

Identification of Security Mechanism in Java Based Crypto Apps Using Reverse Engineering



By

Muhammad Haseeb Javed

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

THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS Thesis written by **Muhammad Haseeb Javed** Registration No. **00000172051**, 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.

ACKNOWLEDGMENTS

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; Muneeb Ahmad and Asad Mehdi, 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 am very grateful to my supervisor, Dr. Mehreen Afzal who provided me a platform and gave me the liberty to work in the area of my interest and continuously supported me during the course of this research. Her technical guidance, encouragement, ideas and perspective were vital for completion of this tedious task. Her support gave me confidence and helped me to understand the subject matters deeply and inspired me towards my goals.

I would also like to thank Dr. Fawad Khan and Asst. Professor Mian Muhammad Waseem Iqbal for being an important part of my Research Supervisory Committee. Their scholarly guidance, assistance and knowledge have been meaningful for successful completion of my research. Finally, 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.

Muhammad Haseeb Javed

November 2018

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

Cryptographic algorithms are used in number of applications to provide different security services. Correctness of algorithm and their implementation is a question in the face of today's threat perspective. In the situation where companies are involved in manipulating the security algorithms, it becomes important that code used for providing security is analyzed for its correctness before they are being used. For open source applications, the subject analysis is possible but for proprietary applications and devices user has to trust the respective company. In recent years, some work can be found in the direction of reverse Engineering for the analysis of algorithms in researches as well as by companies. Reverse Engineering process involves disassembling the Binary code. Binary code of applications can be reverse engineered to get the working information, which can then be modified according to the requirements. Moreover, these can be tested for conformance that they are working according to expectations or otherwise.

This research will focus on the disassembly and de-compilation of Cryptographic application to get the code. Then the decompiled code obtained can be used to analyze the correctness of implemented cryptographic algorithms and key management system.

A solution is proposed in this research in which once we have the apk file we use ApkTool to decode the apk. Then extract the java code from the jar file. Now the next challenge is to get to the right code from thousands of java files, for this we introduced a tool "Crypto Surveillance" which in return give us only the files in which crypto code potentially exists, Then the code obtained can be used to analyze the correctness of implemented cryptographic algorithms and key management system. The second approach used in this research is using reflection API provided by java.it is useful in case when decompiled source code is not available and we have to work on .class files

ACRONYMS

DEFINITION	ACRONYM
Java Development Kit	JDK
Java Runtime Enviroment	JRE
Java Cryptographic Extension	JCE
Reverse Engineering	RE
Software Reverse Engineering	SRE
Application Program Interface	API
Advance Encryption Standard	AES
Secure Hash Algorithm	SHA
Extended Triple Diffie-Hellman	X3DH
Pseudo Random Number Generator	PRNG
Message Authentication Code	MAC
National Institute of Standards and Technology	NIST
National Security Agency	NSA
Public Key Crypto System	PKCS

Contents

1	INTRODUCTION	1
1.1	Introduction	1
1.2	Research Overview	1
1.3	Motivation and Problem Statements	3
1.4	Aims and Objectives	3
1.5	Research Methodology	3
1.6	Relevance to National Needs	4
1.7	Advantages	4
1.8	Area of Application	4
2	LITERATURE REVIEW	5
2.1	Introduction	5
2.2	Java Programming language	5
2.2.1	Java-Based Mobile Applications	6
2.2.2	Java-Based Desktop Applications	8
2.3	Reverse Engineering	8
2.3.1	Malware Reversing	9
2.3.2	Firmware Reversing	10
2.3.3	Cryptographic Algorithms Reversing	11

2.4	Related Work	12
2.5	Conclusion	13
3	TARGETTED APPS AND PROPOSED SOLUTION	14
3.1	Tools for Reverse Engineering of Android App	14
3.1.1	Static Analysis Tools	15
3.1.2	Dynamic Analysis Tools	16
3.2	Methodology	17
3.2.1	Getting Targeted Apks	17
3.3	Source Code Extraction	18
4	Code Searching and Refinement	22
4.1	Introduction	22
4.2	Others Code Searching Techniques	22
4.3	Developed tool for Code Searching	23
4.4	Java Reflection API	29
4.4.1	Reflection Use cases	29
4.4.2	Use of reflection in code identification	30
4.5	Conclusion	30
5	EXPLORATION OF ENCRYPTION KEY	31
5.1	Introduction	31
5.2	Whatsapp and Signal	31
5.2.1	Results (Signal and Whatsapp)	37
5.3	IMO	39
5.3.1	Results (IMO)	41
5.4	CONCLUSION AND FUTURE DIRECTIONS	41

5.4.1	Conclusion	42
5.4.2	Future Directions	42

References		43
-------------------	--	-----------

List of Figures

2.1	Java Applications.	6
2.2	Android Low-Level System Architecture	7
3.1	Decoding Apk	18
3.2	Apk Reversed	19
3.3	Converting .dex to .jar	19
3.4	.jar File Successfully Retrieved	19
3.5	Extracted Code by JD-GUI De-Compiler	20
3.6	Extracted Code by Luyten De-Compiler	20
3.7	Extracted Code of Selected Apps (JD-GUI - Luyten)	21
4.1	"Crypto-Surveillance" Flow Chart	24
4.2	Java Files in Whatsapp Package	25
4.3	Java Files in Signal Package	25
4.4	Crypto Surveillance Tool UI	26
4.5	All Selected Applications Source Folders	26
4.6	All Java Files from Package Selected	27
4.7	"AESEngine" Class from Signal Code	27
4.8	Imports in AESEngine Class in Signal	28
4.9	Imports in AES Class in Whatsapp	28

4.10 Reflection Method Output	30
5.1 Message Send Function Against the “send” Button Click	32
5.2 Key Exchange Function in Signal Application	32
5.3 "sendMessage" Function in Signal Application	33
5.4 "deliver" Function in Signal Application	33
5.5 Un-Readable Form of Actual Key	34
5.6 Secure Random Secret Bytes	34
5.7 Padded Message and Encrypted Message Body	35
5.8 "encrypt" Method of "CipherTextMessage" Class	36
5.9 Final Shape of Message to be Send	37
5.10 All Java Files in IMO Package	39
5.11 All Crypto-related files in IMO Package	40
5.12 "AES" class in IMO application	40

List of Tables

5.1	Session Key Types	38
5.2	Public Key Types	38
5.3	Comparison of Application	42

INTRODUCTION

1.1 Introduction

This introductory chapter will help in giving a brief introduction of this research thesis. It begins with the power of reverse engineering. It will put some light on emerging java based applications and introduce the advancement of cryptographic algorithms used in those applications. The last section will include the significance of study to the industry and academia.

1.2 Research Overview

Cryptographic algorithms are used in number of applications to provide different security services. Correctness of algorithms and their implementations is a question in the face of today's threat perspective. Nowadays, cryptography is considered as best and powerful way of security. Its power can be represented via the fact that correctly implemented cryptographic algorithms are considered secure and are only breakable with a successful brute force attack which might take decades. In the situation where companies are involved in manipulating the security algorithms, it becomes important that code used for providing security is analyzed for its correctness before they are being used. For open source applications, the subject analysis is possible but for proprietary applications and devices user has to trust the respective company. In recent years, some

work can be found in the direction of reverse Engineering for the analysis of algorithms in researches and by the companies as well.[1] Reverse Engineering is the process of getting the implementation and design information of any application, and reproduce it based on the extracted information. This process involves disassembling the Binary code. Binary code of applications can be reverse engineered to get the working information, which can then be modified according to the requirements. Moreover, these can be tested for conformance that they are working according to expectations and the cryptographic properties used are according to latest standards or otherwise. Reverse engineering tools[2] have been used for Vulnerability and threat detection, Source code recovery and App Modding. However, there are some limitations for reverse engineering tools such as Possible loss of some code, lack of accuracy in code recovery, and some times Custom framework are required to do reverse engineering.

When assessing a mobile app, it is important to make sure that it does not use cryptographic algorithms and protocols that have known weaknesses or insufficient for modern cryptography requirements. Algorithms becomes insecure over time that were reckoned enough secure in the past; Hence the important task needs to be done is continuously checking the available best practices and adjust our security configurations accordingly. It is important to verify the cryptographic algorithms are up to date and in-line with industry standards. Cryptographic Algorithms are meant to provide enhanced security however correct version needs to be implemented correctly Vulnerable algorithms include outdated block ciphers (such as DES), stream ciphers (such as RC4), hashing functions (such as MD5), and broken random number generators. Note that even algorithms that are certified (for example, by NIST) can become insecure over time. A certification does not replace periodic verification of an algorithm's soundness. Algorithms with known weaknesses should be replaced with more secure alternatives.

The following algorithms are recommended[3]. In Confidentiality algorithms AES-GCM-256 or Poly1305[4]. In Integrity algorithms SHA-256[5], SHA-384, SHA-512, Blake2. In Digital signature algorithms RSA (3072 bits and higher), ECDSA with NIST P-384. And in Key establishment algorithms RSA (3072 bits and higher), DH (3072 bits or higher), ECDH with NIST P-384.

1.3 Motivation and Problem Statements

Android operating system has become the most popular and used operating system for smart phones, with an estimated market share of 70% to 80%,[6], and in the past several years, the popularity of smart phones has risen significantly. There are a lot of applications which are in use in our daily routines and user have no knowledge what it actually going on with in the apps and user just trust the developers.

Analysis of the malicious behavior of app is generally carried out through traffic analysis or behavior of the app[7]. However, any weakness in the crypto mechanism cannot be detected through these techniques. It is therefore important to analyze the cryptographic code for its correctness. Few researches can be found in this direction however there is a need to further explore this domain.

1.4 Aims and Objectives

The objectives of this research include

- 1) Exploring reverse engineering tools for disassembly and de-compilation of crypto implementations of Android applications
- 2) De-compilation of the Assembly code through appropriate open source tools and finding their limitations
- 3) Find Cryptographic properties from code.
- 4) Analyze the strength of crypto implementations including Key generation mechanism.

1.5 Research Methodology

At first literature review is conducted to find out the existing mechanisms and techniques used to analyze the behavior of applications. Applications are then chosen for applying reverse engineering and then we get our desired information like information of encryption and key mechanism from code. The libraries used in code for crypto-

graphic implementation are either open source or closed source. Based on our findings a proposed output has been presented.

