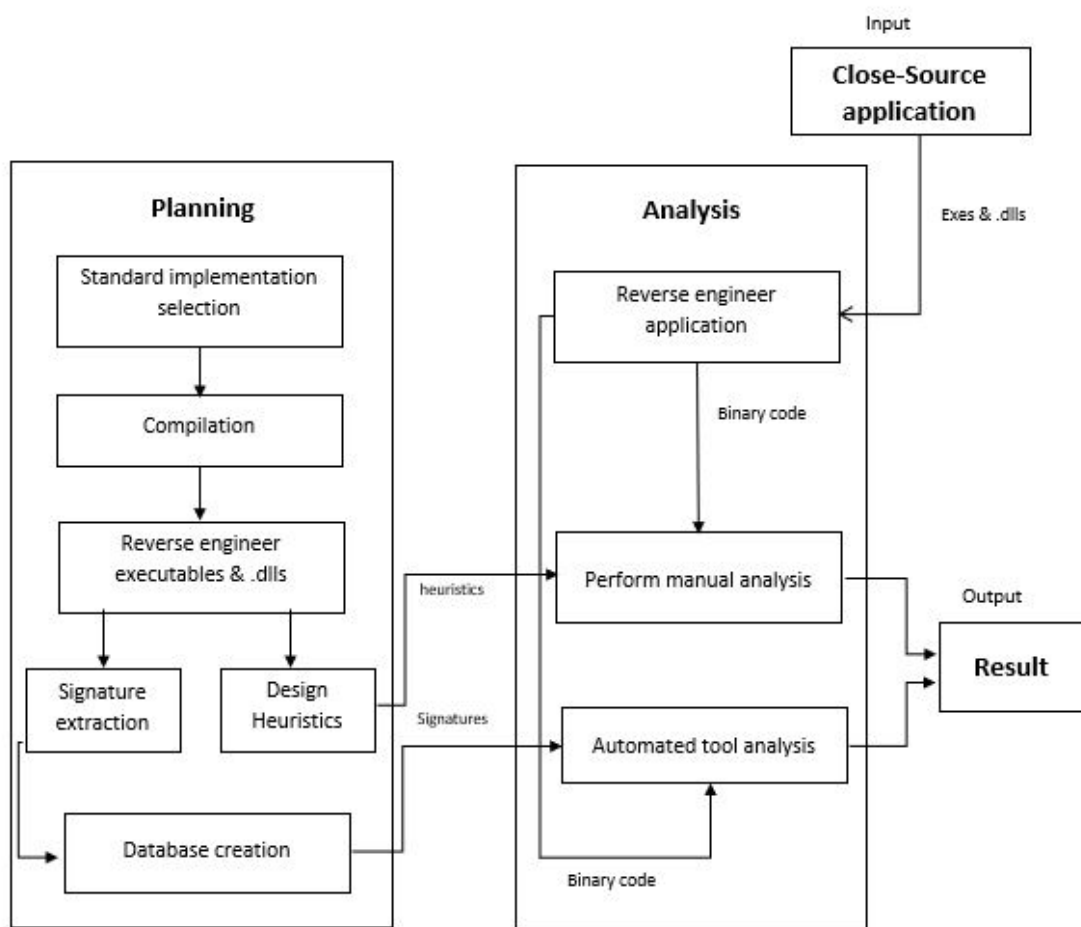


Figure 3.1: Schematic overview of proposed framework

Here is the brief overview of every step and the tasks to be performed. The findings and results of these steps will be discussed in chapter 4 when it will be implemented on real applications.

3.2.1 Define Well Known and Standard Algorithm Implementations of AES

Standard algorithms like AES needs verified implementations which means that the library is verified by FIPS or open source community. In the absence of this, verification of correctness becomes even more difficult. Therefore, in this work we have focused on finding standard libraries implementation in crypto applications.

3.2.2 Compilation and Signature Extraction

For signature extraction, it is very essential to have a compiled representation of required implementation as reversing that implementation will generate signature that will be used for detection. These days, almost every famous implementation has its compiled code, source code and even assembly code available on GitHub which can be used to define signatures for detection.

It is beneficial to compile the code on multiple compilers as it is possible that the signature might vary with the change of compiler [12]. When the compilation is done then the next step is signature extraction and that is possible when the executable and .dll files will be reversed using tool like IDA pro or OllyDbg etc. Once the application is reversed then the binary representation of the code will be available and the signatures can be extracted from that binary code and execution traces can be found.

3.2.3 Signature Extraction from Open-Source Applications

Another approach is the use of open-source applications for signatures extraction as the source code is available which can be used for verification. It will reduce the efforts of compilation phase as the code will be already compiled and that compiled application can be used for analysis.

3.2.4 Design Heuristics

This is very critical and main step of this framework which provides the heuristics data that will be used for detailed manual analysis. During this research, a number of signatures and patterns were found for AES that help in the analysis of applications and determine the type of implementation used. The developed heuristics are related to s-box detection, lookup-tables detection, number of rounds detection, mode of operations and the type of file needs to be analysed.

Analyse exe and dll files: It was seen that the critical information regarding application like libraries, functions etc. does not only resides in exe file of application but also in dll file. Usually for implementing any crypto library, developers import the standard and famous libraries which save them a lot of time for writing code for that library. If this is the case then reverse engineering the dll files will give the information about library uses. Exe file might only give the function call detail but other detail will be found in the dll file so, it is highly recommended to analyze both dll and exe file while performing analysis.

Signatures: It was found that the signatures for AES usually contain the AES keyword like in encryption it was seen that aes_encrypt and aesenc both instructions were used in different libraries. Although it is not the case always and is not the only method but this trick is also useful and it can be used initially before any detailed analysis.

S-Box and lookup-tables: Finding standard s-box and lookup-table in assembly code is relatively simple task as the values of standard can be check in the Hex view of IDA Pro or even in the assembly code. Finding s-box or lookup-tables can also assist in the detailed analysis as in the assembly code all calls to s-box or lookup-tables can be seen which will highlight the key parameters of AES. For example, the s-box memory location can be noted and then in assembly code all the calls made to that memory location can be recorded, these calls to s-box represent that the calling might be encryption function for AES which is utilizing this s-box in the function.

No. of Rounds: If AES encryption set [42] is used then the chunks of AES encryption or decryption can be used for the detection of no of rounds. For example, it can be seen that there is a specific chunk of encryption/decryption which is repeated 10 times, 12 times or 14 times. If this pattern is found it will clearly represent the total number of rounds used like if 14 times repetition found than it means that 14 rounds are used for encryption/decryption and it will ultimately give clue about the key size which is 256.

Mode of Operations: Although detecting mode of operations/encryptions can be best found using dynamic analysis but the signatures can also be searched in the binary code which is a rare case. It was found that the simple CBC mode utilized very few AES instructions as compare to XTS which uses a number of instructions. Secondly in CBC the same chunk that is repeated in rounds is small while in XTS it is large and the repetitions is also more than simple CBC mode.

3.2.5 Database Creation

Here the signatures extracted from the above steps will be stored so that it can assist in manual analysis as well as it can be used in the tool for detection purpose. In database, all those signatures will be stored which will convey some meaning like it can be signature or constant for encryption/decryption routine or some other parameter. In simple words it will be the signatures in the database which will actually be used for analysis. The database will save the information of the signature, algorithm or library name and the function of signature that will describe the signature. The signatures in the database will now be the test cases for analyst which will be used to identify AES and its parameters.

The planning phase ends here and at this moment an analyst has the parameters that will be used for the detection of AES in real application. The next phase after planning is analysis, and here analyst will use all the knowledge gained in the planning phase and will use the repository created to detect AES. The following two steps are from analysis phase.

3.2.6 Perform Analysis on Real Applications

Perform analysis on real applications is the first step of analysis phase which states that analyse and scan the real application against standard AES implementation. Here real application means the desktop software that are sold by different companies or developer and it guarantees the best and standard implementation. In this step the analysis of software will be performed to verify the implementation using the repository information created in the last step of planning phase.

The key parameters of AES to be find in the application are:

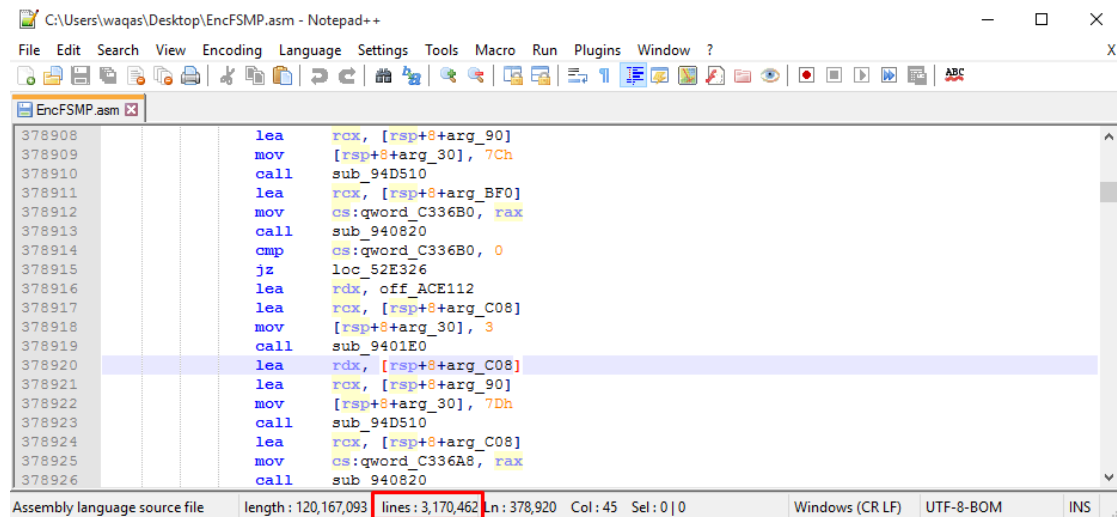
- S-Box
- Lookup-Tables
- Key Length/Size
- No. of Rounds
- Implementation used
- Mode of Operations

Here if the signatures are found in the application it means that the application is using the standard implementation specified by FIPS or the algorithm selected in this work and if the signatures are not found then it means that no standard algorithm or library is used in the code which is defined in this work. The reason could be the developer has used his own implementation of AES in source code or used the library that is not catered in this work.

3.2.7 Development of Tool for Quick Detection

As in the modern age the manual analysis is of less importance. For detection, it is recommended to deploy an automated way to analyse and detect the signatures in an application under analysis. Secondly the modern programs/applications are very complex

and they generate millions of lines of code which is not possible to analyse manually so, here the automated tool comes to aid. For example, during analysis of an application EncFSMP it was observed that the binary file contains approx. 37 lac line of code after it was reversed using IDA Pro as shown in Figure 3.2.



```
C:\Users\waqas\Desktop\EncFSMP.asm - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
EncFSMP.asm
378908 lea rcx, [rsp+8+arg_90]
378909 mov [rsp+8+arg_30], 7Ch
378910 call sub_94D510
378911 lea rcx, [rsp+8+arg_BF0]
378912 mov cs:qword_C336B0, rax
378913 call sub_940820
378914 cmp cs:qword_C336B0, 0
378915 jz loc_52E326
378916 lea rdx, off_ACE112
378917 lea rcx, [rsp+8+arg_C08]
378918 mov [rsp+8+arg_30], 3
378919 call sub_9401E0
378920 lea rdx, [rsp+8+arg_C08]
378921 lea rcx, [rsp+8+arg_90]
378922 mov [rsp+8+arg_30], 7Dh
378923 call sub_94D510
378924 lea rcx, [rsp+8+arg_C08]
378925 mov cs:qword_C336A8, rax
378926 call sub_940820
Assembly language source file length : 120,167,093 lines : 3,170,462 Ln : 378,920 Col : 45 Sel : 0 | 0 Windows (CR LF) UTF-8-BOM INS
```

Figure 3.2: EncFSMP lines of code

So manual analysis in this case will take a lot of time, efforts and the tool will give a quick analysis report.

The above proposed framework is designed for the AES but it can be used for any cryptographic algorithm given that the signatures are stored in the repository.

3.3 Selected Implementation of AES

There are number of implementation available online and they are free to use in the code but not all the implementations are correct and according to the standard. It is recommended to use the standard library as there is less chances of it being compromised or to use the library which is tested by a large open-source community and it is being used as industry standard.

It was found that there are number of libraries available for AES implementation which is either approved by FIPS or being used as industry standard. For this work the follow-

ing libraries are selected that confirms the defined standard. This is not the complete list as other implementations can also be added depending on the fact that they are tested well by open-source community or standard body.

Intel® Advanced Encryption Standard (AES) New Instructions Set: Intel has provided instruction set which can be used in source code to improve the performance and security of application [42]. It has 6 main instructions that perform the major operations of AES. Out of 6 instructions the first 4 instructions are used for performance while remaining 2 instructions are used for key expansion.

- **AESENC:** It is used for one round encryption of AES.
- **AESENCLAST:** It is used for last round encryption of AES.
- **AESDEC:** It is used for one round decryption of AES.
- **AESDECLAST:** It is used for last round decryption of AES.
- **AESIMC:** It is used for mixing the column and performs transformation.
- **AESKEYGENASSIST:** It assists in the generation of AES round keys.

OpenSSL AES: OpenSSL is basically a toolkit that is used for SSL and TLS protocols but it is also famous for its use in cryptographic libraries [43]. It is freely available for developers to use in commercial or non-commercial applications. The implementation of OpenSSL's AES uses table lookups which reduces the risk of successful cache timing attack [44].

Bouncy Castle: Bouncy Castle is a package that contains the cryptographic implementation. It is a lightweight cryptographic API for C# and Java which contains a number of cryptographic libraries including AES. It is a standard cryptographic library which is approved by FIPS 140-2 as it meets level 1 requirements of FIPS 140-2 standard overall Level 1 requirements [28].

LibTomCrypt: LibTomCrypt is very comprehensive cryptographic toolkit which enables developers a huge set of Cryptographic algorithms and it contains support for block ciphers, different chaining modes, hash functions and pseudo-random generator [35]. It is free library and its code is publicly available on GitHub [34]. Although it is not FIPS recognized but a large number of open-source community analyses the code and its bugs are frequently fixed as found.

Gladman AES: Brian Gladman's also developed a code in C/C++ languages for efficient use in software. It is also freely available on GitHub repository for both commercial and non-commercial purposes [45].