1.6 Relevance to National Needs

Pakistan is the user of most of the foreign-based technologies and has developed a few. Analysis of the implementations up to the code level is very challenging and very less explored area. This research is related to an important area of cyber security.

1.7 Advantages

- 1) It will help in analyzing the applications in different perspectives.
- 2) It will provide awareness to the user to understand how their personal data can be exploited for example by having extra permission or by sending data insecurely.
- 3) It will help in letting user know that how to have a check on key generation mechanism used in code whether is good enough or outdated.

1.8 Area of Application

It will be helpful for the organizations who wish to build a secure mobile product or to secure themselves from malicious activities on their Java based desktop application or mobile application.

LITERATURE REVIEW

2.1 Introduction

This chapter extensively addresses about Java programming language, Java Applications working. In section 2.3, Reverse engineering is explained that how reverse engineering can help in code retrieval. In section 2.3, types of tools that are commonly used in reverse engineering are introduced and also provide an outline of the technical basics that are important to apprehend before start reversing and in the later part some related work discussed.

2.2 Java Programming language

Java is designed explicitly to have few implementation dependencies. The idea behind this is to let developers “write Once Run Anywhere”.

Sun Micro-systems developed Java in 1995 and nowadays Java has turned out as backbone of millions of applications over numerous platforms. According to Oracle, Java currently installed on more than 3 billion devices[8].

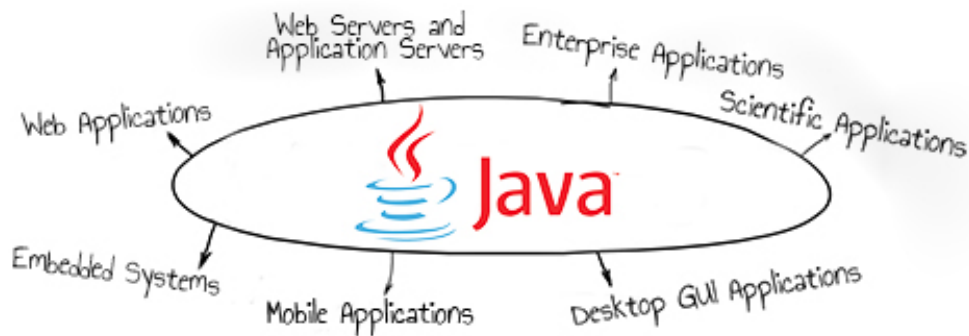


Figure 2.1: Java Applications.

2.2.1 Java-Based Mobile Applications

When talking about Java mobile development there are two branches, Java ME, and Android development. Java ME on mobile devices is pretty much dead or dying, while Android is still good to work.

Android is a Linux-based operating system for smart phones, which is open source. It was created by the Open-Handset-Alliance, led by Google, and other organizations. Android programming is based on Java language[9]. Android phones are usually equipped with pre-installed applications and at the same time also support applications by third-parties. Android applications can be developed by using free Android SDK[10]. Java[9] is used to write Android programs and run on Java virtual machine which is optimized for mobile devices. The “Dalvik” JVM was utilized through Android 4.4 and was replaced by Android Runtime[11] or “ART” in Android 5.0. One can easily download and run Android apps from online app store such as Google Play[12], although there are also other online app stores. Android operating system is a bundle of software components that are divided into five categories and four main layers as shown in Figure 2.2.

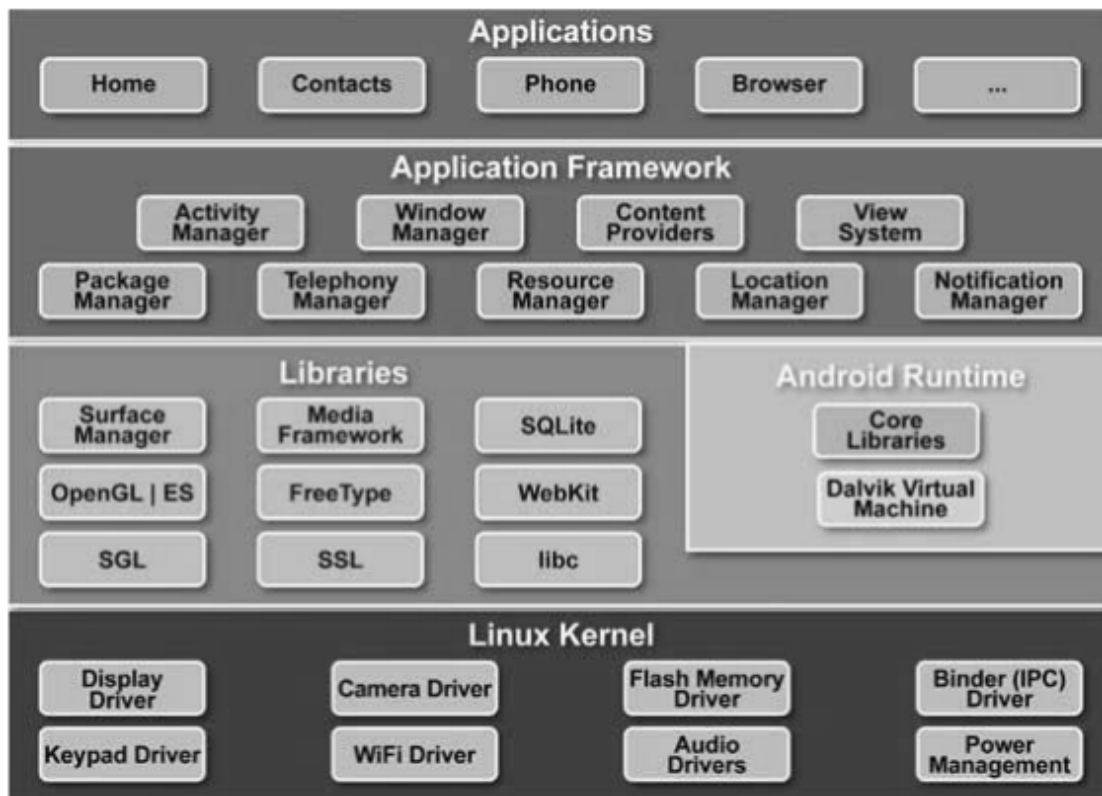


Figure 2.2: Android Low-Level System Architecture

Android uses .apk (Android Package Kit) file format for distribution and installation of applications. Similar to Windows applications which uses .exe file for installation of applications on a system. There are several significant components in an .apk file.

Android Manifest: AndroidManifest.xml file is the essential part of all android applications in its root directory, the file contains necessary information belonging the app to the Android System e.g. content providers, services, broadcast, unique identifier and activities. The AndroidManifest.xml declares the permissions for an application that of which protected parts of the API it has access to and communicate with other applications. Minimum level of the API an application requires and the lists of libraries with which the application must be linked are also declared in this file.

Classes.dex: A program in Android programming language is first compiled to make an apk file, and after that the majority of its components are bundled into the only file which holds all of the program's code, assets, certificates and App Manifest file. These

are .dex files of the application. Projects are generally written in Java and compiled to bytecode. They are then changed over from “.class” files to “.dex” files. The compact Dalvik Executable configuration is intended to be reasonable for frameworks that are constrained as far as memory and processor-speed concerns.

App Resources: Android Apk contains a directory named as “res” contains all the resources added into application. Resources are extra files and static content that are use in application code, such as bitmaps, user interface strings, layout definitions, animation instructions, and others. As we add app resources the more app size will increase.

Signature: Application signing provides unique identification of the author of an application and helps them update the application without creating complicated permissions and interfaces. Application must be signed by the author before running on an android platform. Google Play and the package installer both on Android device do not allow installation of any application of unsigned application.

2.2.2 Java-Based Desktop Applications

When we talk about java desktop applications there are many softwares and application comes in our mind which is based on Java language e.g. Many popular Desktop Integrated Development Environments (IDEs) are built using Java programming language. Some examples are Eclipse, IntelliJ and NetBeans. Gmail is a popular free email service by Google. It uses Java as the back-end and Java-Script/Ajax on the front-end[13]. so we can easily reverse engineer any of the application or software to get its source code. and then later on IDA pro can be use to convert machine code into assembly and then its Java readable form.

2.3 Reverse Engineering

Reverse Engineering (RE) or (SRE) is the process of getting the implementation and design information of any application, and reproduce it based on the extracted infor-

mation. This process involves disassembling the Binary code, then binary code of applications can be reverse to get the working information, which can then be modified according to the requirements. Reverse engineering is performed in order to acquire missing information, getting design philosophy and core ideas of an application if not available. Such information is normally made unavailable, may be because someone does not want to share it or some time the information has been lost.

Reverse engineering tools have been used for Vulnerability and threat detection, Source code recovery and App Modding. However, there are limitations for reverse engineering tools such as Possible loss of some code, lack of accuracy in code recovery, and some times Custom framework are required to do reverse engineering.

2.3.1 Malware Reversing

Reverse Engineering can be use in Malware analysis. Nowadays organizations face one of the greatest threats in the shape of Malwasre and they know their Antivirus tools can only protect them from malware but they dont know what malware might have done and what they might have lost if malware were executed. Answers to such questions can be difficult to find, but the only hope in such conditions can be reverse engineering of malware because its the only case where we get the internal code of malware and then we can analyze its working and the damage it might have done if executed. Multiple researches can be found in this field, one of them is briefly explained below.

In a research[14], researchers were trying on nugache worm to understand the attacker's methodology for discovering the vulnerable sites in the system and also they were trying to know is behaviour and basic design philosophy, to solve this problem they used reverse engineering methodology and they were successful in their intentions. The nugache worm came in early 2006 when it came as a simple Trojan horse, simple as compared to a more dangerous worm of the time called as Storm worm[16]. The researcher kept on reversing 49 more malware executables and successfully extracted many of the properties like printable strings, number of API calls made, URLs, MD5 Hash etc. By using these highlighted properties they create a dataset and because of the

multi-dimensional nature of dataset, they used a tool BLEM2[17] to create a dynamic patterns which would be helpful in analyzing a obscure malware. The results were not so satisfactory because of the small in size of dataset and very few decision rules were created.

2.3.2 Firmware Reversing

The combination of a hardware device, computer instructions and data which is considered to be read-only software on that device is Firmware[18]. The ability analyze and read the data from a firmware is extremely useful. It allows to test an embedded device for bugs without having access to the device. The code inside every executable binary is Machine Code for to be processed by the CPU. A disassembler can be used to go through the binary and convert it into assembly code which is not like the original code, but at least it's human readable. There are many dis-assemblers for popular architectures; some are comparably better in usability and functionality. These three are the popular, robust and effective dis-assemblers available in the market.

IDA Pro is undoubtedly the most popular disassembler and debugger in the market. The reason behind is its multi-platform behaviour and also there are many users, tutorials, plugins available to learn about it. Sadly, it is very expensive; License of the Pro version costs over \$1000 for a single person [19]

Radare2 is disassembler which is open source having advanced command line interface. It is available for many different architectures. It can run on all platforms OSX, windows, Linux, iOS, Android, solaris and Haiku.[20]

Binary Ninja is Not open source, but for a personal license it has reasonable priced at \$149.[21] it is a middle way between IDA and radare. It's still a very new and simple tool. it is improving rapidly and gaining popularity. it contain fully featured Hex editor and also provide 1 year of updates.

2.3.3 Cryptographic Algorithms Reversing

Cryptography means secrecy: person-1 encrypt a message by the secret key known to both person-1 and person-2 and then send this message to person-2. Cryptographic algorithms generally break down into two groups one is restricted algorithms and second is key-based algorithms. Restricted algorithms are like composing a letter to a companion with each letter moved a few letters up or down. the secret of this type of algorithms is algorithm itself, if the algorithm exposed it'll be no longer secure. It has very poor security because once the reverser the the code or sequence of algorithm it will be matter of few seconds to disclose the algorithm. In contrast to this algorithm we have key-based algorithms in which secret is a key not the algorithm itself. key is based on some numeric value to encrypt and decrypt the message. user encrypt the message by private key. The algorithm often made public and the keys are kept private, this make the reversing process useless. but still its possible to get the key by reversing only in case if the programmer have hard coded the key or the key is generating within the code (luck for reversers).In key-based cipher if we want to decipher a message, we would have to either:

- 1) get the key.
- 2) we have to try all the possible combinations in expecting to get the key.
- 3) try to find a flaw or vulnerability in algorithm, that can lead you to the key or more luckily to the plain text

so we can conclude that encryption algorithms are mild and a little mistake or implementation error can completely disprove the level of security. The only way to confirm about the working of implemented security algorithm is to go through the code or reverse the code if not available.

2.4 Related Work

Currently, the erroneous use of cryptographic protocols and functionalities by applications has attained serious attention of researchers. Georgiev and others have proved that correct use of conventional or hardened security protocol such as Transport Layer Security (TLS) is a challenging task. Particularly, malformed certificate validation makes the widely used applications a soft target for Man-in-the-Middle (MITM) attacks. Besides identifying these issues, several techniques for mitigation are also proposed. Several PC and mobile devices have been analyzed by Georgiev et al. on the other hand Fahl et al. have targeted the same issue in Android Applications. They developed a tool named MalloDroid[22] and analyzed over 13,500 android applications, out of which 1,074 applications were caught using TLS with deficient certificate validation. Out of those Fahl et al. performed manual analysis of 100 applications and successfully launched MITM attacks on 41 apps. MalloDroid is built on Androguard reverse engineering framework[23] to detect use of TLS through static code analysis. Semdroid incorporates machine learning techniques to recognize cryptographic implementations in a code.

One way of verifying cryptographic implementations is the use of verification tools such as Microsoft Crypto Verification Kit[24], Murphi[25] and others. The advantage of such tools is that they ensure strong guarantees but on the other hand they are heavyweight, require expertise and a lot of manual effort. These limitations make such a tool unwanted for huge experiments and a software developer who has no knowledge of cryptographic mechanisms. Therefore, a lightweight yet effective approach for static analysis to identify common flaws. The tool used in this research, is named CryptoLint, built upon Androguard Android program analysis framework[23].

In another research Egele-et-al (2013) have examined the Android applications for programming faults while using cryptographic features. For that they have created CryptoLint[26], a tool that employs static code analysis to point out the applications that utilize cryptographic features and decides the parameters with which the application conjures these cryptographic features. CryptoLint can check these parameters

against an arrangement of guidelines characterizing basic programming faults. Their test demonstrates that of the 145,095 Google Play Store applications they inspected, 15,134 utilize cryptographic features, of which CryptoLint could effectively dissect 11,748. Just 1,421 of these applications did not oppose any tenets. Like Semdroid, CryptoLint utilizes static code examination on compiled android applications.

2.5 Conclusion

This chapter highlights some of previous work related to identification of crypto implementations. Some of the work is in static analysis and the others in dynamic analysis. it can be concluded that most of the tools used in the above mentioned researches are CryptoLint, SemiDroid, Murphi, MalloDroid and others which perform code identification on binary-level only.

TARGETTED APPS AND PROPOSED SOLUTION

In this chapter, we explain about some available tools which can help us in reversing the android applications then we explain the applications chosen for analysis and what are the possible ways through which we can get the actual Apk (without any modification). At the end of this chapter we discuss about the extraction of source code from Apks. As we have to select some application for analysis so we select **Whatsapp**, **Signal** and **IMO** for analysis as these are the apps with more chances of having cryptographic implementations.

3.1 Tools for Reverse Engineering of Android App

In this section we discuss about the static and dynamic analysis tools for android apps. Apps can be statically or dynamically analyzed to check its behavior or working. Applications are statically analysed utilizing few techniques aiming at un-packaging and disassembling apps. this process usually performed by Androguard. We use ApkTool and dex2jar tools for un-packaging and re-packaging applications into a modified application. JD-GUI and Luyten are graphical utilities that display java code. We can also use dynamic analysis tool known as name Droidbox to keep monitoring different activities to get the app behavior, later in this section we also discuss about Taintdroid.

3.1.1 Static Analysis Tools

Androguard is a static analysis tool for Android applications. Through its API we can disassemble applications and also we can have access to its components. Androguard's API [23] gives access to every attribute (classes, methods, variables) of the binary code.

The Java Runtime Environment (JRE) is a platform where we have set of tools for Java applications development. It combines the Java Virtual Machine, core classes and libraries. For using ApkTool one must have atleast JRE 1.7 installed

ApkTool is a tool for reverse engineering for Android applications. It can decode resources from the Apk to nearly original form and recompile those resources after making potential modifications.

- 1) Disassembling all resources (classes.dex, Xmls, pngs etc.) to nearly original form
- 2) converting the decoded resources back into binary APK-JAR form
- 3) Organizing and handling framework-resources dependent APKs
- 4) Smali Debugging

dex2jar is used to create .jar file. it will create a file with a name like “someApk-dex2jar.jar” in the working directory. In other words, tools such as undx and dex2jar[27] can be used to convert the Dalvik-VM-bytecode into JVM-bytecode to obtain the java code and a Java de-compiler can then be used to decompile the Java code and then code like in original form will be available.

JD-GUI[28] is a standalone graphical based java decompiler that displays Java source codes of “.class” files. It also provides an option to open the de-compiled source code in JD-GUI for quick access to code.

Luyten[29] is an Open Source Java De-compiler Gui for java , All in all, Luyten is a useful and efficient Java de-compiler GUI for Procyon that does exactly what a tool like

this should do, with the added benefit of multiple customization features.

Further information about the tools briefly explained above can be found in the references given at the end of document.

3.1.2 Dynamic Analysis Tools

Droid-box is an android base dynamic analysis tool, that allows the applications execution and provides a variety of running data about the app behavior. Through this tool it can analyze or monitor the execution of 11 various activities: crypto, netopen, netread, netwrite, fileopen, fileread, filewrite, leak, call, sms, dexload. crypto activity generated when cryptographic API called, netopen, netread, netwrite is for network activities, sms is for text message sending or receiving, call activity generated whenever a call is made etc.

Taint-Droid is for dynamic taint analysis to track sensitive information. TaintDroid automatically labels (taints) data from privacy-sensitive sources applies labels and propagate through program variables, files, and messages. TaintDroid always logs the data labels whenever the tainted data leave or enter the system, the application responsible for transmitting the data, and the data destination. Such realtime feedback gives security researcher a greater insight into what applications are doing, and can identify misbehaving applications.

3.2 Methodology

In this section we explain the applications chosen for analysis and what are the possible ways through which we can get the actual Apk (without any modification)

3.2.1 Getting Targeted Apks

Apks can easily be downloaded from web, but it'll not be sure of apk being real and un-malicious. So here are some clean ways of getting real and un-malicious app Apk right from our mobile phones.

APKOptic: APKoptic is an application manager that empowers you to launch or uninstall applications from your phone. You can likewise make a backup of the application on SD card and in addition restore or install the APK from SD card.

Astro File Manager: Astro File Manager can Connect all our storages in single place internal memory, SD Card, cloud services like Google Drive, Dropbox, Microsoft OneDrive, or local networks PC, Linux and Mac. Apk can be extracted from installed application in your phone by Astro file manager in few seconds.

3.3 Source Code Extraction

The first step is to get the apk file of the app which is needed to be analyze. The ways of getting the apk are explained in the above section. Once you have the apk file the second step is to use ApkTool to decode the apk for this open cmd and go to the folder where APK file is placed, type “apktool d -m Threema.apk” and hit enter button to start decoding process.

```
C:\Users\hasee\Desktop\apk extract>apktool d -m Threema.apk
I: Using Apktool 2.3.3 on Threema.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
S: WARNING: Could not write to (C:\Users\hasee\AppData\Local\apktool\framework),
\ instead...
S: Please be aware this is a volatile directory and frameworks could go missing,
ult storage directory is unavailable
I: Loading resource table from file: C:\Users\hasee\AppData\Local\Temp\1.apk
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
```

Figure 3.1: Decoding Apk

it will extract the dex and the resources file here d is for decode and Threema.apk is the app name needs to be decode. -m is for match option if there is no need to rebuild the apk once you have modified it. So if only the exact same xml files and resource files are required then this command with the match option i.e -m is useful.

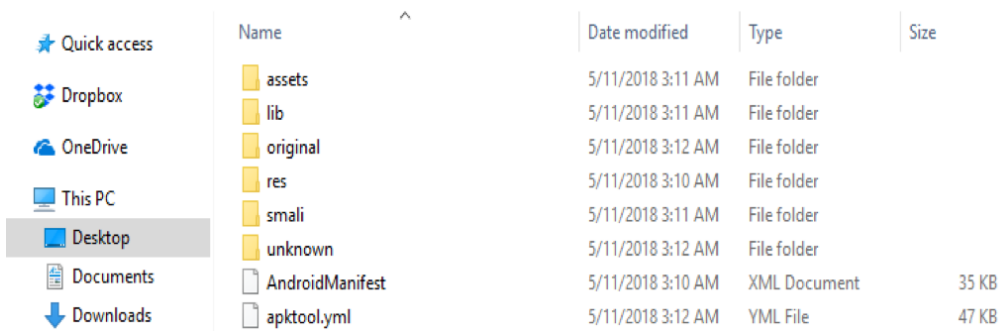


Figure 3.2: Apk Reversed

Here in the Figure 3.2 it can be seen that Android Manifest file is present. There is lib directory and resources directory present as well and it has all the images so all the resources can be properly extracted.