3.4 Significance of Proposed Framework

The purpose of the proposed framework is to contribute effectively in the improvement of already existing solutions for Standard AES algorithm detection in applications. The existing available solutions are generalized for a list of cryptographic algorithms and there is no single solution or tool available which only checks AES and its parameters. The disadvantage of generalized method is that the in-depth analysis cannot be performed for any specific algorithm as the only purpose of those methods are only detection. Keeping this in mind, a framework for AES has developed in this work which will overcome the deficiencies of already existence methods and tools. The proposed framework will add the following benefits to the already deployed methods and tools:

Identification of standard implementation: The existence solutions are for the purpose of detecting crypto algorithms in applications but there is no solution which checks the application against standard implementation. It is very important to check application against standard compliance as it will give user the sense of surety that the implementation is bug free. There are many implementations freely available for AES but not all the implementations are compliance with FIPS secondly any other implementation which is widely used and tested is also a valid implementation as it is universally used

and tested against vulnerabilities and bugs. So, it is suggested to use the implementation compliance with standard or the implementation which is universally followed.

Dedicated approach for AES: The existence research focused on the detection of crypto algorithms and not only AES but all the well-known Crypto algorithms. The benefit of this approach is this that all the algorithms detection can be performed under the one umbrella but the constraint of this approach is in-depth analysis cannot be performed and secondly there are a lot of false negatives. As the proposed framework is dedicated for the detection of AES so it overcomes the constraints of existence approach by only having the heuristics related to AES algorithm in database which nearly remove false negatives.

In-depth analysis: As the proposed framework is designed dedicatedly for only one algorithm i.e. AES then it makes it easy to perform in-depth analysis, design and generalized a heuristic that work on nearly all the applications under analysis. As it is a dedicated approach so, it will detect and identify maximum parameters of AES like Standard S-Box, key size, no of rounds, and mode of operations used.

The major benefit of in-depth analysis is not only the identification of key size, rounds, mode of operations or s-box used but also the implementation used. It is very important to know which AES implementation is used in the application as it will be very beneficial in the case when some AES implementation might encounter any bug. So, if in future any implementation of AES found vulnerable to any attack, the knowledge of implementation used will come to aid as analyst would be in a position to check if our application is using the same implementation or not and if the same implementation is in place then analyst should stop the use of that application and replace it with some other standard application.

Easy to follow and scalable: The proposed framework is very simple in terms of understanding and it can be easily followed. It has well defined phases and steps which makes it easy to understand and follow. Secondly it requires one-time effort as once the analyst has performed phase 1 completely then it can be used for all the applications

unless some other standard implementation found.

The proposed framework is also scalable for other cryptographic algorithms and the same method can be used and deployed for the detection of other algorithms such as DES etc. Although the same work will be repeated for the other algorithm and new database will be created for that algorithm.

Automated approach: When it comes to the detection and identification of some cryptographic algorithms in binary code then manual detection is out of question. The detection part should be handled by some automated tool and further analysis then can be performed manually if required. In this work, a tool named AES Crypto Scanner was also developed which identify and locate the signatures found in the binary code and if further analysis is required then the information displayed on the output screen can be used as a base line for further manual analysis.

Comprehensive output/result: The output/result acquired using this framework is very detailed against any application under analysis as compare to other tools/methods available for the same purpose. Secondly the tool developed in this work does not only output the statement that “AES is detected” but also display the detail result for other parameters like:

- Standard S-Box detected (if used)
- Encryption Routine (if found)
- Decryption Routine (if found)
- Key mechanism (if found)
- Implementation used
- Mode of operations (if found)

The above information provide a complete insight of an application and it became easy to rate any real application.

3.5 Limitation of Proposed Framework

Although the proposed framework has the above significance over existent work but there are some limitations which keeps it away from being 100% accurate. These limitations are actually inherent from the type of analysis method used in this research work i.e. static analysis. The limitation of the proposed framework are as follows:

No dynamic analysis: The proposed framework only used static analysis for the detection of cryptographic parameters. Although this approach is very common but it cannot give an analyst a complete picture of the algorithm or flow. Dynamic analysis gives more information about the code and flow which gives a better insight for analysis.

Obfuscation not handled: If the application is compressed, obfuscated or packed then this technique will not work as the signatures will be hidden and cannot be found. This is the main disadvantage of this approach means only relying on static analysis.

Real applications are complex: Real applications are complex means they are multi-purpose so, sometimes it gets really tough to identify the different parameters or encryption/decryption routines as there is no coherence in the binary code. The same implementation of AES in two applications might give two different binary pattern because the applications are usually multi-purpose. Although the signatures will be found that will ensure that the same implementation is used in both applications.

Mode of operations: This is the limitation of choice of method used in this work i.e. static analysis and using static analysis the mode of operations cannot always be guaranteed. In static analysis it is possible to find signatures for mode used even the flow of AES instructions can also give hint about the mode used but this method is not successful for all the applications. It was found that using Intel AES Instruction set [42], the simple CBC mode utilized very few AES instructions as compare to XTS which uses a number of instructions. Secondly in CBC the same chunk that is repeated in rounds is small while in XTS it is large and the repetitions is also more than simple

CBC mode but the issue arose when there are multiple modes used like if CBC and CTR used then it gets complex to detect the mode used in assembly code.

3.6 Summary

In this chapter a framework has been suggested which contains multiple phases and steps. The framework shows that how the detection and analysis of low-level code that is acquired after reverse engineering of an application will be performed. This framework uses both signatures detection and heuristic data to detect and analyse the applications. In addition to the framework, few popular implementations are also suggested for AES along with significance and limitation of proposed framework.

Implementation & Results

4.1 Introduction

This chapter is related to the implementation of proposed framework, analysis and results. It will contain the information of applications selected for analysis both open-source and close-source, description of our automated tool, detailed analysis results and different tools comparison with AES Crypto Scanner.

4.2 Selected Applications for Analysis

To check the effectiveness of proposed method a number of tools were analysed over a period of time. Total of 11 applications were analysed in which 5 of them are open-source and 6 of them are close-source.

Open-Source Tools	Close-Source Tools
7-Zip	Privacy Drive
AxCrypt	Boxcryptor
EncFSMP	Rohos Mini Drive
DiskCryptor	Private Disk
VeraCrypt	SensiGuard
	BestCrypt

Table 4.2: Selected applications for analysis

The detailed analysis and results are discussed in the section [4.3](#).

4.3 Analysis of Real Applications

The purpose of this work is to design a framework that can effectively contribute in the analysis of AES algorithm in real application. Here the term real applications are used for those applications that are being used by users or organizations for various purposes like data encryption etc. These applications can be either free or paid and open-source or closed-source. Analysing open-source application has a benefit over close-source that it does not need to be reverse engineer as the source code is publicly available but in case of close-source application it has to be reverse engineered so that it can be analysed.

This section will provide the detail of the analysis that were carried out over the period of this research. The analysis is divided into two sections, one for open-source software and second for close-source software.

4.3.1 Analysis Platform

To perform analysis on real applications it was required to have at least 2 dedicated machines that can be used. The purpose of using dedicated machine is this that it is possible that few applications might behave differently than expected as it might be in-

ected with some malware. So, to be on a safer end it is always advise to use separate machine for this kind of analysis.

VMware Workstation Pro was used to separate the work and operating system. The complete process starting from downloading tool to final analysis was carried out in separate virtual machines which were installed on a VMware Workstation. The VMware Workstation was installed on windows 10 and it contained 2 operating systems Windows 10 (64 bit) and Windows 7 (32 bit) as shown:

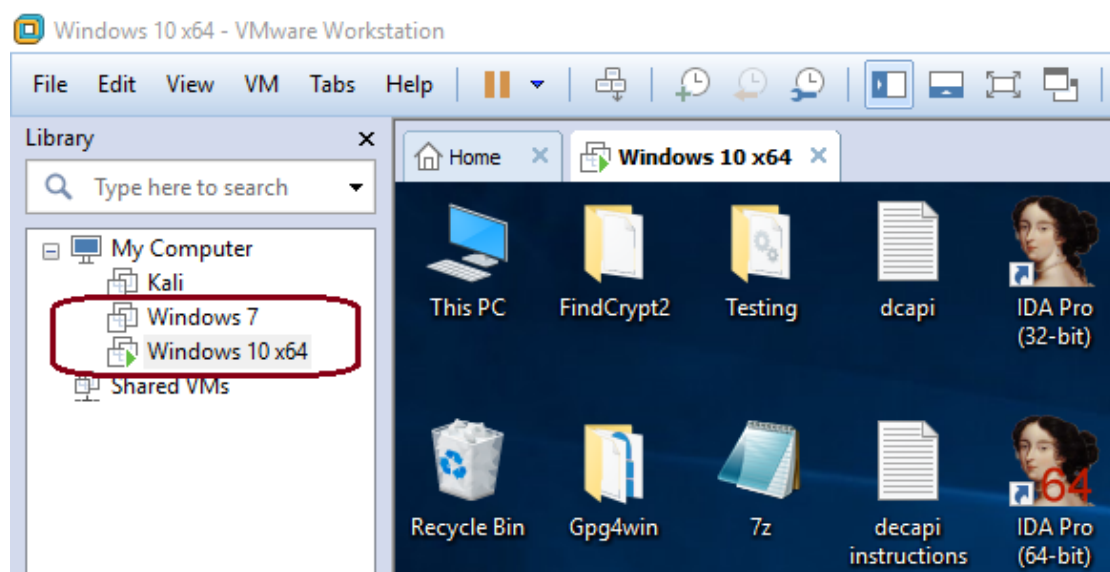


Figure 4.1: VMware Workstation

Each machine contains the following applications and tools:

- All 11 applications under analysis
- IDA Pro
- OllyDbg
- WinDbg
- FindCrypt (version 1 and version 2)
- Signsrch
- SND Crypto Scanner

- And, AES Crypto Scanner

4.3.2 Open-Source Applications Analysis

In this work 5 open-source applications were selected for analysis. Those applications were analyzed using our proposed framework and the results are then verified from the source code to check the accuracy of our method.

The detailed findings were obtained from the heuristic's analysis of applications after reverse engineering them on IDA Pro tool.

7-Zip

7-Zip is a free tool which is mainly used for compression and it is open source [26]. It is a powerful tool with number of supported formats. It was downloaded from the official website, installed on the operating system and after installation its exes and dll files were analysed using IDA Pro.

Findings: It was observed that 7-Zip contain the standard AES S-Box which was found in Hex view of 7z.dll file.

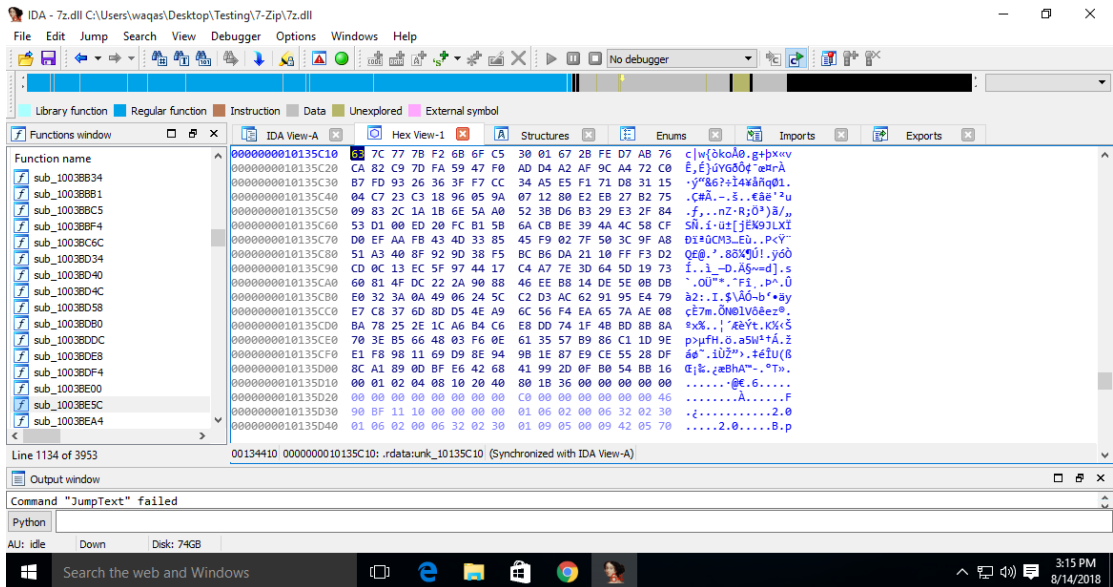


Figure 4.2: 7-zip S-Boxes

It can be seen that the S-Box is found at the memory address 10135C10. Other than this it was also found that 7-Zip is using AES instruction set which is a standard AES implementation.

AES encryption routine found: Standard AES encryption routine found in the code.



Figure 4.3: 7-zip AES Instruction Set (Encryption)

AES decryption routine found: Standard AES decryption routine found in the code.

sub_10038B34	.text:000000001011B971	sub_1011B910	aesdec xmm0, xmm7
sub_10038B81	.text:000000001011B976	sub_1011B910	aesdec xmm1, xmm7
sub_10038BC5	.text:000000001011B97B	sub_1011B910	aesdec xmm2, xmm7
sub_10038BF4	.text:000000001011B980	sub_1011B910	aesdec xmm3, xmm7
sub_10038C6C	.text:000000001011B98A	sub_1011B910	aesdec xmm0, xmm7
sub_10038D34	.text:000000001011B98F	sub_1011B910	aesdec xmm1, xmm7
sub_10038D40	.text:000000001011B994	sub_1011B910	aesdec xmm2, xmm7
sub_10038D4C	.text:000000001011B999	sub_1011B910	aesdec xmm3, xmm7
sub_10038D58	.text:000000001011B9A4	sub_1011B910	aesdec xmm0, xmm7
sub_10038D80	.text:000000001011B9A9	sub_1011B910	aesdec xmm1, xmm7
sub_10038DDC	.text:000000001011B9AE	sub_1011B910	aesdec xmm2, xmm7
sub_10038DE8	.text:000000001011B9B3	sub_1011B910	aesdec xmm3, xmm7
sub_10038DF4	.text:000000001011B9C2	sub_1011B910	aesdeclast xmm0, xmm7
sub_10038E00	.text:000000001011B9C7	sub_1011B910	aesdeclast xmm1, xmm7
sub_10038E5C	.text:000000001011B9CC	sub_1011B910	aesdeclast xmm2, xmm7
sub_10038EA4	.text:000000001011B9D1	sub_1011B910	aesdeclast xmm3, xmm7

Figure 4.4: 7-zip AES Instruction Set (Decryption)

Mode of encryption: It can be seen that CBC mode is used in the application. Although this kind of signature cannot guarantee the use of CBC but as the software code is publicly available so it was confirmed either CBC is used or not and the results was the same as found.

```

.rdata:0000000010135730 dq offset aAes256cbc ; "AES256CBC"
.rdata:0000000010135738 db 1
.rdata:0000000010135739 db 0
.rdata:000000001013573A db 0
.rdata:000000001013573B db 0
.rdata:000000001013573C db 1
.rdata:000000001013573D db 0
.rdata:000000001013573E db 0
.rdata:000000001013573F db 0
.rdata:0000000010135740 aAes256cbc db 'AES256CBC',0 ; DATA XREF: .rdata:0000000010135730fo
.rdata:000000001013574A align 10h
.rdata:0000000010135750 stru_10135750 FuncInfo_V1 <19930520h, 1, rva stru_1016982C, 0, 0, 3, \
.rdata:0000000010135750 ; DATA XREF: .rdata:0000000010169828fo

```

Figure 4.5: 7-zip Modes

Compression information: The analysis hinted that 7-zip is using pk-256, pk-192 and pk-128 with some other compressions as well and it was confirmed from the source code as well.

```

.rdata:0000000010132340 aPkaes256 db 'pkAES-256',0 ; DATA XREF: .rdata:00000000101322B8fo
.rdata:000000001013234A align 10h
.rdata:0000000010132350 aPkaes192 db 'pkAES-192',0 ; DATA XREF: .rdata:00000000101322A8fo
.rdata:000000001013235A align 20h
.rdata:0000000010132360 aPkaes128 db 'pkAES-128',0 ; DATA XREF: .rdata:0000000010132298fo
.rdata:000000001013236A align 10h

```

Figure 4.6: 7-zip compression info

AxCrypt

AxCrypt is an open-source software used for encryption. Its specification states that it provides strong encryption by using 128/256-bit AES encryption [29]. It was down-

loaded from the official website, installed on the operating system and after installation its exes and dll files were analysed using IDA Pro disassembler.