Now open Cmd and type “d2j Threema.apk”

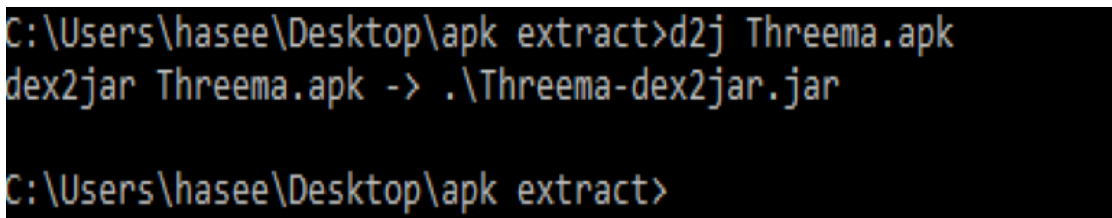


Figure 3.3: Converting .dex to .jar

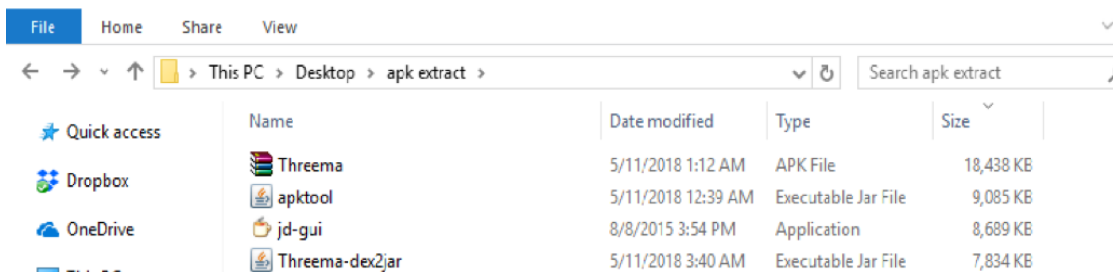


Figure 3.4: .jar File Successfully Retrieved

Now java code can be extracted from the jar file but since the conversion from android dex code to jar file is not perfect so at this stage certain information potentially lost which might lead to Improper java code. we will use two applications to convert jar file into java code, So in case if a particular file is not decoded properly by the first application then we will check the code generated by second application

- 1) JD-GUI
- 2) Luyten

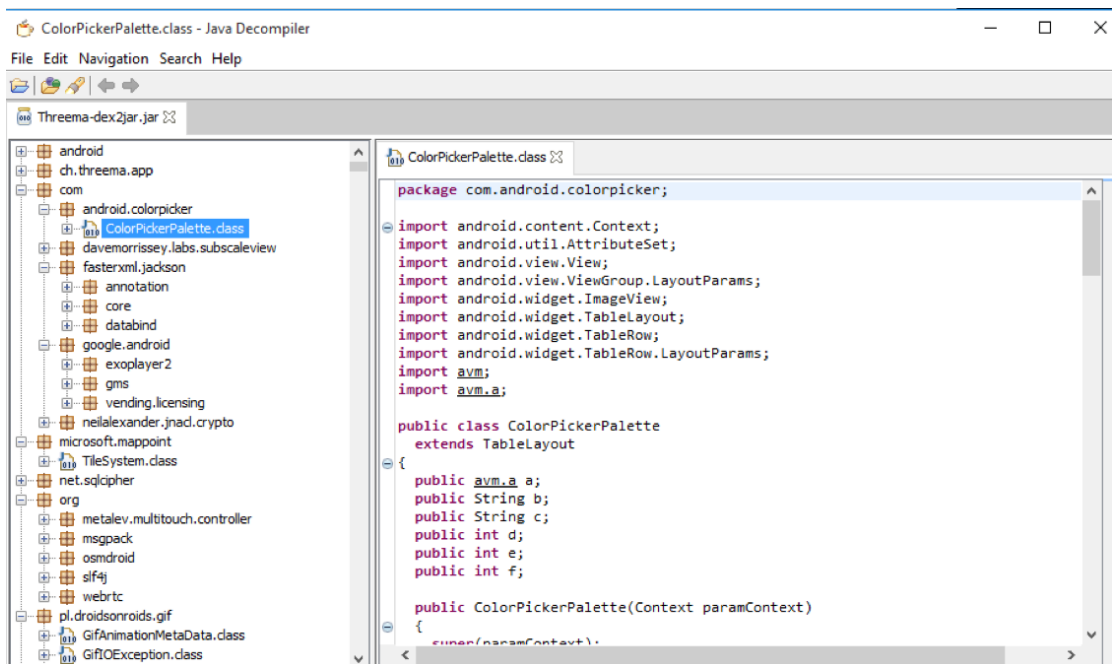


Figure 3.5: Extracted Code by JD-GUI De-Compiler

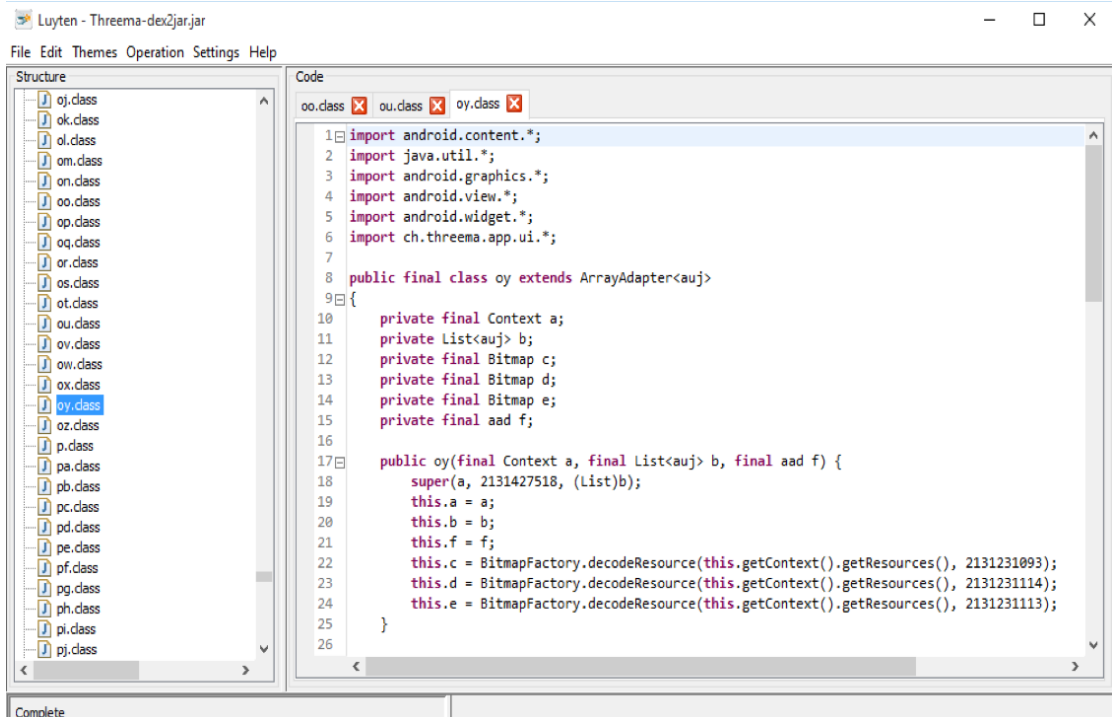


Figure 3.6: Extracted Code by Luyten De-Compiler

There is an option in both of the de-compilers to save all extracted sources, In this research all the apks been extracted and saved the source code at one place for analysis. As the extracted code of selected applications shown in Figure 3.7 now code analyzation can be done on these files.

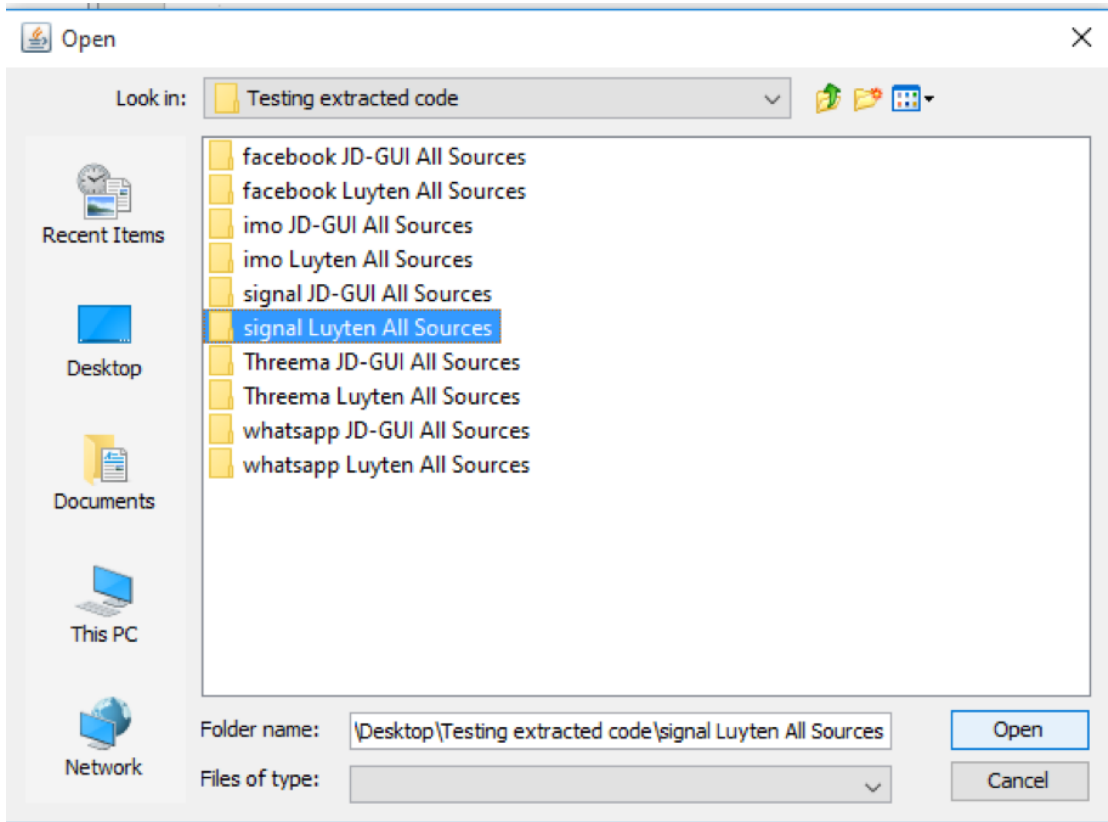


Figure 3.7: Extracted Code of Selected Apps (JD-GUI - Luyten)

Code Searching and Refinement

4.1 Introduction

In this section, we describe some of the previous approaches employed by researchers for detecting the useful code from the bulk. There are various methods in the literature having various strategies to identify the useful code. These can be roughly grouped into binary level code searching and high level code searching. Below, we briefly review the different approaches of these two categories. In the later part we also discuss our approach used in this research.