Its output is in Microsoft.Net assembly where AES implementation and other implementations can easily be found. This .Net is a portable reference library that is used for the compilation of .Net language source code [27].

Findings: AxCrypt uses Bouncy Castle API that includes AES implementation. Here the real challenge is to analyse the running state of application as the assembly code will show all the available implementation used in the library.

AES Implementation: AES implementation can be seen along with other algorithms.

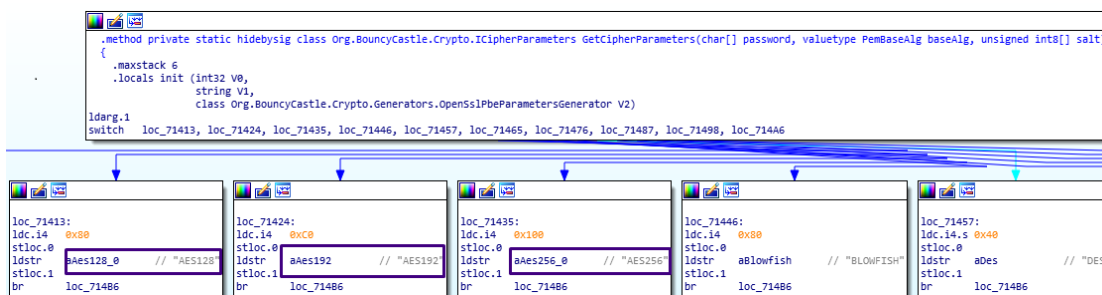


Figure 4.7: AxCrypt Code view

It can also be seen that multiple modes of encryption were found in the assembly code of AxCrypt application like ECB with pkcs7 padding, CBC with pkcs7 padding, OFB with no padding and CFB with no padding.

Address	Function	Instruction
seg000:71A47	Org.BouncyCastle.Security....	ldstr aAesCbcPkcs7pad // "AES/CBC/PKCS7PADDING"
seg000:71A56	Org.BouncyCastle.Security....	ldsfld class Org.BouncyCastle.Asn1.DerObjectIdentifier Org.Bouncy...
seg000:71A60	Org.BouncyCastle.Security....	ldstr aAesCbcPkcs7pad // "AES/CBC/PKCS7PADDING"
seg000:71A6F	Org.BouncyCastle.Security....	ldsfld class Org.BouncyCastle.Asn1.DerObjectIdentifier Org.Bouncy...
seg000:71A79	Org.BouncyCastle.Security....	ldstr aAesCbcPkcs7pad // "AES/CBC/PKCS7PADDING"
seg000:71A88	Org.BouncyCastle.Security....	ldsfld class Org.BouncyCastle.Asn1.DerObjectIdentifier Org.Bouncy...
seg000:71A92	Org.BouncyCastle.Security....	ldstr aAesOfbNopaddin // "AES/OFB/NOPADDING"
seg000:71AA1	Org.BouncyCastle.Security....	ldsfld class Org.BouncyCastle.Asn1.DerObjectIdentifier Org.Bouncy...
seg000:71AAB	Org.BouncyCastle.Security....	ldstr aAesOfbNopaddin // "AES/OFB/NOPADDING"
seg000:71ABA	Org.BouncyCastle.Security....	ldsfld class Org.BouncyCastle.Asn1.DerObjectIdentifier Org.Bouncy...
seg000:71AC4	Org.BouncyCastle.Security....	ldstr aAesOfbNopaddin // "AES/OFB/NOPADDING"
seg000:71AD3	Org.BouncyCastle.Security....	ldsfld class Org.BouncyCastle.Asn1.DerObjectIdentifier Org.Bouncy...
seg000:71ADD	Org.BouncyCastle.Security....	ldstr aAesCfbNopaddin // "AES/CFB/NOPADDING"
seg000:71AEC	Org.BouncyCastle.Security....	ldsfld class Org.BouncyCastle.Asn1.DerObjectIdentifier Org.Bouncy...
seg000:71AF6	Org.BouncyCastle.Security....	ldstr aAesCfbNopaddin // "AES/CFB/NOPADDING"
seg000:71B05	Org.BouncyCastle.Security....	ldsfld class Org.BouncyCastle.Asn1.DerObjectIdentifier Org.Bouncy...
seg000:71B0F	Org.BouncyCastle.Security....	ldstr aAesCfbNopaddin // "AES/CFB/NOPADDING"

Figure 4.8: AxCrypt Modes & padding

Here the problem is this that bouncy crypto is a rich API which contain a number of modes like shown in the Figure 4.8. It can only be seen through dynamic analysis that which mode is being used with AES.

DiskCryptor

DiskCryptor is an open-source solution for full disk encryption. It supports AES-256, Serpent and Twofish algorithms [30].

Findings: After reverse engineering using IDA Pro, it was confirmed that DiskCryptor is using AES and also AES instruction set which is a standard instruction set approved by FIPS.

Crypto Code: The following crypto code and pattern was found in the assembly code:

```

aesenc xmm4, xmmword ptr [r11+10h]
aesenc xmm4, xmmword ptr [r11+20h]
aesenc xmm4, xmmword ptr [r11+30h]
aesenc xmm4, xmmword ptr [r11+40h]
aesenc xmm4, xmmword ptr [r11+50h]
aesenc xmm4, xmmword ptr [r11+60h]

```

```
aesenc xmm4, xmmword ptr [r11+70h]
aesenc xmm4, xmmword ptr [r11+80h]
aesenc xmm4, xmmword ptr [r11+90h]
aesenc xmm4, xmmword ptr [r11+0A0h]
aesenc xmm4, xmmword ptr [r11+0B0h]
aesenc xmm4, xmmword ptr [r11+0C0h]
aesenc xmm4, xmmword ptr [r11+0D0h]
aesenclast xmm4, xmmword ptr [r11+0E0h]
```

It can be seen that how the above instructions are written in the assembly code. It gives a clear clue that the application is AES with 14 rounds. Similar was the case for the below instructions:

```
aesenc xmm0, xmm8
aesenc xmm1, xmm8
aesenc xmm2, xmm8
aesenc xmm3, xmm8
```

This chunk of instructions was also repeated like the above and same number of times. It also ended with aesenclast instruction and same repetition was found in decryption mechanism.

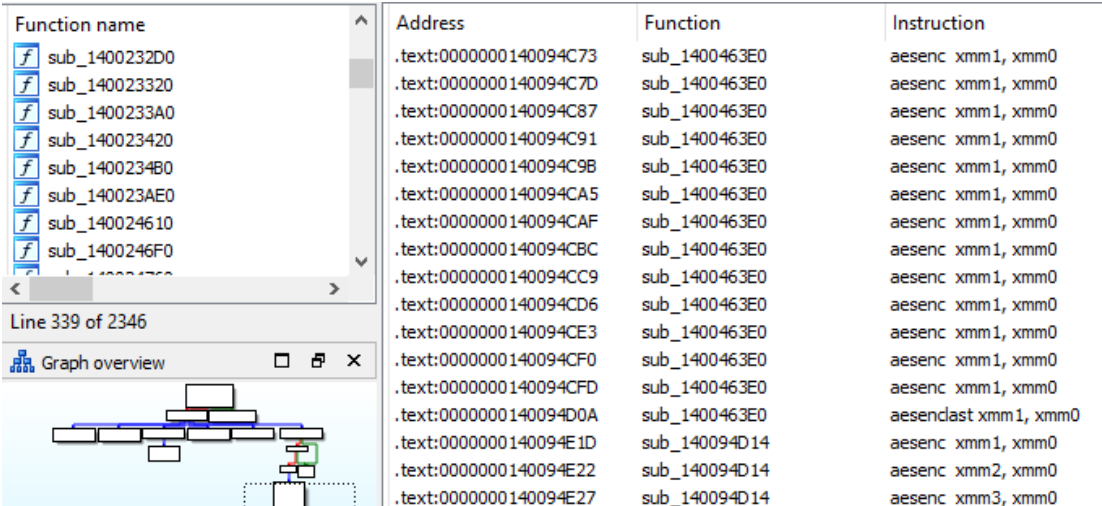
From the code it can easily be seen that the crypto code uses AES instruction set which contains encryption, decryption and last rounds routines. It can also be guessed from the code (by counting the blocks) that the number of rounds is 14 which means it uses 256-bit AES encryption. The IDA graph view also represents the flow of code which shows the flow of instructions.

VeraCrypt

VeraCrypt is an improved version of TrueCrypt which fixes the vulnerabilities of TrueCrypt and provides more security by implementation AES. It is free software and its code is open-source. It is used for full disk encryption [31].

Findings: VeraCrypt software was installed on windows 10 and its exe was analysed. It was seen that VeraCrypt was using AES instruction set and it uses 256-bit AES encryption.

Encryption routine was detected in assembly code which clearly shows number of rounds used.



Function name	Address	Function	Instruction
sub_1400232D0	.text:0000000140094C73	sub_1400463E0	aesenc xmm1, xmm0
sub_140023320	.text:0000000140094C7D	sub_1400463E0	aesenc xmm1, xmm0
sub_1400233A0	.text:0000000140094C87	sub_1400463E0	aesenc xmm1, xmm0
sub_140023420	.text:0000000140094C91	sub_1400463E0	aesenc xmm1, xmm0
sub_1400234B0	.text:0000000140094C9B	sub_1400463E0	aesenc xmm1, xmm0
sub_140023AE0	.text:0000000140094CA5	sub_1400463E0	aesenc xmm1, xmm0
sub_140024610	.text:0000000140094CAF	sub_1400463E0	aesenc xmm1, xmm0
sub_1400246F0	.text:0000000140094CBC	sub_1400463E0	aesenc xmm1, xmm0
	.text:0000000140094CC9	sub_1400463E0	aesenc xmm1, xmm0
	.text:0000000140094CD6	sub_1400463E0	aesenc xmm1, xmm0
	.text:0000000140094CE3	sub_1400463E0	aesenc xmm1, xmm0
	.text:0000000140094CF0	sub_1400463E0	aesenc xmm1, xmm0
	.text:0000000140094CFD	sub_1400463E0	aesenc xmm1, xmm0
	.text:0000000140094D0A	sub_1400463E0	aesenclast xmm1, xmm0
	.text:0000000140094E1D	sub_140094D14	aesenc xmm1, xmm0
	.text:0000000140094E22	sub_140094D14	aesenc xmm2, xmm0
	.text:0000000140094E27	sub_140094D14	aesenc xmm3, xmm0

Figure 4.9: VeraCrypt AES Instruction Set (Encryption)

Decryption routine was also detected in assembly code.

Function name	Address	Function	Instruction
sub_1400232D0	.text:000000014009440C	sub_1400465E0	aesdec xmm1, xmm0
sub_140023320	.text:0000000140094416	sub_1400465E0	aesdec xmm1, xmm0
sub_1400233A0	.text:0000000140094420	sub_1400465E0	aesdec xmm1, xmm0
sub_140023420	.text:000000014009442A	sub_1400465E0	aesdec xmm1, xmm0
sub_1400234B0	.text:0000000140094434	sub_1400465E0	aesdec xmm1, xmm0
sub_140023AE0	.text:000000014009443E	sub_1400465E0	aesdec xmm1, xmm0
sub_140024610	.text:0000000140094448	sub_1400465E0	aesdec xmm1, xmm0
sub_1400246F0	.text:0000000140094455	sub_1400465E0	aesdec xmm1, xmm0
	.text:0000000140094462	sub_1400465E0	aesdec xmm1, xmm0
	.text:000000014009446F	sub_1400465E0	aesdec xmm1, xmm0
	.text:000000014009447C	sub_1400465E0	aesdec xmm1, xmm0
	.text:0000000140094489	sub_1400465E0	aesdec xmm1, xmm0
	.text:0000000140094496	sub_1400465E0	aesdec xmm1, xmm0
	.text:00000001400944A3	sub_1400465E0	aesdeclast xmm1, xmm0
	.text:00000001400945B6	sub_1400944AD	aesdec xmm1, xmm0
	.text:00000001400945B8	sub_1400944AD	aesdec xmm2, xmm0
	.text:00000001400945C0	sub_1400944AD	aesdec xmm3, xmm0

Figure 4.10: VeraCrypt AES Instruction Set (Decryption)

In assembly code it was also seen that VeraCrypt is also using VAESENCLAST instruction which is used to perform last round flow encryption. It uses 3 operands as compare to simple aesenc instruction which uses 2 instruction. This instruction uses 2 different round keys which increase the security of last round.

Function name	Address	Function	Instruction
sub_1400232D0	.text:00000001400B2BDF	sub_1400B2880	vaesendclast xmm0, xmm0, xmm12
sub_140023320	.text:00000001400B2BE4	sub_1400B2880	vaesendclast xmm7, xmm7, xmm12
sub_1400233A0	.text:00000001400B2BE9	sub_1400B2880	vaesendclast xmm1, xmm1, xmm12
sub_140023420	.text:00000001400B2BEE	sub_1400B2880	vaesendclast xmm4, xmm4, xmm12
sub_1400234B0	.text:00000001400B2BF3	sub_1400B2880	vaesendclast xmm2, xmm2, xmm12
sub_140023AE0	.text:00000001400B2BF8	sub_1400B2880	vaesendclast xmm5, xmm5, xmm12
sub_140024610	.text:00000001400B2BF0	sub_1400B2880	vaesendclast xmm3, xmm3, xmm12
sub_1400246F0	.text:00000001400B2C02	sub_1400B2880	vaesendclast xmm6, xmm6, xmm12
	.text:00000001400B2F67	sub_1400B2880	vaesendclast xmm4, xmm4, xmm12
	.text:00000001400B2F6C	sub_1400B2880	vaesendclast xmm3, xmm3, xmm12
	.text:00000001400B2F71	sub_1400B2880	vaesendclast xmm5, xmm5, xmm12
	.text:00000001400B2F76	sub_1400B2880	vaesendclast xmm0, xmm0, xmm12
	.text:00000001400B2F7B	sub_1400B2880	vaesendclast xmm6, xmm6, xmm12
	.text:00000001400B2F80	sub_1400B2880	vaesendclast xmm1, xmm1, xmm12
	.text:00000001400B2F85	sub_1400B2880	vaesendclast xmm7, xmm7, xmm12
	.text:00000001400B2F8A	sub_1400B2880	vaesendclast xmm2, xmm2, xmm12
	.text:00000001400B32EF	sub_1400B2880	vaesendclast xmm0, xmm0, xmm12

Figure 4.11: VeraCrypt AES Instruction Set (Flow Encryption)

EncFSMP

EncFSMP creates an encrypted folder where user places and store their sensitive data. Users can create and edit the password of EncFSMP folder. It is free and open-source tool which has a user-friendly interface [32].

Findings: EncFSMP was downloaded from official site [32], installed on windows 10 and its exe was analyzed using IDA Pro tool.

S-Box: Found standard S-box in the hex dump.

0000000000737900	63 7C 77 7B F2 6B 6F C5 30 01 67 2B FE D7 AB 76	c w{òkoÅ0.g+þx«v
0000000000737910	CA 82 C9 7D FA 59 47 F0 AD D4 A2 AF 9C A4 72 C0	Ê,É}úYGð0¢~æPrÀ
0000000000737920	B7 FD 93 26 36 3F F7 CC 34 A5 E5 F1 71 D8 31 15	·ý"&6?÷Ï4¥ãñq01.
0000000000737930	04 C7 23 C3 18 96 05 9A 07 12 80 E2 EB 27 B2 75	.Ç#Ã.-.š..€âë'²u
0000000000737940	09 83 2C 1A 1B 6E 5A A0 52 3B D6 B3 29 E3 2F 84	.f,..nZ·R;0³)ã/,
0000000000737950	53 D1 00 ED 20 FC B1 5B 6A CB BE 39 4A 4C 58 CF	SÑ.í·ü±[jË%9JLXİ
0000000000737960	D0 EF AA FB 43 4D 33 85 45 F9 02 7F 50 3C 9F A8	Øİ±0CM3...Eù..P<ÿ"
0000000000737970	51 A3 40 8F 92 9D 38 F5 BC B6 DA 21 10 FF F3 D2	QË@.'·8ð%ŷÚ!.ÿóò
0000000000737980	CD 0C 13 EC 5F 97 44 17 C4 A7 7E 3D 64 5D 19 73	Í..ì_-D.ÄŞ~=d].s
0000000000737990	60 81 4F DC 22 2A 90 88 46 EE B8 14 DE 5E 0B DB	`·0Û"*·^Fî.·P^·Û
00000000007379A0	E0 32 3A 0A 49 06 24 5C C2 D3 AC 62 91 95 E4 79	à2:.I.\$\Â0~b'·äy
00000000007379B0	E7 C8 37 6D 8D D5 4E A9 6C 56 F4 EA 65 7A AE 08	çÈ7m.ÖN01Vðëez®.
00000000007379C0	BA 78 25 2E 1C A6 B4 C6 E8 DD 74 1F 4B BD 8B 8A	ex%.. 'Æèÿt.K%<Š
00000000007379D0	70 3E B5 66 48 03 F6 0E 61 35 57 B9 86 C1 1D 9E	p>µfH.ð.a5W+Á.ž
00000000007379E0	E1 F8 98 11 69 D9 8E 94 9B 1E 87 E9 CE 55 28 DF	áø~.iùž"'.#éİU(B
00000000007379F0	8C A1 89 0D BF E6 42 68 41 99 2D 0F B0 54 BB 16	Ëj§.çæBhA™-.°T".

Figure 4.12: EncFSMP S-Box

Encryption & decryption routine: The assembly code of EncFSMP was very detailed containing more than 30 lac lines of codes. It contains multiple encryption and decryption routines as compare to the other software and its pattern cannot be guessed. The reason can be it uses multiple key length with multiple modes of encryption which makes the pattern complex, but it was found that it uses AES instruction set in the source code.

sub_401010	.text:000000000073AE1A	aesenc xmm2, xmm1
sub_401060	.text:000000000073AE2B	aesenclast xmm2, xmm1
sub_401180	.text:000000000073AE6A	aesdec xmm2, xmm1
start	.text:000000000073AE7B	aesdecbast xmm2, xmm1
sub_4014E0	.text:000000000073AEC0	aesenc xmm2, xmm1
sub_401500	.text:000000000073AEC5	aesenc xmm3, xmm1
sub_401510	.text:000000000073AED2	aesenc xmm2, xmm0
sub_4018E0	.text:000000000073AED7	aesenc xmm3, xmm0
sub_401E30	.text:000000000073AEE3	aesenc xmm2, xmm1
sub_4025B0	.text:000000000073AEE8	aesenc xmm3, xmm1
sub_4026E0	.text:000000000073AEEF	aesenclast xmm2, xmm0
sub_4026F0	.text:000000000073AEF2	aesenclast xmm3, xmm0

Figure 4.13: EncFSMP AES Instruction Set (Pattern1)

This is one of the patterns for encryption but there are number of patterns exists in EncFSMP assembly code for encryption like the one shown in Figure 4.14, which suggests it can be encryption routine for 256-bit encryption call.

sub_401010	.text:0000000007654C3	sub_7647C0	aesenc xmm2, xmm0
sub_401060	.text:00000000076551F	sub_7647C0	aesenc xmm2, xmm1
sub_401180	.text:00000000076555F	sub_7647C0	aesenc xmm2, xmm0
start	.text:0000000007655CA	sub_7647C0	aesenc xmm2, xmm1
sub_4014E0	.text:000000000765615	sub_7647C0	aesenc xmm2, xmm0
sub_401500	.text:00000000076564F	sub_7647C0	aesenc xmm2, xmm1
sub_401510	.text:0000000007656D4	sub_7647C0	aesenc xmm2, xmm0
sub_4018E0	.text:000000000765714	sub_7647C0	aesenc xmm2, xmm1
sub_401E30	.text:000000000765761	sub_7647C0	aesenc xmm2, xmm0
sub_4025B0	.text:0000000007657B2	sub_7647C0	aesenc xmm2, xmm1
sub_4026E0	.text:0000000007657BC	sub_7647C0	aesenc xmm2, xmm0
sub_4026F0	.text:0000000007657C8	sub_7647C0	aesenc xmm2, xmm1
sub_402700	.text:0000000007657D2	sub_7647C0	aesenc xmm2, xmm0
sub_402710	.text:0000000007657D7	sub_7647C0	aesendast xmm2, xmm1

Figure 4.14: EncFSMP AES Instruction Set (Pattern2)

Similarly, there are number of patterns exists for decryption in the assembly code:

nullsub_5	.text:00000000073AEF2	sub_73AEAO	aesendast xmm3, xmm0
sub_401010	.text:00000000073AF20	sub_73AF00	aesdec xmm2, xmm1
sub_401060	.text:00000000073AF25	sub_73AF00	aesdec xmm3, xmm1
sub_401180	.text:00000000073AF32	sub_73AF00	aesdec xmm2, xmm0
start	.text:00000000073AF37	sub_73AF00	aesdec xmm3, xmm0
sub_4014E0	.text:00000000073AF43	sub_73AF00	aesdec xmm2, xmm1
sub_401500	.text:00000000073AF48	sub_73AF00	aesdec xmm3, xmm1
sub_401510	.text:00000000073AF4D	sub_73AF00	aesdecclast xmm2, xmm0
sub_4018E0	.text:00000000073AF52	sub_73AF00	aesdecclast xmm3, xmm0

Figure 4.15: EncFSMP AES Instruction Set (Decryption)

In assembly code, vaesenc instruction was also found like shown in Figure 4.16:

Function name	Address	Function	Instruction
nullsub_5	.text:000000000745B81	sub_745B00	vaesenc xmm9, xmm9, xmm2
sub_401010	.text:000000000745B97	sub_745B00	vaesenc xmm10, xmm10, xmm2
sub_401060	.text:000000000745BAB	sub_745B00	vaesenc xmm11, xmm11, xmm2
sub_401180	.text:000000000745BB9	sub_745B00	vaesenc xmm12, xmm12, xmm2
start	.text:000000000745BCC	sub_745B00	vaesenc xmm13, xmm13, xmm2
sub_4014E0	.text:000000000745BE4	sub_745B00	vaesenc xmm14, xmm14, xmm2
sub_401500	.text:000000000745BF3	sub_745B00	vaesenc xmm9, xmm9, xmm15
sub_401510	.text:000000000745C0A	sub_745B00	vaesenc xmm10, xmm10, xmm15
sub_4018E0	.text:000000000745C15	sub_745B00	vaesenc xmm11, xmm11, xmm15
sub_401E30	.text:000000000745C20	sub_745B00	vaesenc xmm12, xmm12, xmm15
sub_4025B0	.text:000000000745C2A	sub_745B00	vaesenc xmm13, xmm13, xmm15
sub_4026E0	.text:000000000745C3A	sub_745B00	vaesenc xmm14, xmm14, xmm15
sub_4026F0	.text:000000000745C4E	sub_745B00	vaesenc xmm9, xmm9, xmm15
sub_402700	.text:000000000745C5D	sub_745B00	vaesenc xmm10, xmm10, xmm15
sub_402710	.text:000000000745C6C	sub_745B00	vaesenc xmm11, xmm11, xmm15
sub_402770	.text:000000000745C7D	sub_745B00	vaesenc xmm12, xmm12, xmm15
	.text:000000000745C82	sub_745B00	vaesenc xmm13, xmm13, xmm15

Figure 4.16: EncFSMP AES Instruction Set (vaesenc)

Key Generation: Key generation instruction was also found in the assembly code:

Function name	Address	Function	Instruction
nulsub_5	.text:00000000073E386	sub_73E330	aesimc xmm0, xmm0
sub_401010	.text:00000000073E40E	sub_73E3A0	aeskeygenassist xmm1, xmm0, 1
sub_401060	.text:00000000073E419	sub_73E3A0	aeskeygenassist xmm1, xmm0, 2
sub_401180	.text:00000000073E424	sub_73E3A0	aeskeygenassist xmm1, xmm0, 4
start	.text:00000000073E42F	sub_73E3A0	aeskeygenassist xmm1, xmm0, 8
sub_4014E0	.text:00000000073E43A	sub_73E3A0	aeskeygenassist xmm1, xmm0, 10h
sub_401500	.text:00000000073E445	sub_73E3A0	aeskeygenassist xmm1, xmm0, 20h
sub_401510	.text:00000000073E450	sub_73E3A0	aeskeygenassist xmm1, xmm0, 40h
sub_4018E0	.text:00000000073E45B	sub_73E3A0	aeskeygenassist xmm1, xmm0, 80h
sub_401E30	.text:00000000073E466	sub_73E3A0	aeskeygenassist xmm1, xmm0, 1Bh
sub_4025B0	.text:00000000073E471	sub_73E3A0	aeskeygenassist xmm1, xmm0, 36h

Figure 4.17: EncFSMP AES Instruction Set (Key Generation)

Mode of Encryption: A number of modes were detected in the assembly code but due to the complexity of code it cannot be guaranteed that which mode is actually being executed during run-time.

```

.text:0000000006A07D1      call     sub_6E8E40
.text:0000000006A07D6      lea     rcx, aRC2Cbc ; "RC2-CBC"
.text:0000000006A07DD      mov     cs:qword_C58FD0, rax
.text:0000000006A07E4      call   sub_6E8E40
.text:0000000006A07E9      lea     rcx, aAes128Cbc ; "AES-128-CBC"
.text:0000000006A07F0      mov     cs:qword_C58FD8, rax
.text:0000000006A07F7      mov     cs:qword_C58FE0, 0
.text:0000000006A0802      call   sub_6E8E40
.text:0000000006A0807      lea     rcx, aAes256Cbc ; "AES-256-CBC"
.text:0000000006A080E      mov     cs:qword_C58FF0, rax
.text:0000000006A0815      call   sub_6E8E40
.text:0000000006A081A      lea     rcx, aCamellia128Cbc ; "CAMELLIA-128-CBC"
.text:0000000006A0821      mov     cs:qword_C58FF8, rax
.text:0000000006A0828      call   sub_6E8E40
.text:0000000006A082D      lea     rcx, aCamellia256Cbc ; "CAMELLIA-256-CBC"
.text:0000000006A0834      mov     cs:qword_C59000, rax
.text:0000000006A083B      call   sub_6E8E40
.text:0000000006A0840      lea     rcx, aGost89Cnt ; "gost89-cnt"
.text:0000000006A0847      mov     cs:qword_C59008, rax
.text:0000000006A084E      call   sub_6E8E40
.text:0000000006A0853      lea     rcx, aSeedCbc ; "SEED-CBC"
.text:0000000006A085A      mov     cs:qword_C59010, rax
.text:0000000006A0861      call   sub_6E8E40
.text:0000000006A0866      lea     rcx, aIdAes128Gcm ; "id-aes128-GCM"
0029FBF0 0000000006A07F0: sub_6A07A0+50 (Synchronized with Hex View-1)

```

Figure 4.18: EncFSMP Modes

In the following Figure 4.19 it can be seen that the entry for GCM mode is found in the assembly code:

```

IDA View-A x Occurrences of: aes x Hex View-1 x Structures x Enums x
.text:00000000006A2012      cmp     eax, 400h
.text:00000000006A2017      lea    r12, aGost89256 ; "GOST89(256)"
.text:00000000006A201E      jz     short loc_6A2054
.text:00000000006A2020      jbe   loc_6A22A0
.text:00000000006A2026      cmp     eax, 1000h
.text:00000000006A2028      lea    r12, aAesgcm128 ; "AESGCM(128)"
.text:00000000006A2032      jz     short loc_6A2054
.text:00000000006A2034      cmp     eax, 2000h
.text:00000000006A2039      lea    r12, aAesgcm256 ; "AESGCM(256)"
.text:00000000006A2040      jz     short loc_6A2054
.text:00000000006A2042      cmp     eax, 800h
.text:00000000006A2047      lea    r12, aSeed128 ; "SEED(128)"
.text:00000000006A204E      jnz   loc_6A22EE

```

Figure 4.19: EncFSMP GCM Mode

4.3.3 Close-Source Applications Analysis

This section contains the analysis detail of close-source applications that were analysed using our proposed framework. The real target of this research was to analyse the close-source applications against their specifications as every application announces that it is developed using the standard library.