4.2 Others Code Searching Techniques

There is a research work which confronted a test in distinguishing the cryptographic features embedded in code by assuming that it is (for the most of the parts) not possible to analyse the entire functionalities of binary programs through static analysis. In this paper they present a novel methodology for distinguishing particular cryptographic algorithm through control flow examination based on symbolic execution. The creation of control flow graph and symbolic execution done by the angr framework[30]. In the proof of concept implementation they could recognize and separate DES, Triple-DES and a few variations of the AES. Their solution can recognize the presence of these calculations without access to the source code of the program.

There is another research introducing K-Hunt[31], which is working on Binary executables to spot in-secure keys. K-Hunt takes advantages from the properties on crypto operations for identifying the memory buffer(a place to store crypto keys).

One way of verifying cryptographic implementations is the use of verification tools such as Microsoft Crypto Verification Kit[24], Murphi[25] and others. The advantage of such tools is that they ensure strong guarantees but on the other hand they are heavy-weight, require expertise and a lot of manual effort. These limitations make such a tool unwanted for huge experiments and a software developer who has no knowledge of cryptographic mechanisms. Therefore, a lightweight yet effective approach for static analysis to identify common flaws. The tool used in this research, is named CryptoLint, build upon Androguard Android program analysis framework[23].

4.3 Developed tool for Code Searching

The searching method introduced in this research is to create a list of keywords that can be used in cryptography e.g. AES, ECC, ElGamal, MD5, SHA256, PRNG, CPRNG etc. Then there is a tool programmed in java language named as “Crypto Surveillance”.Using this tool, all the keywords listed in the keyword list will search in all the extracted java file and in return this tool tell us how many java files are there in the extracted folder and only the files in which crypto code potentially exist will be shown in table. the source code of this tool is also available on gitlab[32].

In figure 4.2 there is a flow chart of the tool used in this research named as “Crypto Surveillance”. the tool have two options one is with .java files and the other is with .class files.

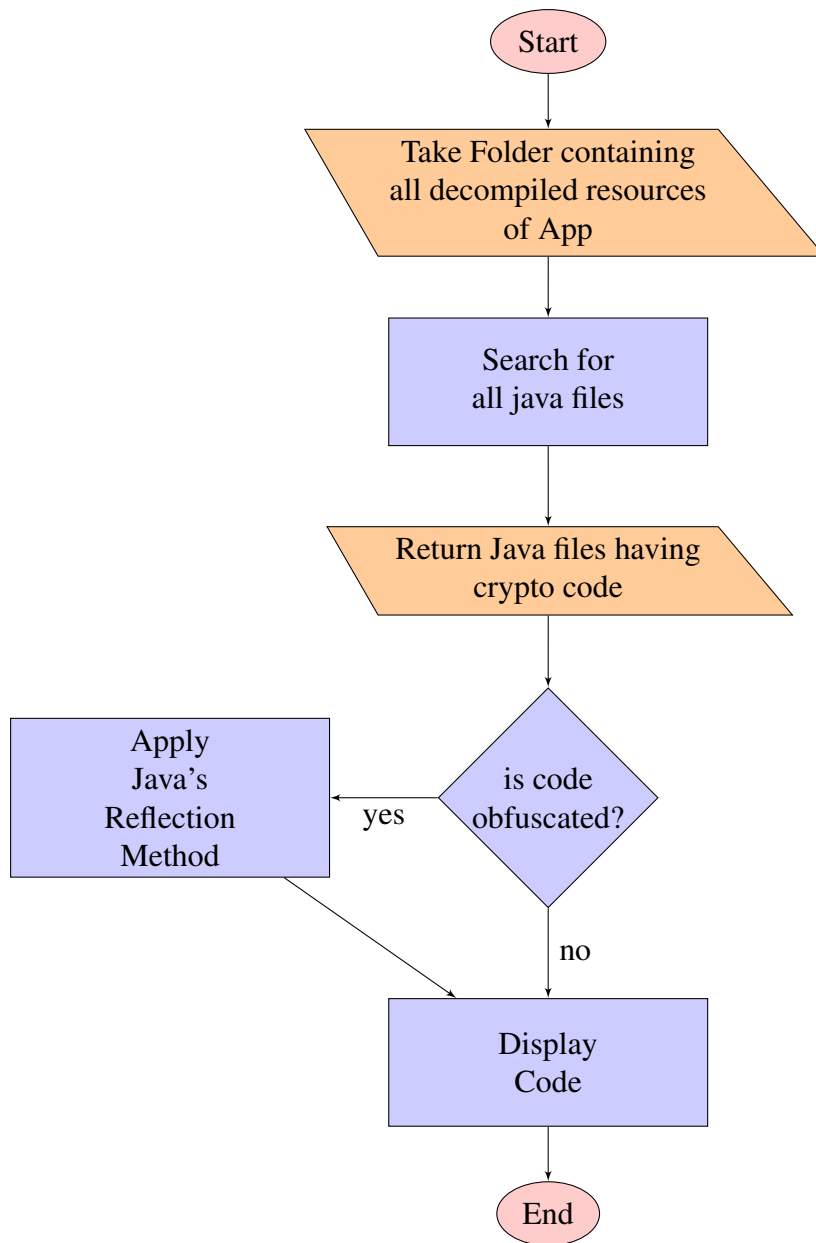


Figure 4.1: "Crypto-Surveillance" Flow Chart

When tested it on “Whatsapp” extracted folder 5865 java source code files were found shown in Figure 4.1.

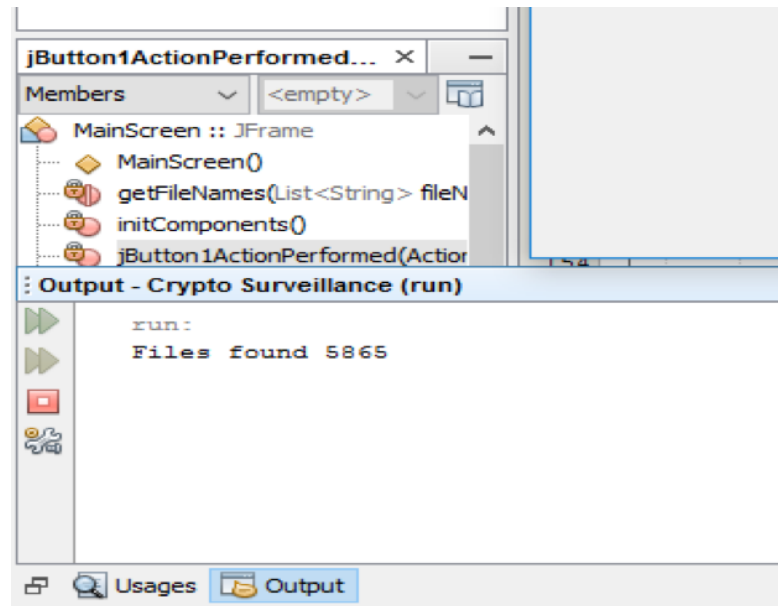


Figure 4.2: Java Files in Whatsapp Package

And when the same thing applied on “Signal” extracted files, 3632 java source code files were found as shown in Figure 4.2.

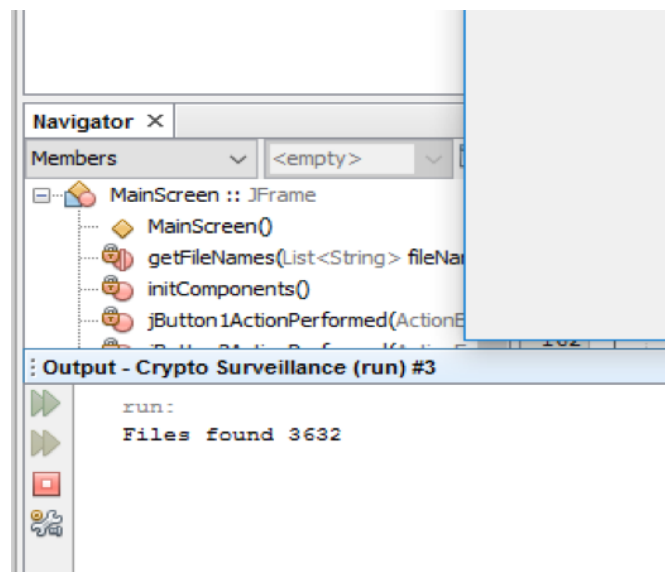


Figure 4.3: Java Files in Signal Package

As 5865 java files were found from Whatsapp apk and 3632 java files from Signal Apk which is not possible to read one by one in quick time. So this tool “Crypto Surveillance” will help us getting only the files in which there is a chance of crypto code. So first the tool will ask to select a folder in which all the files related to an app placed.

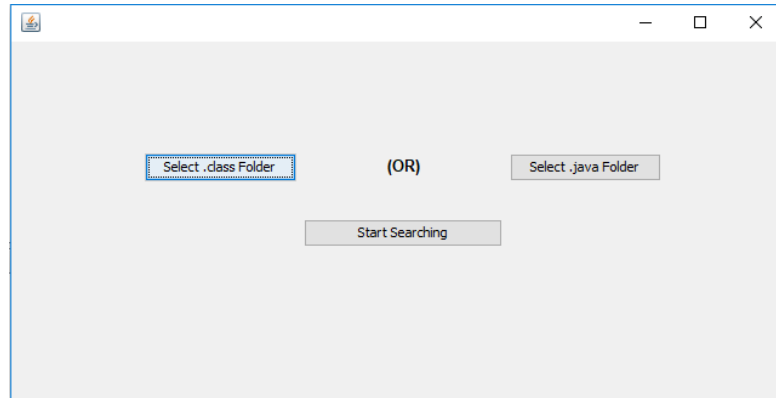


Figure 4.4: Crypto Surveillance Tool UI

here the data set containing some app sources that were extracted by apktool can be seen in Figure 4.4.