In this section all the findings of 6 close-source applications are mentioned including figures from IDA Pro that were found during the analysis phase. Here the results are based on the findings acquired from the analysis of open-source applications and research.

Privacy Drive

Privacy drive is disk encryption software which is used to encrypt complete disk rather than encrypting individual files and folders [33]. It has many features like hiding, locking and encryption. Its specification states that it supports standard industry encryption algorithm which supports AES 128/256-bit.

Findings: The application was downloaded [33], installed and the exe file was analyzed against AES signatures. It was found that it is not using AES instruction set but some other signatures for AES can be found in the assembly code. No standard S-box

for AES was detected but the detailed analysis revealed that the application is using lookup-tables in AES implementation.

During analysis of the assembly code, it was also found that privacy drive is using the LibTomCrypt cryptographic library which is freely available on GitHub [34]. The following Figure 4.20 provides a hint that it uses LibTomCrypt library for AES.

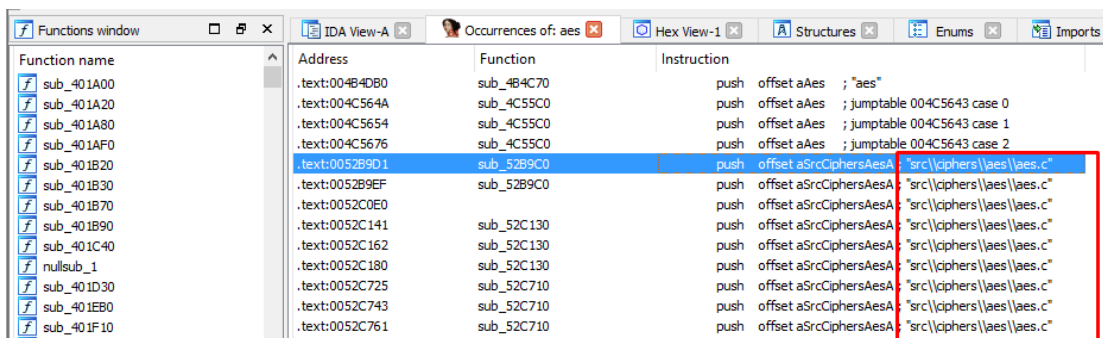


Figure 4.20: Privacy Drive library

The highlighted code shows the library location from where it was included:

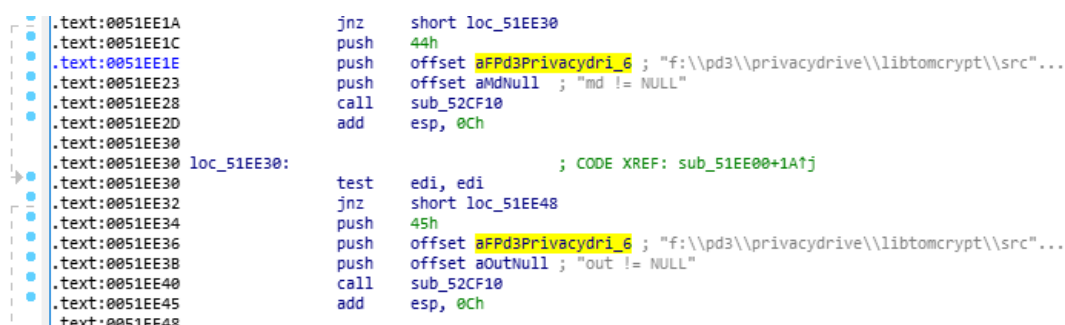


Figure 4.21: Privacy Drive Crypto library

It was found that library which is available at GitHub, also has the same directory like shown below:


```

Branch: develop ▾ libtomcrypt / src / ciphers / aes / aes.c
karel-m rename macro byte >> LTC_BYTE - related to #439
4 contributors
747 lines (675 sloc) | 19.2 KB
1 /* LibTomCrypt, modular cryptographic library -- Tom St Denis
2 *
3 * LibTomCrypt is a library that provides various cryptographic
4 * algorithms in a highly modular and flexible manner.
5 *
6 * The library is free for all purposes without any express

```

Figure 4.22: LibTomCrypt Library directory

Mode of Encryption: It uses XTS mode of encryption which can be seen in the assembly code. Secondly it uses all the functions for XTS mode given at GitHub.

```

.text:0052B744      jnz     snort_10C_52B75A
.text:0052B746      push   47h
.text:0052B748      push   offset aSrcModesXtsXts_0 ; "src\\modes\\xts\\xts_decrypt.c"
.text:0052B74D      push   offset aPtNull ; "pt != NULL"
.text:0052B752      call   sub_52CF10
.text:0052B757      add    esp, 0Ch
.text:0052B75A      loc_52B75A:                                     ; CODE XREF: sub_52B720+241j
.text:0052B75A      mov    edi, [ebp+arg_0]
.text:0052B75D      test   edi, edi
.text:0052B75F      jnz    short loc_52B775
.text:0052B761      push   48h
.text:0052B763      push   offset aSrcModesXtsXts_0 ; "src\\modes\\xts\\xts_decrypt.c"
.text:0052B768      push   offset aCtNull ; "ct != NULL"
.text:0052B76D      call   sub_52CF10
.text:0052B772      add    esp, 0Ch
.text:0052B775

```

Figure 4.23: Privacy Drive Mode (xts_decrypt.c function)

It can be seen in the Figure 4.23 that xts_decrypt.c is detected in the assembly code.

```

.text:0052B8F8      push   47h
.text:0052B8FB      push   offset aSrcModesXtsXts_1 ; "src\\modes\\xts\\xts_init.c"
.text:0052B900      push   offset aKey1Null ; "key1 != NULL"
.text:0052B905      call   sub_52CF10
.text:0052B90A      add    esp, 0Ch
.text:0052B90D      loc_52B90D:                                     ; CODE XREF: sub_52B8F0+71j
.text:0052B90D      cmp    [ebp+arg_8], 0
.text:0052B911      jnz    short loc_52B927
.text:0052B913      push   28h
.text:0052B915      push   offset aSrcModesXtsXts_1 ; "src\\modes\\xts\\xts_init.c"
.text:0052B91A      push   offset aKey2Null ; "key2 != NULL"
.text:0052B91F      call   sub_52CF10
.text:0052B924      add    esp, 0Ch

```

Figure 4.24: Privacy Drive Mode (xts_init.c function)

It can be seen in the Figure 4.24 that xts_init.c is detected in the assembly code.

```

loc_52CD87:
.text:0052CD87      jnz     short loc_52CD90
.text:0052CD89      push   4Ah
.text:0052CD8B      push   offset aSrcModesXtsXts_2 ; "src\\modes\\xts\\xts_encrypt.c"
.text:0052CD90      push   offset aPtNull ; "pt != NULL"
.text:0052CD95      call   sub_52CF10
.text:0052CD9A      add    esp, 0Ch
.text:0052CD9D      loc_52CD9D:                                     ; CODE XREF: sub_52CD60+271j
.text:0052CD9D      test   ebx, ebx
.text:0052CD9F      jnz   short loc_52CDB5
.text:0052CDA1      push   4Bh
.text:0052CDA3      push   offset aSrcModesXtsXts_2 ; "src\\modes\\xts\\xts_encrypt.c"
.text:0052CDA8      push   offset aCtNull ; "ct != NULL"
.text:0052CDAE      call   sub_52CF10
.text:0052CDB2      add    esp, 0Ch
.text:0052CDB5

```

Figure 4.25: Privacy Drive Mode (xts_encrypt.c function)

Figure 4.25 shows the signatures of xts_encrypt.c function and similarly Figure 4.26 also show another function used i.e. xts_done .

```

.text:00529290      push   eop
.text:00529291      mov    ebp, esp
.text:00529293      push   esi
.text:00529294      mov    esi, [ebp+arg_0]
.text:00529297      test   esi, esi
.text:00529299      jnz   short loc_5292AF
.text:0052929B      push   18h
.text:0052929D      push   offset aSrcModesXtsXts ; "src\\modes\\xts\\xts_done.c"
.text:005292A2      push   offset aXtsVoid0 ; "xts != ((void *)0)"
.text:005292A7      call   sub_52CF10
.text:005292AC      add    esp, 0Ch
.text:005292AF

```

Figure 4.26: Privacy Drive Mode (xts_done.c function)

It can be seen that XTS mode in LibTomCrypt also uses all these functions which gives clear hint the source code is using LibTomCrypt library and also XTS mode of encryption.

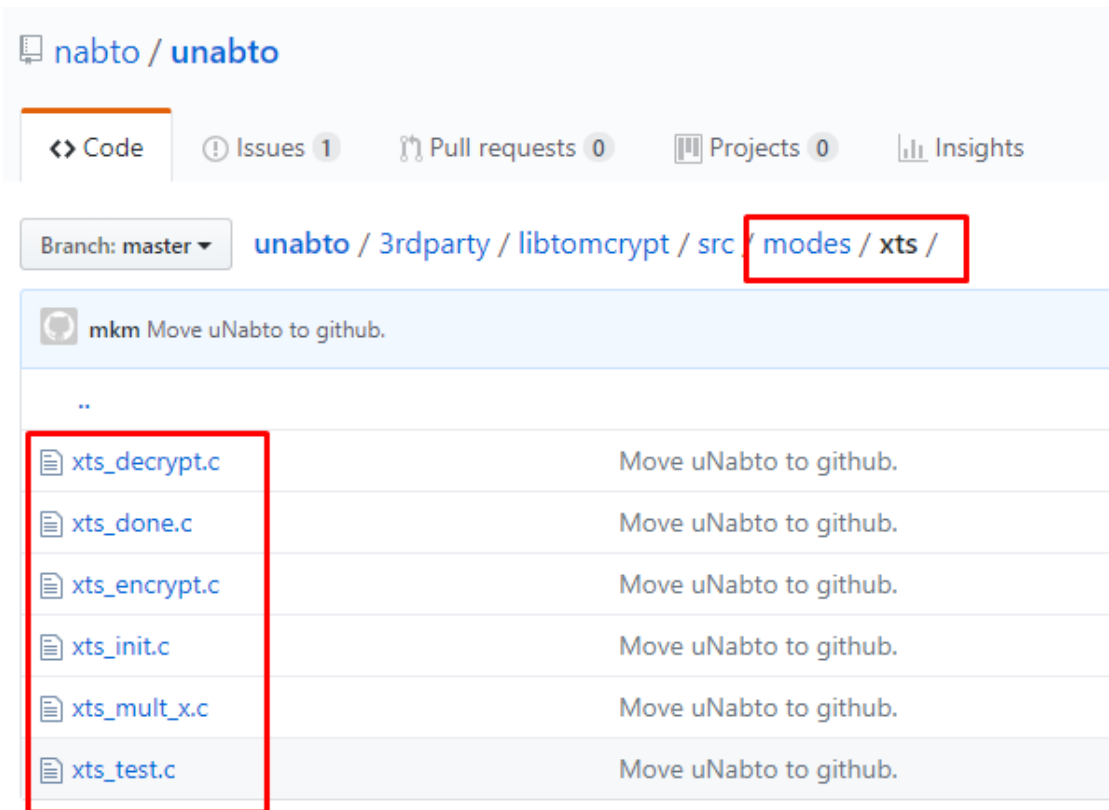


Figure 4.27: LibTomCrypt XTS Mode Functions

SensiGuard

SensiGuard is encryption software which provides strong encryption using AES algorithm. It uses 256-bit AES key which makes it a good choice to use for sensitive data [36]. It is close-source application which offers encryption and folder locking features.