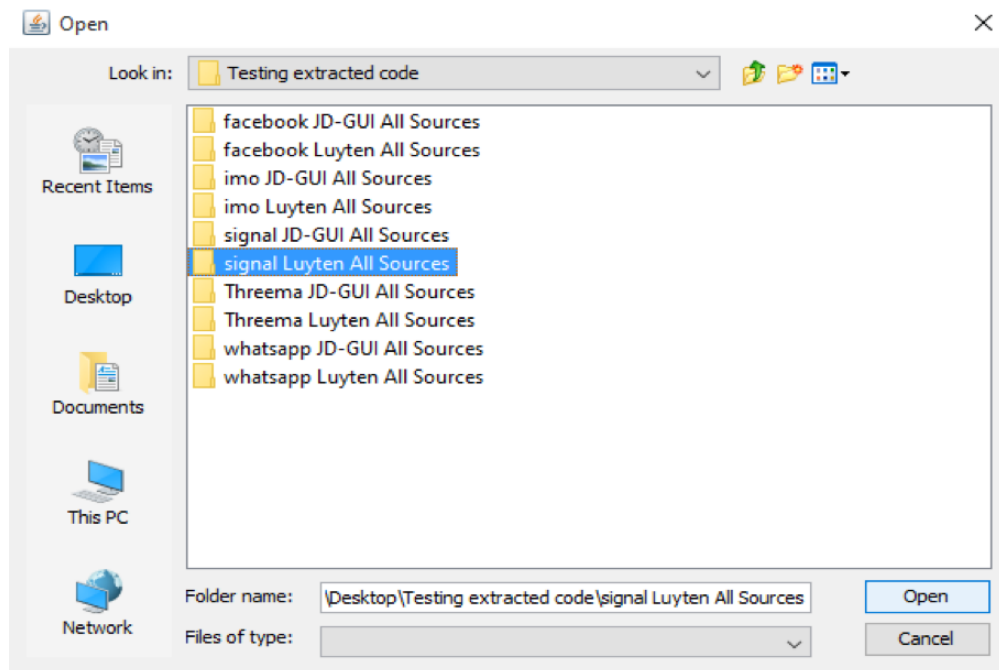


Figure 4.5: All Selected Applications Source Folders

After providing the root folder to this tool here it can be seen that the tool will prompt about the number of java files found as seen in Figure 4.5.

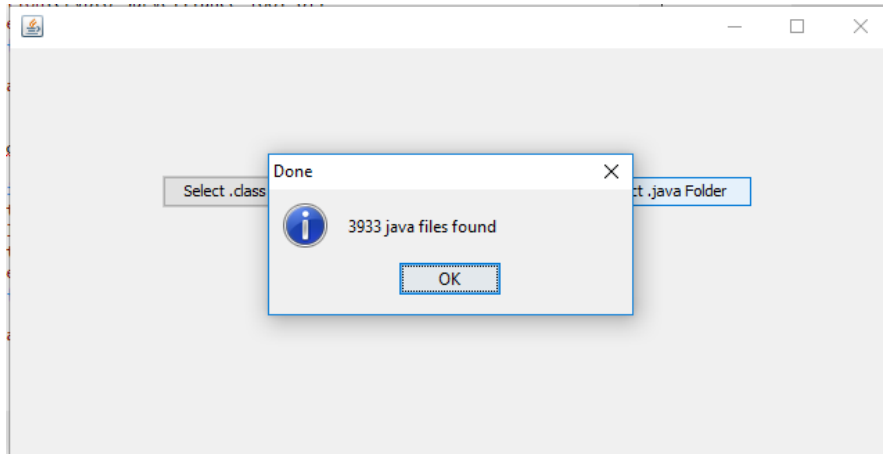


Figure 4.6: All Java Files from Package Selected

In the Figure 4.6, For “Signal” app after selecting the files it will display them in the form of table. Here it can be seen in class “AEngine.java” the keyword AES in the function “decrypt block” so we can have a guess that it is using AES for encryption/decryption process.

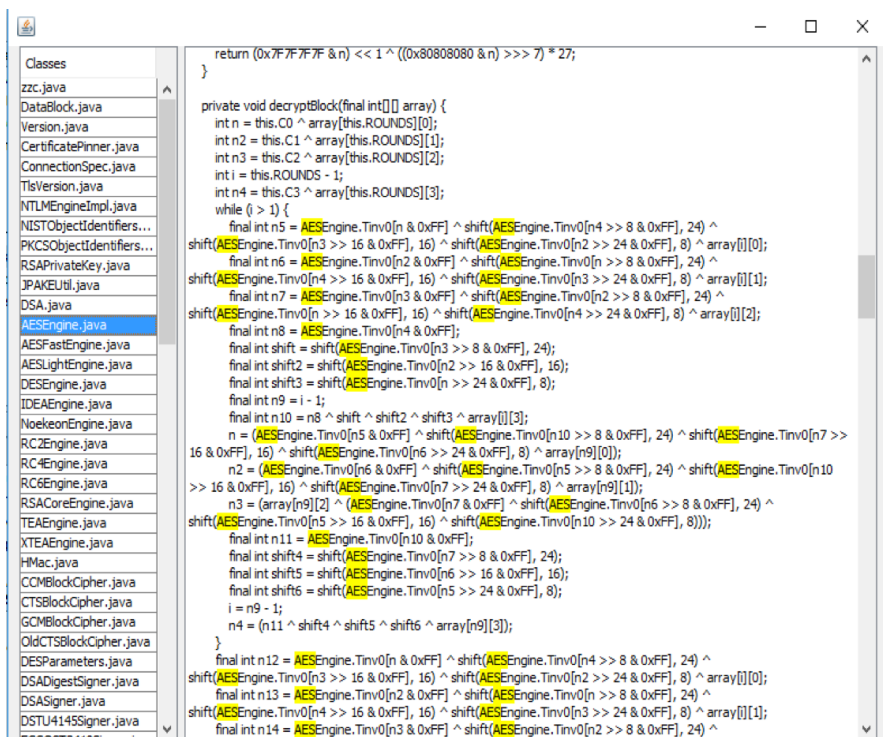


Figure 4.7: "AEngine" Class from Signal Code

```

package org.spongycastle.crypto.engines;

import java.lang.reflect.*;
import org.spongycastle.crypto.params.*;
import org.spongycastle.crypto.*;

public class AESEngine implements BlockCipher
{
    private static final int BLOCK_SIZE = 16;
    private static final byte[] S;

```

Figure 4.8: Imports in AESEngine Class in Signal

In the figure 4.7 At the top of "AESEngine" class where we include some classes/libraries that we have to use in particular class. Here in the Signal app "AESEngine" class, it can be seen a package named as spongycastle[33] which is open source and publically available on github[34].

Same is the case in "Whatsapp" it can be seen that the package imported in one to the class named "AES" which shows that both the apps are using same cryptographic library i.e. spongycastle as shown in figure 4.8.

```

package org.spongycastle.jcajce.provider.symmetric;

import java.security.AlgorithmParameters;
import java.security.InvalidAlgorithmParameterException;
import java.security.SecureRandom;
import java.security.spec.AlgorithmParameterSpec;
import javax.crypto.spec.IvParameterSpec;
import org.spongycastle.crypto.BlockCipher;
import org.spongycastle.crypto.CipherKeyGenerator;
import org.spongycastle.crypto.engines.AESFastEngine;
import org.spongycastle.jcajce.provider.config.ConfigurableProvider;
import org.spongycastle.jcajce.provider.symmetric.util.BaseAlgorithmParameterGenerator;
import org.spongycastle.jcajce.provider.symmetric.util.BaseBlockCipher;
import org.spongycastle.jcajce.provider.symmetric.util.BaseKeyGenerator;
import org.spongycastle.jcajce.provider.symmetric.util.BlockCipherProvider;
import org.spongycastle.jcajce.provider.symmetric.util.IvAlgorithmParameters;
import org.spongycastle.jcajce.provider.util.AlgorithmProvider;

public final class AES
{
    public static class AlgParamGen
        extends BaseAlgorithmParameterGenerator
    {

```

Figure 4.9: Imports in AES Class in Whatsapp

The bundle “spongycastle” is used in both the app for security related tasks and this package contains a light weight API works in any condition or environment with the additional infrastructure to conform the algorithms to the Java Cryptographic Extension (JCE) framework[35].

4.4 Java Reflection API

Reflection is normally used in programs to read and adjust the runtime behavior of application while running in JVM. This is an advanced feature and used only by programmers having a strong knowledge of the particular language. Reflection is a powerful technique it can enable applications for performing such operations which is not possible in any other way. It is often used as part of software testing, such as for the runtime creation of mock objects. Reflection is also a key technique for metaprogramming[36]. In some object oriented programming languages like C-Sharp and Java, reflection can be used to override member accessibility rules. using Reflection in Java, it is possible to inspect Attributes, Classes, Functions, Interfaces, Constructors, Getter, Setter, Enum, Collections, Arrays etc

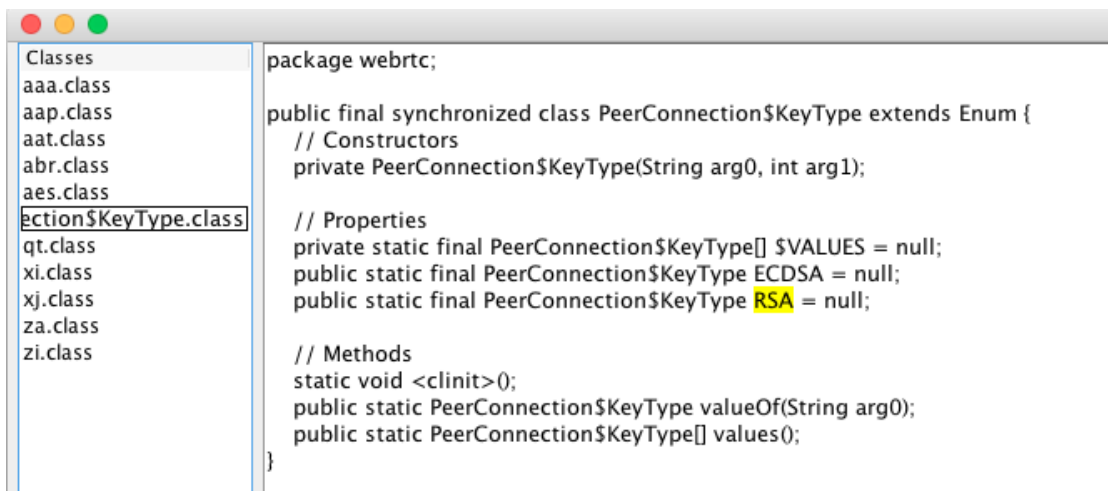
4.4.1 Reflection Use cases

Despite the limitations, security researcher and tester uses reflection because it is very powerful tool that can be useful in several use-cases.

- 1) Reflection is used in debuggers to inspect dynamic behaviour of code.
- 2) Reflection is used in some of test tools like Junit or Mockito for calling methods containing specific syntax.
- 3) External tools which utilize code dynamically, may use reflection.
- 4) Reflection is used in code analysis tools like PMD or Findbugs, to analyze the code against the code violations list that were found earlier.

4.4.2 Use of reflection in code identification

In this research reflection API is used to get original class name, class constructors, properties and method signature (method name, return type, input parameters and any exceptions that a function may throws) and any class or interface that a given class overrides or implements. Name of crypto-libray used in a particular app can be found easily using reflection method. Here is the result of our tool after applying reflection method to .class files on one of our app.



```
Classes
aaa.class
aap.class
aat.class
abr.class
aes.class
PeerConnection$KeyType.class
qt.class
xi.class
xj.class
za.class
zi.class

package webrtc;

public final synchronized class PeerConnection$KeyType extends Enum {
    // Constructors
    private PeerConnection$KeyType(String arg0, int arg1);

    // Properties
    private static final PeerConnection$KeyType[] $VALUES = null;
    public static final PeerConnection$KeyType ECDSA = null;
    public static final PeerConnection$KeyType RSA = null;

    // Methods
    static void <clinit>();
    public static PeerConnection$KeyType valueOf(String arg0);
    public static PeerConnection$KeyType[] values();
}
```

Figure 4.10: Reflection Method Output

4.5 Conclusion

This chapter highlights some of previous code searching techniques and few tools such as K-Hunt and Murphi have been discussed. In our research we build a tool named as "Crypto Surveillance" which can search through thousands of files and in return will provide only the files having crypto code in it. The other technique used is Java Reflection API. we used this API to get some important parts of code from .class files.

EXPLORATION OF ENCRYPTION KEY

5.1 Introduction

This is the most important chapter of this research work as it describes the way of analyze the keys generation mechanism in our selected applications. There are various methods or functions in the sequence, calling one another to generate the key. we discuss about the key and the other parameters on which the key is dependent. first we discuss about Whatsapp and Signal application and then in the later part we discuss about IMO and Threema applications.

5.2 Whatsapp and Signal

In the previous chapter we have seen that how we can get the apk file of an application and also how can we get our hands directly to the crypto code when we have lines of code in thousands. Now we are looking for the key and its generation mechanism.

In the previous chapter “spongycastle” was the package name imported in the classes of both apps, which shows that both the apps are using same cryptographic library. So any one of the apps can be tested for getting to know about its key generation mechanism

and encryption and decryption ways.

“Signal” application been chosen for verification because its open source and code can be easily verified, the source code is available on Github and compiled after downloading using android studio to run on device for debugging. its important to resolve few dependencies in the source code in order to compile and run the application.

Signal app contain various screens (Activities) but we were interested in send Message screen. This screen contains a message typing area and a send button on the action of this button we got a method named as “send” which take plain text message and get its recipient and also check for the key that its already exchanged or not in the figure 5.1.

```
package org.thoughtcrime.securesms.sms;
import ...
public class MessageSender {
    private static final String TAG = MessageSender.class.getSimpleName();

    public static long send(final Context context,
        final OutgoingTextMessage message,
        final long threadId,
        final boolean forceSms,
        final SmsDatabase.InsertListener insertListener) {
        SmsDatabase database = DatabaseFactory.getSmsDatabase(context);
        Recipient recipient = message.getRecipient();
        boolean keyExchange = message.isKeyExchange();

        long allocatedThreadId;
        if (threadId == -1) {
            allocatedThreadId = DatabaseFactory.getThreadDatabase(context).getThreadIdFor(recipient);
        } else {
            allocatedThreadId = threadId;
        }

        long messageId = database.insertMessageOutbox(allocatedThreadId, message, forceSms, System.currentTimeMillis(), insertListener);
        sendTextMessage(context, recipient, forceSms, keyExchange, messageId, message.getExpiresIn());

        return allocatedThreadId;
    }
}
```

Figure 5.1: Message Send Function Against the “send” Button Click

```
public boolean isKeyExchange() {
    return false;
}
```

Figure 5.2: Key Exchange Function in Signal Application

As in the Figure 5.2, it was noticed that the method "isKeyExchange()" always returns false and each time the "send" button is pressed a new key exchange process would occurs.

In the Figure 5.1, it can be noticed that "sendTextMessage" is called which in turn calls sendTextPush message after validating that message had already not been sent.

```
private static void sendTextPush(Context context, Recipient recipient, long messageId) {
    JobManager jobManager = ApplicationContext.getInstance(context).getJobManager();
    jobManager.add(new PushTextSendJob(context, messageId, recipient.getAddress()));
}
```

Figure 5.3: "sendTextMessage" Function in Signal Application

In the above method a "PushTextSendJob" object was added in the jobManager queue. PushTextSendJob class contain a deliver method shown in Figure 5.4 that gets called when a message is to be sent from the messages queue.

```
private void deliver(SmsMessageRecord message)
    throws UntrustedIdentityException, InsecureFallbackApprovalException, RetryLaterException {
    try {
        1 SignalServiceAddress address = getPushAddress(message.getIndividualRecipient().getAddress());
        2 Optional<byte[]> profileKey = getProfileKey(message.getIndividualRecipient());
        3 SignalServiceDataMessage textSecureMessage = SignalServiceDataMessage.newBuilder()
            .withTimestamp(message.getDateSent())
            .withBody(message.getBody())
            .withExpiration((int) (message.getExpiresIn() / 1000))
            .withProfileKey(profileKey.orNull())
            .asEndSessionMessage(message.isEndSession())
            .build();
        4 messageSender.sendMessage(address, textSecureMessage);
    } catch (UnregisteredUserException e) {
        Log.w(TAG, e);
        throw new InsecureFallbackApprovalException(e);
    } catch (IOException e) {
        Log.w(TAG, e);
        throw new RetryLaterException(e);
    }
}
```

Figure 5.4: "deliver" Function in Signal Application

Deliver method do following things

- 1) Get recipient address.
- 2) Get profile key
- 3) Create a SignalServiceDataMessage object from the message object received from database.
- 4) Pass the message and address to sendMessage method of MessageSender class.

Since key generation mechanism is the important part in this research therefore we are going to describe how a profile key is generated. Here the profile key is returned from getProfileKey method from ProfileKeyUtil class.

This method first tries to fetch the profile key from TextSecurePreferences (if present) else it create a new profile key, save it into TextSecurePreferences and then return the newly generated profile key in base64 format. In the Figure 5.5 it can be seen that the actual key is not in a readable format this is because randomly 32 bytes (256 bits)

were selected as shown in Figure 5.6. so thatswhy we prefer the key in base64 encoded format.

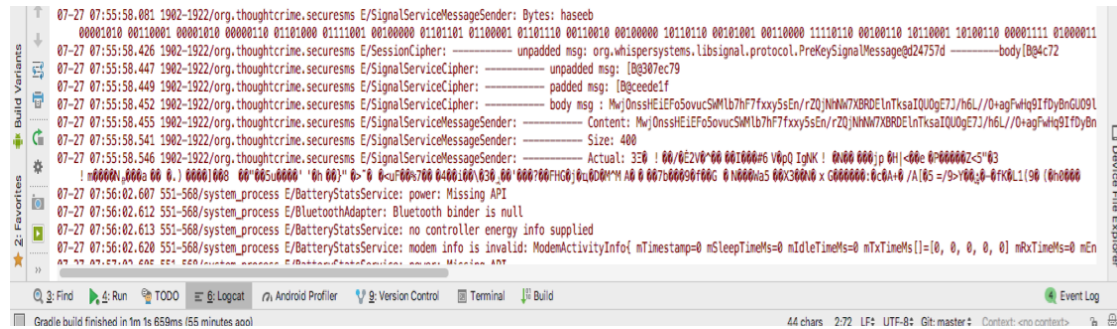


Figure 5.5: Un-Readable Form of Actual Key

```
public static byte[] getSecretBytes(int size) {
    byte[] secret = new byte[size];
    getSecureRandom().nextBytes(secret);
    return secret;
}

public static SecureRandom getSecureRandom() {
    return new SecureRandom();
}
```

Figure 5.6: Secure Random Secret Bytes

Next some function calls were skipped that start from function `getEncryptedMessages()` which calls `getEncryptedMessage()`, that will create a cipher object of class "SignalServiceCipher" and from here is our next point of interest that can be seen in the Figure 5.7.