Findings: It was found that there were few functions calls for AES encryption, decryption and keys. By analysing the function call signatures, it looks like that the source code is using OpenSSL Implementation of AES. Figure 4.28 shows the function calls found in the assembly code:

```

.text:0000001800EEF8      lea     rsi, [rbp+120h+var_04]
.text:0000001800EEFF      call   cs:AES_set_encrypt_key
.text:0000001800EF05      cmp     rsi, 10h
.text:0000001800EF09      jnb    short loc_1800EF3D
.text:0000001800EF0B      mov     r14d, 10h
.text:0000001800EF11      sub     r14, rbx
.text:0000001800EF14      nop    dword ptr [rax+00h]
.text:0000001800EF18      nop    dword ptr [rax+rax+00000000h]
.text:0000001800EF20      loc_1800EF20: ; CODE XREF: sub_1800ED30+20B4j
.text:0000001800EF20      lea     r8, [rbp+128h+var_178]
.text:0000001800EF24      mov     rdx, rbx
.text:0000001800EF27      mov     rcx, rbx
.text:0000001800EF2A      call   cs:AES_encrypt
.text:0000001800EF30      add     rdx, 10h
.text:0000001800EF34      lea     rax, [r14+rbx]
.text:0000001800EF38      cmp     rax, rsi
.text:0000001800EF3B      jbe    short loc_1800EF20
.text:0000001800EF3D

```

Figure 4.28: SensiGuard AES_Encrypt and AES_Encrypt_key function call

```

.text:0000001800F6A1      call   cs:AES_set_decrypt_key
.text:0000001800F6A7      lea     rax, [rsp+168h+data]
.text:0000001800F6AF      lea     rbx, [rsp+168h+data]
.text:0000001800F6B7      sub     rdi, rax
.text:0000001800F6BA      nop    word ptr [rax+rax+00h]
.text:0000001800F6C0      loc_1800F6C0: ; CODE XREF: sub_1800F5F0+ED4j
.text:0000001800F6C0      lea     r8, [rsp+168h+var_148]
.text:0000001800F6C5      mov     rdx, rbx
.text:0000001800F6C8      mov     rcx, rbx
.text:0000001800F6CB      call   cs:AES_decrypt
.text:0000001800F6D1      add     rdx, 10h
.text:0000001800F6D5      lea     rax, [rdi+rbx]
.text:0000001800F6D9      cmp     rax, 10h

```

Figure 4.29: SensiGuard AES_Decrypt and AES_Decrypt_key function call

Mode of Encryption: It was also found that the source code is using IGE cipher mode. IGE stands for Infinite Garble Extension which has the property that errors in bits are propagated indefinitely [37]. OpenSSL implemented this mode in 2006 which is mainly used for AES algorithm.

```

.text:0000001800257CE      movaps [rdp+200h+var_70], xmm0
.text:0000001800257D5      movaps [rbp+200h+var_60], xmm1
.text:0000001800257DC      call   cs:AES_ige_encrypt
.text:0000001800257E2      lea     r8, [rbp+200h+Buf2]
.text:0000001800257E9      lea     rcx, [rsp+300h+var_2AC]
.text:0000001800257EE      mov     edx, 20h
.text:0000001800257F3      call   cs:SHA256
.text:0000001800257F9      lea     rdx, [rbp+200h+Buf2] ; Buf2
.text:000000180025800      lea     rcx, [rsp+300h+Buf1] ; Buf1
.text:000000180025805      mov     r8d, 20h ; Size
.text:00000018002580B      call   memcmp

```

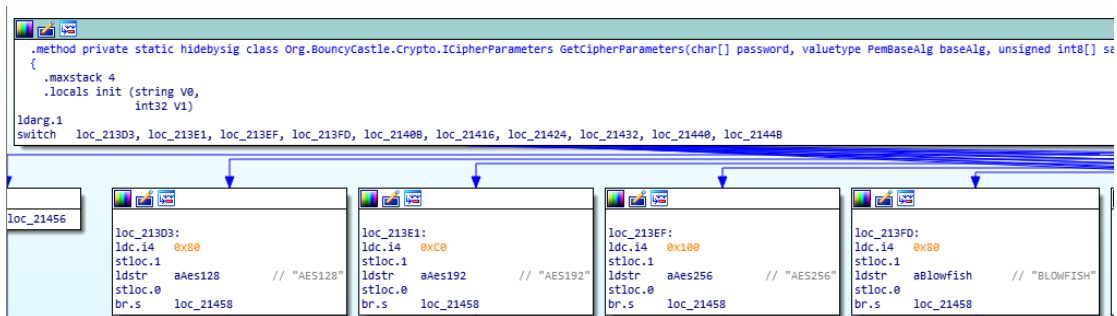
Figure 4.30: SensiGuard IGE Mode

Boxcryptor

Boxcryptor is encryption software which offer end-to-end security that primarily purpose is to provide security for cloud [38]. It is free if used for non-commercial purpose.

Findings: After reverse engineering, it was found that the application was compiled in C# language that's why after reverse engineering, it gave Microsoft.Net assembly code for analysis. As the code is in .Net assembly language so it can be clearly seen that which cryptographic library is used for encryption and decryption of file.

It was seen that the application used bouncy crypto API for encryption and decryption but it can only be seen through dynamic analysis of application that which crypto routine is being executed for encryption and decryption as static analysis will list complete cryptographic routines like AES, blowfish etc.



```
.method private static hidebysig class org.BouncyCastle.Crypto.ICipherParameters GetCipherParameters(char[] password, valuetype PemBaseAlg baseAlg, unsigned int& se
{
    .maxstack 4
    .locals init (string v0,
                 int32 v1)
ldarg.1
switch loc_21303, loc_213E1, loc_213EF, loc_213FD, loc_21408, loc_21416, loc_21424, loc_21432, loc_21448, loc_21448

loc_21456
loc_21303:
ldc.i4 0x80
stloc.1
ldstr aAes128 // "AES128"
stloc.0
br.s loc_21458

loc_213E1:
ldc.i4 0xC0
stloc.1
ldstr aAes192 // "AES192"
stloc.0
br.s loc_21458

loc_213EF:
ldc.i4 0x100
stloc.1
ldstr aAes256 // "AES256"
stloc.0
br.s loc_21458

loc_213FD:
ldc.i4 0x80
stloc.1
ldstr aBlowfish // "BLOWFISH"
stloc.0
br.s loc_21458
```

Figure 4.31: Boxcryptor Code View

Mode of Encryption: Multiple modes detected but it is not clear which mode will be executed at runtime.

Address	Function	Instruction
seg000:219A5	Org.BouncyCastle.Security.CipherUtilities::.cctor	ldstr aAesPkcs5 // "AES//PKCS5"
seg000:219AA	Org.BouncyCastle.Security.CipherUtilities::.cctor	ldstr aAesEcbPkcs7pad // "AES/ECB/PKCS7PADDING"
seg000:219B9	Org.BouncyCastle.Security.CipherUtilities::.cctor	ldstr aAesPkcs5padding // "AES//PKCS5PADDING"
seg000:219BE	Org.BouncyCastle.Security.CipherUtilities::.cctor	ldstr aAesEcbPkcs7pad // "AES/ECB/PKCS7PADDING"
seg000:219CD	Org.BouncyCastle.Security.CipherUtilities::.cctor	ldsfid class Org.BouncyCastle.Asn1.DerObjectIdentifier Org.Bouncy...
seg000:219D7	Org.BouncyCastle.Security.CipherUtilities::.cctor	ldstr aAesCbcPkcs7pad // "AES/CBC/PKCS7PADDING"
seg000:219E6	Org.BouncyCastle.Security.CipherUtilities::.cctor	ldsfid class Org.BouncyCastle.Asn1.DerObjectIdentifier Org.Bouncy...
seg000:219F0	Org.BouncyCastle.Security.CipherUtilities::.cctor	ldstr aAesCbcPkcs7pad // "AES/CBC/PKCS7PADDING"
seg000:219FF	Org.BouncyCastle.Security.CipherUtilities::.cctor	ldsfid class Org.BouncyCastle.Asn1.DerObjectIdentifier Org.Bouncy...
seg000:21A09	Org.BouncyCastle.Security.CipherUtilities::.cctor	ldstr aAesCbcPkcs7pad // "AES/CBC/PKCS7PADDING"
seg000:21A18	Org.BouncyCastle.Security.CipherUtilities::.cctor	ldsfid class Org.BouncyCastle.Asn1.DerObjectIdentifier Org.Bouncy...
seg000:21A22	Org.BouncyCastle.Security.CipherUtilities::.cctor	ldstr aAesOfbNopaddin // "AES/OFB/NOPADDING"
seg000:21A31	Org.BouncyCastle.Security.CipherUtilities::.cctor	ldsfid class Org.BouncyCastle.Asn1.DerObjectIdentifier Org.Bouncy...
seg000:21A3B	Org.BouncyCastle.Security.CipherUtilities::.cctor	ldstr aAesOfbNopaddin // "AES/OFB/NOPADDING"
seg000:21A4A	Org.BouncyCastle.Security.CipherUtilities::.cctor	ldsfid class Org.BouncyCastle.Asn1.DerObjectIdentifier Org.Bouncy...
seg000:21A54	Org.BouncyCastle.Security.CipherUtilities::.cctor	ldstr aAesOfbNopaddin // "AES/OFB/NOPADDING"
seg000:21A63	Org.BouncyCastle.Security.CipherUtilities::.cctor	ldsfid class Org.BouncyCastle.Asn1.DerObjectIdentifier Org.Bouncy...

Figure 4.32: Boxcryptor Modes

Rohos Mini Drive

Rohos Mini Drive is a desktop software primarily used for securing USB drives by creating a hiding encrypted partition which will only be accessible using correct password [39]. It has a number of features including browsers profile data encrypting, history and even skype chat and profile encryption and hiding.

Findings: It was found after reverse engineering that Rohos Mini Drive is using AES instruction set and standard s-box was also found in memory.

S-box: Standard S-box was found during the analysis of assembly code of Rohos Mini Drive application that was acquired after reverse engineering of the Rohos Mini.exe file.

```

007BF910 64 6F 6D 00 00 00 00 00 63 7C 77 7B F2 6B 6F C5 dom...c|w{ðkoÅ
007BF920 30 01 67 2B FE D7 AB 76 CA 82 C9 7D FA 59 47 F0 0.g+þx«vÊ,É}úYGð
007BF930 AD D4 A2 AF 9C A4 72 C0 B7 FD 93 26 36 3F F7 CC Ôç`æþrÀ.ý“&6?÷Ï
007BF940 34 A5 E5 F1 71 D8 31 15 04 C7 23 C3 18 96 05 9A 4×âñq01..ç#Ã.-.š
007BF950 07 12 80 E2 EB 27 B2 75 09 83 2C 1A 1B 6E 5A A0 ..€âë'²u.f,..nZ·
007BF960 52 38 D6 B3 29 E3 2F 84 53 D1 00 ED 20 FC B1 5B R;Ö³)ã/,SÑ.í.ü±[
007BF970 6A CB BE 39 4A 4C 58 CF D0 EF AA FB 43 4D 33 85 jĚX9JLXIĐi±ûCM3...
007BF980 45 F9 02 7F 50 3C 9F A8 51 A3 40 8F 92 9D 38 F5 Eù..P<ÿ`Qe@.' .8ð
007BF990 BC B6 DA 21 10 FF F3 D2 CD 0C 13 EC 5F 97 44 17 %Ÿú!.ÿóôí..i_-D.
007BF9A0 C4 A7 7E 3D 64 5D 19 73 60 81 4F DC 22 2A 90 88 Āš~=d].s`.OÜ™*.^
007BF9B0 46 EE B8 14 DE 5E 0B DB E0 32 3A 0A 49 06 24 5C Fi,.P^..Ùà2:.I.$\
007BF9C0 C2 D3 AC 62 91 95 E4 79 E7 C8 37 6D 8D D5 4E A9 ĀŌ-b'•äÿĚ7m.ŌŊ0
007BF9D0 6C 56 F4 EA 65 7A AE 08 BA 78 25 2E 1C A6 B4 C6 lVôêez@.²x%.!´Æ
007BF9E0 E8 DD 74 1F 4B 8D 8B 8A 70 3E B5 66 48 03 F6 0E èÝt.K%<Šp>µfH.ö.
007BF9F0 61 35 57 B9 86 C1 1D 9E E1 F8 98 11 69 D9 8E 94 a5W²+Á.žáø".iùž”
007BFA00 9B 1E 87 E9 CE 55 28 DF 8C A1 89 0D BF E6 42 68 >.†éÏU(BQ;š.çæBh
007BFA10 41 99 2D 0F B0 54 BB 16 52 09 6A D5 30 36 A5 38 A™-.°T»R.jŸ006¥8
007BFA20 BF 40 A3 9E 81 F3 D7 FB 7C E3 39 82 9B 2F FF 87 ç@fĚ.óxû|ã9,>/ÿ†

```

Figure 4.33: Rohos Mini Drive S-Box

Similarly, AES instruction set was also found in the assembly code. Figure 4.34 shows AES decryption instruction.

```

.text:004D5F5F      sub_4D5EF0      aesdec xmm0, xmm1
.text:004D5F6A      sub_4D5EF0      aesdec xmm0, xmm1
.text:004D5F75      sub_4D5EF0      aesdec xmm0, xmm1
.text:004D5F80      sub_4D5EF0      aesdec xmm0, xmm1

```

Figure 4.34: Rohos Mini Drive Decryption

Figure 4.35 shows the instruction for last round decryption of AES.

```

.text:004D5F9F      sub_4D5EF0      aesdedast xmm0, xmm1
.text:004D5FAA      sub_4D5EF0      aesdedast xmm0, xmm1
.text:004D5FB5      sub_4D5EF0      aesdedast xmm0, xmm1
.text:004D5FC0      sub_4D5EF0      aesdedast xmm0, xmm1

```

Figure 4.35: Rohos Mini Drive Last Round Decryption

Similarly, AES encryption for all the rounds can be seen in Figure 4.36.

.text:004D60AF	sub_4D6040	aesenc xmm0, xmm1
.text:004D60BA	sub_4D6040	aesenc xmm0, xmm1
.text:004D60C5	sub_4D6040	aesenc xmm0, xmm1
.text:004D60D0	sub_4D6040	aesenc xmm0, xmm1
.text:004D60EF	sub_4D6040	aesendast xmm0, xmm1
.text:004D60FA	sub_4D6040	aesendast xmm0, xmm1
.text:004D6105	sub_4D6040	aesendast xmm0, xmm1
.text:004D6110	sub_4D6040	aesendast xmm0, xmm1

Figure 4.36: Rohos Mini Drive AES Encryption including last round

The instruction for key generation can also be seen in the assembly code:

.text:004D7A50	sub_4D79B0	aeskeygenassist xmm0, xmm1, 0
.text:004D7AFD	sub_4D79B0	aeskeygenassist xmm0, xmm1, 0
.text:004D7B3F	sub_4D79B0	aeskeygenassist xmm0, xmm1, 0

Figure 4.37: Rohos Mini Drive Key Generation

The instruction for column mixing and transformation can also be seen in the Figure 4.38:

.text:004D7BE0	sub_4D79B0	aesimc xmm0, xmmword ptr [edx+eax*4]
.text:004D7BE6	sub_4D79B0	aesimc xmm1, xmmword ptr [edx+ecx*4]
.text:004D7BFE	sub_4D79B0	aesimc xmm0, xmmword ptr [edx+ecx*4]

Figure 4.38: Rohos Mini Drive IMC Function

BestCrypt

BestCrypt is disk and file encryption software which is available for Windows, OS X and Linux platform. It can be used for both volume and files encryption [40].

Finding: BestCrypt was downloaded from official website [40]. It was then installed on a drive and all the exes, dll were analysed against AES signatures.

S-Box: Standard s-box was detected during analysis of assembly code as shown in the Figure 4.39:

sub_40100F	00511ED0	83 A2 B2 8D 9E F5 8B EA	63 7C 77 7B F2 6B 6F C5	f4? z0&e w{0koA
sub_40102B	00511EE0	30 01 67 2B FE D7 AB 76	CA 82 C9 7D FA 59 47 F0	0.g+px«vÉ,É}úY68
sub_401082	00511EF0	AD D4 A2 AF 9C A4 72 C0	B7 FD 93 26 36 3F F7 CC	0t~œRrA-y"86?+I
sub_40109E	00511F00	34 A5 E5 F1 71 D8 31 15	04 C7 23 C3 18 96 05 9A	4#âñq01..C#Ã.-.š
sub_4010B9	00511F10	07 12 80 E2 EB 27 B2 75	09 83 2C 1A 1B 6E 5A A0	..éâë'2u.f,..nZ.
sub_4010E3	00511F20	52 38 D6 B3 29 E3 2F 84	53 D1 00 ED 20 FC B1 5B	R;0³)ã/,„SN.i.ü±[
sub_40113F	00511F30	6A CB BE 39 4A 4C 58 CF	D0 EF AA FB 43 4D 33 85	jÊK9JLXIĐi#0CM3...
sub_401159	00511F40	45 F9 02 7F 50 3C 9F A8	51 A3 40 8F 92 9D 38 F5	Eü..P<ÿ~QE@.'80
sub_40117C	00511F50	BC B6 DA 21 10 FF F3 D2	CD 0C 13 EC 5F 97 44 17	%Ų! .yôŲ. .i.-D.
CPaneContainerGC::~CPaneContainer	00511F60	C4 A7 7E 3D 64 5D 19 73	60 81 4F DC 22 2A 90 88	Äš~=d] .s`.OU~*.'^
sub_4011E9	00511F70	46 EE B8 14 DE 5E 0B DB	E0 32 3A 0A 49 06 24 5C	Fi .p^ .Ûà2:..I.\$\
sub_401206	00511F80	C2 D3 AC 62 91 95 E4 79	E7 C8 37 6D 8D D5 4E A9	Ä0-b'•âyçE7m.0N0
sub_401222	00511F90	6C 56 F4 EA 65 7A AE 08	BA 78 25 2E 1C A6 B4 C6	lVôëez0.šx%.!'Æ
sub_401270	00511FA0	E8 DD 74 1F 4B 8D 8B 8A	70 3E B5 66 48 03 F6 0E	èÿt.K%<Šp>µfH.0.
sub_401294	00511FB0	61 35 57 B9 86 C1 1D 9E	E1 F8 98 11 69 D9 8E 94	a5W³+Á.žá0~.iÜž"
sub_401294	00511FC0	9B 1E 87 E9 CE 55 28 DF	8C A1 89 0D BF E6 42 68	> .#éÍU(BE;š.æBh
sub_401294	00511FD0	41 99 2D 0F B0 54 BB 16	52 09 6A D5 30 36 A5 38	A"- .°T>.R.j006#8

Figure 4.39: BestCrypt S-Box

Other Signatures: There are also few signatures found in the binary code but it did not give any meaningful information about the algorithm used. Similarly, there are many strings found the hex dump but the problem remains the same as it did not give any information about the algorithm used.

sub_40102B	.rdata:004E3518	aBenchmarkSuppo db 'Benchmark_SupportAESHardwareAcceleration',0
sub_401082	.rdata:004E3544	aBenchmarkSetae db 'Benchmark_SetAESHardwareAcceleration',0
sub_40109E	.rdata:004E4600	db ' -a <name>','9',' - name of encryption algorithm, default is...
sub_4010B9	.rdata:004F009C	aAes256Cfb db 'aes256-CFB',0
sub_4010E3	.rdata:004F00B0	aAes256Ofb db 'aes256-OFB',0
sub_40113F	.rdata:004F00C4	aAes256Cbc db 'aes256-CBC',0
sub_401159	.rdata:004F00D8	aAes256Ecb db 'aes256-ECB',0
sub_40117C	.rdata:004F00EC	aAes192Cfb db 'aes192-CFB',0
CPaneContainerGC::~CPaneContainer	.rdata:004F0100	aAes192Ofb db 'aes192-OFB',0
sub_4011E9	.rdata:004F0114	aAes192Cbc db 'aes192-CBC',0
sub_401206	.rdata:004F0128	aAes192Ecb db 'aes192-ECB',0
sub_401222	.rdata:004F013C	aAes128Cfb db 'aes128-CFB',0
sub_401270	.rdata:004F014C	aAes128Ofb db 'aes128-OFB',0
sub_401294	.rdata:004F015C	aAes128Cbc db 'aes128-CBC',0
sub_401294	.rdata:004F016C	aAes128Ecb db 'aes128-ECB',0
sub_4012BE	.rdata:004F121C	aidRsaesOaep db 'id-RSAES-OAEP',0

Figure 4.40: BestCrypt AES Signatures

Private Disk

Private disk is also the similar software which is used for encrypting sensitive data and it is only available for Windows platform [41]. Its specifications states that it uses 256-bit AES data encryption. It also states that it is FIPS approved and the 256-bit encryption algorithm was adopted by the NIST.

Findings: Except for only detecting the standard s-box, our analysis did not find any other signature which can show the algorithm used in the source code. As there is certificate found from NIST on the official website [41] that it fulfils the requirement of

FIPS pub 197, it means that it is possible that they have deployed their own implementation of AES. As the source code is not available so it cannot be justified that which algorithm is used in source code and secondly, the signature for existing algorithm cannot developed for existing algorithm as the access to source code of that algorithm is restricted.

```

1002E240 C0 F7 70 07 53 7C 77 7B F2 6B 6F C5 30 01 67 2B À+p.č|w{òkoÀ0.g+
1002E250 FE D7 AB 76 CA 82 C9 7D FA 59 47 F0 AD D4 A2 AF px«vĚ,É}úYGð0đ~
1002E260 9C A4 72 C0 B7 FD 93 26 36 3F F7 CC 34 A5 E5 F1 œHrÀ.ý“&6?÷İ4¥ãñ
1002E270 71 D8 31 15 04 C7 23 C3 18 96 05 9A 07 12 80 E2 q01..Ç#Ã.-.š..€ã
1002E280 EB 27 B2 75 09 83 2C 1A 1B 6E 5A A0 52 3B D6 B3 ë'²u.f,..nZ·R;Ö³
1002E290 29 E3 2F 84 53 D1 00 ED 20 FC B1 5B 6A CB BE 39 )ã/„SÑ.í·ū±[jĚK9
1002E2A0 4A 4C 58 CF D0 EF AA FB 43 4D 33 85 45 F9 02 7F JLXİĐi=ûCM3...Eù..
1002E2B0 50 3C 9F A8 51 A3 40 8F 92 9D 38 F5 BC B6 DA 21 P<Ÿ~QE@.' .8ð%ŊÚ!
1002E2C0 10 FF F3 D2 CD 0C 13 EC 5F 97 44 17 C4 A7 7E 3D .ÿó0İ..i -D.ĂŞ~=
1002E2D0 64 5D 19 73 60 81 4F DC 22 2A 90 88 46 EE B8 14 d].s`.OŪ"*.^Fî,.
1002E2E0 DE 5E 0B DB E0 32 3A 0A 49 06 24 5C C2 D3 AC 62 P^..Ūà2:.I.$\ĂÓ-b
1002E2F0 91 95 E4 79 E7 C8 37 6D 8D D5 4E A9 6C 56 F4 EA '•äÿçĚ7m.ŌN01Vôê
1002E300 65 7A AE 08 BA 78 25 2E 1C A6 B4 C6 E8 DD 74 1F ez0..²x%..|'ÆèŸt.
1002E310 4B BD 8B 8A 70 3E B5 66 48 03 F6 0E 61 35 57 B9 KŹ<Šp>µfH.ö.a5W²
1002E320 86 C1 1D 9E E1 F8 98 11 69 D9 8E 94 9B 1E 87 E9 †Á.žáø~.iŪž”>.†é
1002E330 CE 55 28 DF 8C A1 89 0D BF E6 42 68 41 99 2D 0F ÎU(BC;ž.¿æBhA™-.
1002E340 B0 54 BB 16 52 09 6A D5 30 36 A5 38 BF 40 A3 9E °T»..R.jŌ06¥8¿@Ěž

```

Figure 4.41: Private Disk S-Box

4.4 AES Crypto Scanner: An Automated Approach

AES Crypto Scanner is the tool developed in this work for the purpose of automating the detection of AES in real applications. It is designed specifically for the extraction of AES different parameters from the binary code. It will take binary code as an input and will output the analysis result.

4.4.1 Tool Specification, GUI and Main Functions

AES Crypto Scanner is a desktop tools designed for Microsoft Windows platform. It is developed in visual studio using C# language. The libraries used in the development of this tool are RegularExpressions, Linq, Drawing and Threading. RegularExpressions is used to extract special characters from file. Linq is used to manipulate lists. Drawing is

used to highlight selected text i.e. AES constants found in the file.

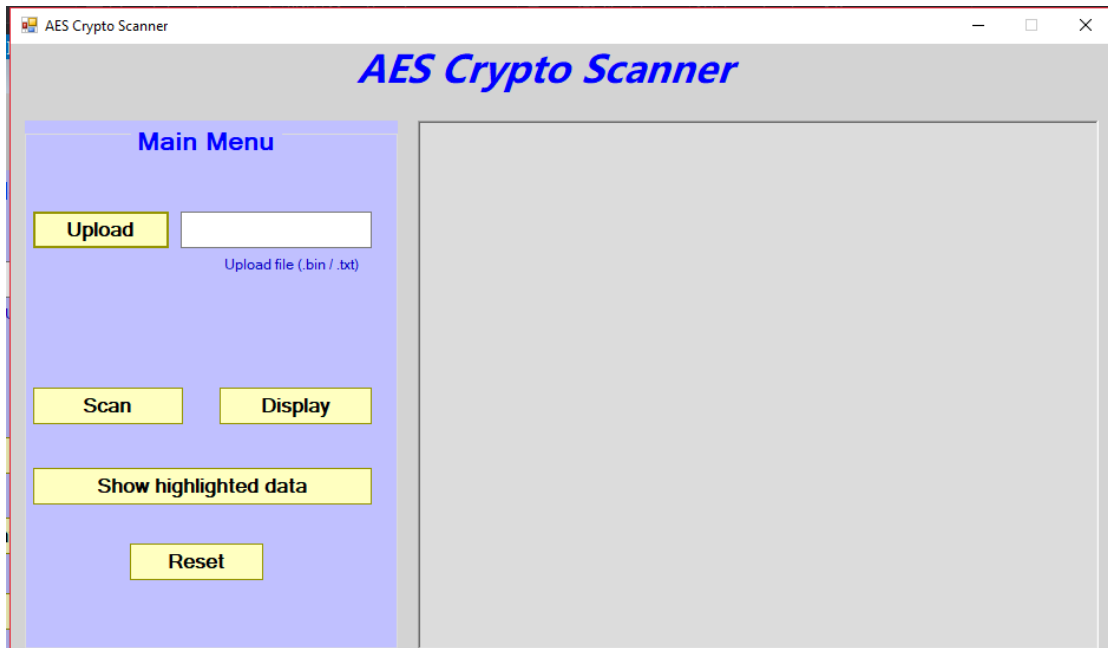


Figure 4.42: AES Crypto Scanner

Upload: Click this button to add the file containing binary code. It supports .txt and binary file format. By clicking upload, System takes input as a text file/ binary file which contains the assembly code of the application under analysis.

Scan: Click this button to scan the uploaded file and locate AES signatures if any.

Display: Click this button to show the output of scan. If the AES is found in the binary code then it will output the findings like which implementation is found, Encryption/Decryption routine used, mode of operations etc.

Show highlighted data: It will show the found signatures in assembly code so that an analyst can see the assembly code and the section where this signature is found. The signature will be highlighted so that it can be easily found.

Reset: This button will be used to reset and clear the main screen so that the tool can be used for another application if required.

Main Screen: This area will be used to show output results and binary code.

4.4.2 Benefits of AES Crypto Scanner w.r.t Existent Tools

The AES Crypto Scanner is developed to identify and locate AES parameters in Assembly code that was acquired after reverse engineering of the application. There are other tools like FindCrypt2, IDA Signsrch and SND Crypto Scanner that is used for the same purpose but AES Crypto Scanner has few features that makes it unique.

Simple Interface: AES Crypto Scanner has user friendly interface which is very easy to use even by novice user. The main menu is very simply designed which provide clear instructions about the tool. It is fully GUI based and independent tool, means it is not a plugin for any other tool. As compare to this FindCrypt2 and SND Crypto Scanner are both plugins whereas IDA Signsrch is command line tool which is not comfortable to uses at beginning.

Rich Interface: AES Crypto Scanner provide rich interface as compare to other similar tools. It can be used for scanning the file, Displaying the constants found in file and even highlighting the signatures in code.

Categorization: This tool does not only output the string “AES found/not found ”but also provide additional information like library used, s-box used, encryption and decryption routine used etc. which can help analyst to look for the specific parameter if required.

Aid in analysis: The option of highlighting crypto data in assembly code is very useful feature specially for analyst as they can quickly skip the other data and can only focus on the highlighted data which will reduce complexity of code and save time as well.

Scalable: This is not a final tool with final signatures. More signatures can be added to its database which makes it scalable. If analyst found few signatures which is used for AES, it can be added into the database and hence the database will increase and the

tool will become more powerful.

4.5 Tools Comparison

This section contains information of tools used for the detection of AES. It also shows the performance of every tools as compare to other.

4.5.1 List of Tools/Plugins

There are number of tools used for the detection and analysis of binary code. Every tool has strength and limitation as well. For this work 3 main tools were used other than AES crypto Scanner. The tools/plugins used for the detection of AES were:

- FindCrypt2
- IDA Signsrch
- SND Crypto Scanner
- AES Crypto Scanner (Developed tool for this framework)

From the results it can be seen that both IDA Signsrch and SND Crypto Scanner have very successful results as compare to FindCrypt2 plugin.

4.5.2 Results

The final results are divided in two potions. One for open-source applications and second for close-source application. Secondly the tools comparison can also be seen in the tables.

Open-Source Applications: A total of 5 open-source applications were analysed using the proposed methodology and the results were confirmed as the source code was available. The following results are obtained from 4 different tools i.e. AES Crypto

Applications	Tools	AES Detection	S-Box\ Table-Lookup Detection	Implementation Detection
7Zip	AES Crypto Scanner	✓	✓	✓
	FindCrypt2	✗	✗	✗
	IDA Signsrch	✓	✓	✗
	SND Crypto Scanner	✓	✗	✗
AxCrypt	AES Crypto Scanner	✓	✗	✓
	FindCrypt2	✗	✗	✗
	IDA Signsrch	✓	✓	✗
	SND Crypto Scanner	✓	✗	✗
Diskcryptor	AES Crypto Scanner	✓	✗	✓
	FindCrypt2	✗	✗	✗
	IDA Signsrch	✗	✓	✗
	SND Crypto Scanner	✓	✗	✗
VeraCrypt	AES Crypto Scanner	✓	✗	✓
	FindCrypt2	✗	✗	✗
	IDA Signsrch	✓	✓	✗
	SND Crypto Scanner	✓	✗	✗
EncFSMP	AES Crypto Scanner	✓	✗	✓
	FindCrypt2	✗	✗	✗
	IDA Signsrch	✓	✓	✗
	SND Crypto Scanner	✓	✗	✗

Table 4.3: Open-Source applications result comparison

Scanner, FindCrypt2, IDA Signsrch, SND Crypto Scanner.

It can be seen in Table 4.3 that FindCrypt2 plugin was not able to detect AES signatures, not even in a single application whereas other 3 tools have successfully detected the AES signatures.

Close-Source Applications: Other than open-source application, 6 close-source applications were also analysed using 4 different tools i.e. AES Crypto Scanner, FindCrypt2, IDA Signsrch, SND Crypto Scanner, and the final result of all the tools are presented in the Table 4.4.

Applications	Tools	AES Detection	S-Box\ Table-Lookup Detection	Implementation Detection
Privacy Drive	AES Crypto Scanner	✓	✓	✓
	FindCrypt2	✓	✗	✗
	IDA Signsrch	✓	✓	✗
	SND Crypto Scanner	✓	✗	✗
SensiGuard	AES Crypto Scanner	✓	✗	✓
	FindCrypt2	✗	✗	✗
	IDA Signsrch	✗	✗	✗
	SND Crypto Scanner	✗	✗	✗
Boxcryptor	AES Crypto Scanner	✓	✗	✓
	FindCrypt2	✗	✗	✗
	IDA Signsrch	✓	✓	✗
	SND Crypto Scanner	✓	✗	✗
Rohos Mini	AES Crypto Scanner	✓	✓	✓
	FindCrypt2	✗	✗	✗
	IDA Signsrch	✓	✓	✗
	SND Crypto Scanner	✓	✗	✗
BestCrypt	AES Crypto Scanner	✓	✓	✗
	FindCrypt2	✗	✗	✗
	IDA Signsrch	✓	✓	✗
	SND Crypto Scanner	✓	✗	✗
Private Disk	AES Crypto Scanner	✗	✗	✗
	FindCrypt2	✗	✗	✗
	IDA Signsrch	✓	✓	✗
	SND Crypto Scanner	✗	✗	✗

Table 4.4: Close-Source applications result comparison

The Table 4.4 shows the performance of all 6 tools. It can be easily seen that FindCrypt2 again has very poor performance and it has only detected AES in 1 application out of 6. Similarly, IDA Signsrch and SND Crypto Scanner has detected 5 applications out of 6 which shows a high detection ratio as compare to FindCrypt. AES Crypto Scanner has detected AES in all applications which represent that it has better detection than all other tools used for similar purpose.

4.6 Summary

This chapter provides the implementation detail of proposed framework. In this chapter, a total of 11 applications were analysed in which 5 of them were open-source and 6 close-source. It can be seen that out of 11, 5 applications were using Intel AES Instruction set [42], 2 applications were detected using bouncy crypto library, 1 application was using LibtomCrypt implementation, 1 application with OpenSSL implementation and 2 application with unknown implementation but standard s-box exists in the binary including some random signatures as well. The results section makes a good evaluation of the 4 tools and it can be seen that AES Crypto Scanner performed convincingly well as compare to other tools.

Conclusion & Future Directions

5.1 Conclusion

It is a belief that increasing the key size has a direct effect on the security of applications which is not completely true as key length can only increase the mathematical complexity and resist in mathematical and algebraic attack [46] but it does not consider algorithmic attack nor implementation-related attacks that bypasses the mathematical complexity of cipher. Secondly it also increases the computation time which decreases the performance of application. Hence developing secure products/applications requires standard key length and standard implementation which is flawless so that it can provide desired security and performance.

The purpose of this work was to evaluate the applications, look for the AES signatures that provides information regarding rounds used, key size, algorithm used, s-box used and mode used if possible. A framework was designed to prepare for evaluation process and then perform analysis that provides meaningful information to analyst. Few AES implementations were selected for this work and the proposed framework was implemented on 5 open-source and 6 close-source applications. It was found that many applications were using the standard implementation of AES and the real challenge was to determining the mode in assembly code. Many applications used standard s-box and it was found that for complete drive encryptions the applications used XTS mode

which is a standard approach. Different tools were also evaluated and the results shows that FindCrypt2 plugin was not up to the mark whereas the other similar tools showed good results. The tools developed for this work had the successful output as compare to other tools. The main challenge of this work is to identify standard implementations and acquiring signatures and if this stage is done correctly the entire framework become successful and if these stages are not catered correctly than the framework will fail. So, it is highly recommended to utilize maximum time/efforts in finalizing the algorithms and signature which actually contribute in the success of this work.

5.2 Future Directions

Apart from contributing in the field of signature detection this work has also increased the reliability of applications by conforming that secure implementation is used in the applications. This work can also be strengthened by implementing the following features:

- Add more libraries and signatures which confirms the defined standard of this work.
- Increase the signature repository and include other cryptographic algorithms like DES, Twofish etc. which will make this tool very rich.
- This work should be enhanced by adding known vulnerabilities into database.
- The proposed framework is based on the static analysis of code that's why it is not possible to acquire all the useful information correctly. For this reason, a hybrid approach should be developed which uses both static and dynamic analysis of code. It will help in the detection of modes, number of rounds and flow of code as well.

5.3 Summary

This chapter has summarized the research work by providing a very brief overview of the research conducted. It gave a complete sketch of the purposed framework and implementation of proposed framework in this research. Furthermore, it has also set future directions which will be useful for researchers of the same field.

List of Abbreviations and Symbols

Abbreviations

AES	Advanced Encryption Standard
DES	Data Encryption Standard
.EXE	Extension for an Executable File Format
DLL	Dynamic Link Library
RE	Reverse Engineering
IDA	Interactive DisAssembler
S-Box	Substitution-Box
FIPS	Federal Information Processing Standards
ECB	Electronic Codebook
CBC	Cipher Block Chaining
CFB	Cipher Feedback
OFB	Output Feedback
XTS	XEX-based tweaked codebook mode with ciphertext stealing
IGE	Infinite Garble Extension
DB	Database

References

- [1] Nazaruk, Vladislav, and Pavel Rusakov. "Implementation of Cryptographic Algorithms in Software: An Analysis of the Effectiveness." *Scientific Journal of Riga Technical University. Computer Sciences* 41.1 (2010): 97-105.
- [2] Kinder, Johannes. *Static analysis of x86 executables*. No. THESIS_LIB. Technische Universitat Darmstadt, 2010.
- [3] B.N.J. Rubenking, "The Best Encryption Software of 2018," *PCMAG*. [Online]. Available: <https://www.pcmag.com/article/347066/the-best-encryption-software-of-2016>. [Accessed: 12-Jul-2018].
- [4] NIST, FIPS PUB. 197, "Advanced Encryption Standard (AES)," November 2001.
- [5] Bassham III, Lawrence E. "The advanced encryption standard algorithm validation suite (AESAVS)." NIST Information Technology Laboratory (2002).
- [6] "Reverse Engineering." [Online]. Available: <https://ethics.csc.ncsu.edu/intellectual/reverse/study.php>. [Accessed: 04-Nov-2018].
- [7] Vigna, Giovanni. "Static disassembly and code analysis." *Malware Detection*. Springer, Boston, MA, 2007. 19-41.
- [8] Grobert, Felix, Carsten Willems, and Thorsten Holz. "Automated identification of cryptographic primitives in binary programs." *International Workshop on Recent Advances in Intrusion Detection*. Springer, Berlin, Heidelberg, 2011.

- [9] Sikorski, Michael, and Andrew Honig. Practical malware analysis: the hands-on guide to dissecting malicious software. no starch press, 2012.
- [10] Aabidi, M. H., et al. "Benefits of reverse engineering technologies in software development makerspace."ITM Web of Conferences. Vol. 13. EDP Sciences, 2017.
- [11] Canfora, Gerardo, Massimiliano Di Penta, and Luigi Cerulo. "Achievements and challenges in software reverse engineering."Communications of the ACM 54.4 (2011): 142-151.
- [12] Nguyen, Phong Q. "Can we trust cryptographic software? Cryptographic flaws in GNU Privacy Guard v1. 2.3."International Conference on the Theory and Applications of Cryptographic Techniques. Springer, Berlin, Heidelberg, 2004.
- [13] Benedetti, Leonard, Aurelien Thierry, and Julien Francq. "Detection of cryptographic algorithms with grap."IACR Cryptology ePrint Archive (2017): 1119.
- [14] Hill, Gregory D., and Xavier JA Bellekens. "Deep Learning Based Cryptographic Primitive Classification."arXiv preprint arXiv:1709.08385 (2017).
- [15] Hosfelt, Diane Duros. "Automated detection and classification of cryptographic algorithms in binary programs through machine learning."arXiv preprint arXiv:1503.01186 (2015).
- [16] Matenaar, Felix, et al. "CIS: The crypto intelligence system for automatic detection and localization of cryptographic functions in current malware."Malicious and Unwanted Software (MALWARE), 2012 7th International Conference on. IEEE, 2012.
- [17] Lestringant, Pierre, Frederic Guihery, and Pierre-Alain Fouque. "Automated identification of cryptographic primitives in binary code with data flow graph isomorphism."Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security. ACM, 2015.
- [18] Calvet, Joan, Jose M. Fernandez, and Jean-Yves Marion. "Aligot: cryptographic

- function identification in obfuscated binary programs.”Proceedings of the 2012 ACM conference on Computer and communications security. ACM, 2012.
- [19] Xu, Dongpeng, Jiang Ming, and Dinghao Wu. “Cryptographic function detection in obfuscated binaries via bit-precise symbolic loop mapping.”Security and Privacy (SP), 2017 IEEE Symposium on. IEEE, 2017.
- [20] Zhao, Ruoxu, et al. “Detection and analysis of cryptographic data inside software.”International Conference on Information Security. Springer, Berlin, Heidelberg, 2011.
- [21] “Welcome to Hex-Rays.”[Online]. Available: <https://www.hex-rays.com/>. [Accessed: 04-Mar-2018].
- [22] “OllyDbg v1.10.”[Online]. Available: <http://www.ollydbg.de/>. [Accessed: 04-Mar-2018].
- [23] “FindCrypt - Hex Blog.”[Online]. Available: <http://www.hexblog.com/?p=27>. [Accessed: 13-June-2018].
- [24] Monnappa K A. “Malware Obfuscation Techniques”in Learning Malware Analysis, 1st ed., Birmingham: Packt Publishing, 2018
- [25] T. Xie, F. Liu, and D. Feng. “Fast collision attack on MD5.”IACR Cryptology ePrint Archive, 2013:170,2013.
- [26] “7-Zip.”[Online]. Available: <https://www.7-zip.org/>. [Accessed: 21-July-2018].
- [27] “Microsoft .NET Portable Library Reference Assemblies 4.6,”Microsoft Download Center. [Online]. Available: <https://www.microsoft.com/en-pk/download/details.aspx?id=40727>. [Accessed: 21-jul-2018].
- [28] Edgar, Michael. (2016). Legion of the Bouncy Castle Inc. BC-FNA (Bouncy Castle FIPS .NET API) FIPS 140-2 Cryptographic Module Security Policy. 10.13140/RG.2.2.20033.04969.

- [29] "AxCrypt - File Security Made Easy,"AxCrypt - File Security Made Easy. [Online]. Available: <https://www.axcrypt.net/>. [Accessed: 01-Aug-2018].
- [30] "DiskCryptor wiki." [Online]. Available: https://diskcryptor.net/wiki/Main_Page. [Accessed: 01-Aug-2018].
- [31] "VeraCrypt," [Online]. Available: <https://sourceforge.net/projects/veracrypt/>. [Accessed: 01-Aug-2018].
- [32] "EncFSMP homepage." [Online]. Available: <https://encfsmp.sourceforge.io/>. [Accessed: 18-Aug-2018].
- [33] "Privacy Drive encryption software: lock, hide & protect data." [Online]. Available: <http://www.cybertronsoft.com/products/privacy-drive/>. [Accessed: 21-Aug-2018].
- [34] "libtomcrypt Git at Google." [Online]. Available: <https://android.googlesource.com/platform/external/dropbear/+donut-release/libtomcrypt/src/ciphers/aes/aes.c>. [Accessed: 22-Aug-2018].
- [35] "libtom," [Online]. Available: <https://www.libtom.net/LibTomCrypt/>. [Accessed: 28-Aug-2018].
- [36] "SensiGuard - File Encryption Software - Lock Files - Lock Folders." [Online]. Available: <https://www.sensiguard.com/>. [Accessed: 28-Aug-2018].
- [37] B. Laurie, "OpenSSL's implementation of Infinite Garble Extension", 2006
- [38] "Boxcryptor | Security for your Cloud." [Online]. Available: <https://www.boxcryptor.com/>. [Accessed: 28-Aug-2018].
- [39] "Rohos Mini Drive - Rohos." [Online]. Available: <https://www.rohos.com/products/rohos-disk-encryption/rohos-mini-drive/>. [Accessed: 02-Sep-2018].
- [40] "Encryption Software & Wiping Software | Jetico." [Online]. Available: <https://www.jetico.com/>. [Accessed: 02-Sep-2018].

- [41] “Disk encryption software - data protection and disk encryption with Dekart Private Disk.”[Online]. Available: https://www.dekart.com/products/encryption/private_disk/. [Accessed: 04-Sep-2018].
- [42] Gueron, Shay. “Intel[®] Advanced Encryption Standard (AES) New Instructions Set.”Intel Corporation (2010).
- [43] “OpenSSL Cryptography and SSL/TLS Toolkit.”[Online]. Available: <https://www.openssl.org/>. [Accessed: 08-Sep-2018].
- [44] D. J. Bernstein, T. Lange, and P. Schwabe, “The Security Impact of a New Cryptographic Library,”in Progress in Cryptology - LATINCRYPT 2012, vol. 7533, A. Hevia and G. Neven, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 159-176.
- [45] “B. Gladman, AES code.”[Online]. Available:<https://github.com/BrianGladman/aes>. [Accessed: 02-Aug-2018].
- [46] M. Neve and K. Tiri, “On the complexity of side-channel attacks on AES-256 – methodology and quantitative results on cache attacks,”318, 2007.