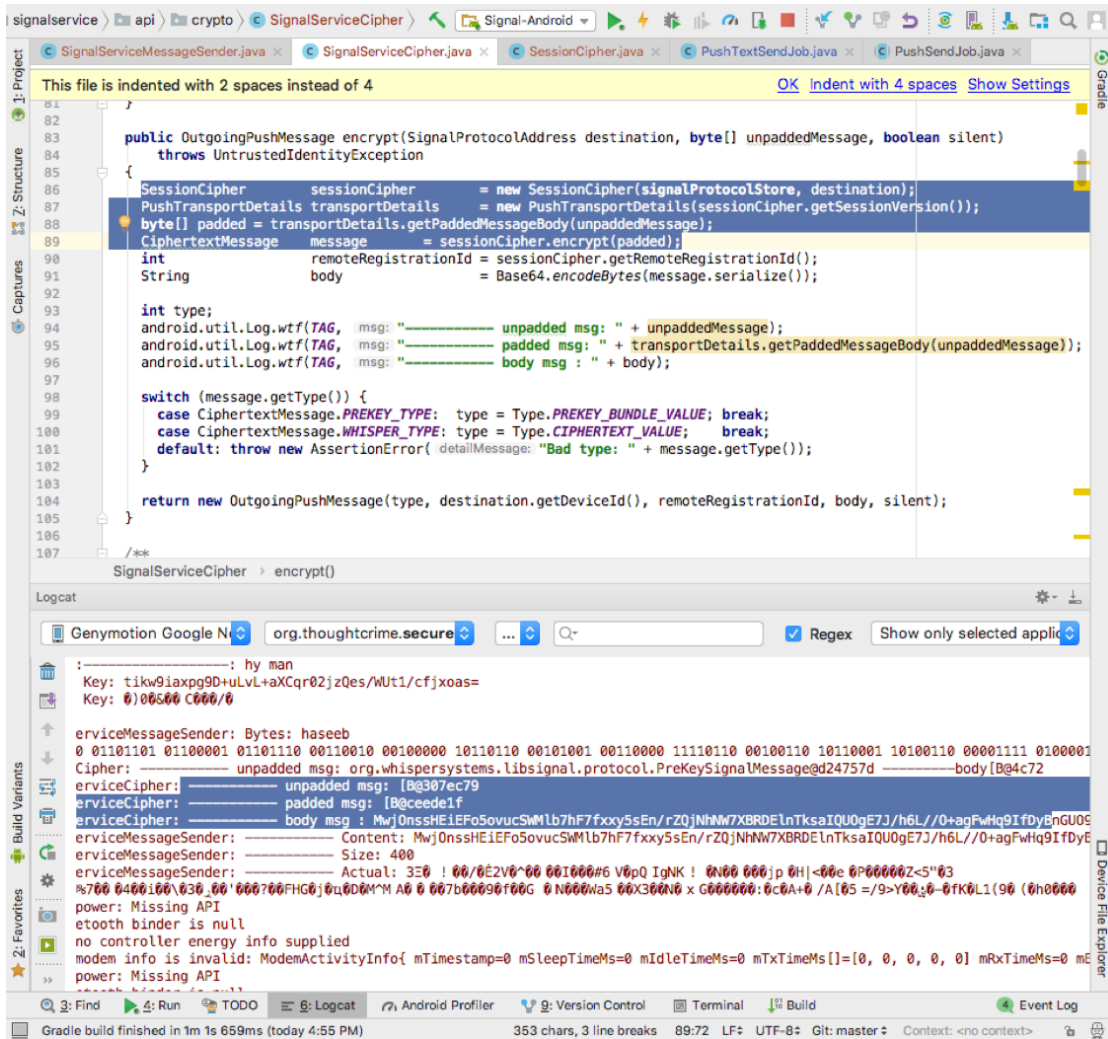


Figure 5.7: Padded Message and Encrypted Message Body

First, the SessionCipher object was created. It is the main entry point for encrypt/decrypt operations in signal protocol, next PushTransportDetails object were created using the session cipher object. PushTransportDetails is responsible for converting the unpadded message body into padded message body. Finally the padded message body been encrypted using the SessionCipher object as shown in Figure 5.8.

```

95 public CiphertextMessage encrypt(byte[] paddedMessage) throws UntrustedIdentityException {
96     synchronized (SESSION_LOCK) {
97         1 SessionRecord sessionRecord = sessionStore.loadSession(remoteAddress);
98         2 SessionState sessionState = sessionRecord.getSessionState();
99         3 ChainKey chainKey = sessionState.getSenderChainKey();
100        4 MessageKeys messageKeys = chainKey.getMessageKeys();
101        5 ECPublicKey senderEphemeral = sessionState.getSenderRatchetKey();
102        int previousCounter = sessionState.getPreviousCounter();
103        int sessionVersion = sessionState.getSessionVersion();
104
105        6 byte[] ciphertextBody = getCiphertext(messageKeys, paddedMessage);
106        7 CiphertextMessage ciphertextMessage = new SignalMessage(sessionVersion, messageKeys.getMacKey(),
107            senderEphemeral, chainKey.getIndex(),
108            previousCounter, ciphertextBody,
109            sessionState.getLocalIdentityKey(),
110            sessionState.getRemoteIdentityKey());
111
112        if (sessionState.hasUnacknowledgedPreKeyMessage()) {
113            UnacknowledgedPreKeyMessageItems items = sessionState.getUnacknowledgedPreKeyMessageItems();
114            int localRegistrationId = sessionState.getLocalRegistrationId();
115
116            ciphertextMessage = new PreKeySignalMessage(sessionVersion, localRegistrationId, items.getPreKeyId(),
117                items.getSignedPreKeyId(), items.getBaseKey(),
118                sessionState.getLocalIdentityKey(),
119                (SignalMessage) ciphertextMessage);
120        }
121
122        8 sessionState.setSenderChainKey(chainKey.getNextChainKey());
123
124        if (!identityKeyStore.isTrustedIdentity(remoteAddress, sessionState.getRemoteIdentityKey(), IdentityKeyStore.Dir
125            ) throw new UntrustedIdentityException(remoteAddress.getName(), sessionState.getRemoteIdentityKey());
126        }
127
128        identityKeyStore.saveIdentity(remoteAddress, sessionState.getRemoteIdentityKey());
129        9 sessionStore.storeSession(remoteAddress, sessionRecord);
130        // Thats what we need to explore
131        android.util.Log.wtf(TAG, msg: "-----unpadded msg: " + ciphertextMessage + " -----body"+ ciphertextB
132        10 return ciphertextMessage;
133    }
134 }

```

Figure 5.8: "encrypt" Method of "CiphertextMessage" Class

Below is the explanation of the code mentioned in Figure 5.8:

- 1) Retrieve the current session in a SessionRecord object. A SessionRecord object maintained the state of an ongoing session.
- 2) Get the state of the current session from the SessionRecord object.
- 3) Get the sender's ChainKey from the session state.
- 4) Get MessageKeys from ChainKey. MessageKey is composed of a cipherKey, macKey and an IV(initialization vector).
- 5) Get a one time use only Ephemeral key from the session state.
- 6) Get the cipher text for the given padded message using MessageKeys.
- 7) Create a CipherTextMessage object using chain key, message keys, sender ephemeral, cipher text and the session state.
- 8) Set the next chain key into the session state.
- 9) Store the session state back into the session record.
- 10) Return the CipherTextMessage object.

```

erviceCipher: ----- unpadded msg: [B@307ec79
erviceCipher: ----- padded msg: [B@ceede1f
erviceCipher: ----- body msg : Mwj0nssHEiEFo5ovucSM1b7hF7fxy5sEn/rZQjNhNw7XBRDElnTksaIQU0gE7J/h6L//0+agFwHq9IfDyBnGU0!
erviceMessageSender: ----- Content: Mwj0nssHEiEFo5ovucSM1b7hF7fxy5sEn/rZQjNhNw7XBRDElnTksaIQU0gE7J/h6L//0+agFwHq9IfDyBnGU0!
erviceMessageSender: ----- Size: 400
erviceMessageSender: ----- Actual: 3E0 !00/0E2V0*00 00I000#6 V0pQ IgNK ! 0N00 000jp 0Hl<00e 0P00000Z<5"03
%700 0400i00\030_00'000?00FHG0j0u0D0M'M A0 0 007b00090f00G 0 N000Na5 00X300N0 x G000000:0c0A+0 /A[05 =/9>Y00;0-0fK0L1(90 (0h0000

```

Figure 5.9: Final Shape of Message to be Send

Figure 5.9 illustrates the final form of message to be send. The message is first padded, then encrypted and finally encoded into base64 format before being sent. On the receiver end the reverse of that process is done to get a readable message. This protocol is known as Signal Protocol the details of signal protocol are mentioned in next chapter.

5.2.1 Results (Signal and Whatsapp)

Signal protocol is defined by Open Whisper Systems to ensure anonymity and secrecy of messages. The general philosophy behind the signal protocol is to send encrypted messages through one server, while keys are maintained and transferred using a separate key distribution center (Server). Since the messages are encrypted and their keys are unknown therefore they cannot be decrypted at server end. And since the keys are sent through the key distribution center without the information about the message, therefore they are useless without a message. This makes signal protocol flawless and hard to break.

The first step is to generate a set of long term identity key pairs, medium term signed key pair, and various other ephemeral pre-key pairs on the client side. Next the public keys (long term and medium term) are bundled along with registration Id (usually mobile number) and are sent to the key distribution center(Server). For example, if Alice wants to communicate with Bob then she must first register herself to key distribution center, and also should know Bob's registration ID and public keys.

Alice receives Bob's public keys (long term and short term) along with the ephemeral key. The ephemeral key is removed from server since it is a one-time use key. Alice generates ephemeral Curve25519 key pair. Alice then calculate a master secret using the ECDH of Bob's public keys and her own keys. Alice finally creates a root key and chain key from the master secret using HKDF. Alice then send this master shared secret

to Bob for validation. Upon successful validation Alice and Bob can send messages to each other. Signal protocol uses X3DH key Agreement that provides forward secrecy and cryptographic deniability along with asynchronicity. This allows Alice and Bob to send off-line messages. Each created message results in the creation of a new set of ephemeral keys that are used to encrypt/decrypt the next message. During a session, Alice sends encrypted messages using the master shared secret along with the previously generated ephemeral key. This create a root key, a chain key and a message chain.

Session Key Types	Description
Root Key	A 32 byte key for creating Chain Keys.
Chain Key	A 32 byte key for creating Message Keys.
Message Key	Message contents encryption (80 byte), AES-256 key (32-bytes), HMAC-SHA256 key (32 bytes) and IV (16-bytes)

Table 5.1: Session Key Types

Public Key Types	Description
Identity-Key Pair	A long-term Curve25519 key pair, generated at install time.
Signed Pre Key	A medium-term Curve25519 key pair, generated at install time, signed by the Identity Key, and rotated on a periodic timed basis.
One-Time Pre Keys	A queue of Curve25519 key pairs for one time use, generated at install time, and replenished as needed

Table 5.2: Public Key Types

X3DH (Key Agreement Protocol):

This thing happens, by generating all the necessary keys between two communicating parties. It establishes the crucial shared secret key between the two parties who mutually authenticate each other based on their public key pairs. X3DH also allows for key exchange to occur where one party is “offline”, and will instead exchange it through a third party server. X3DH involves 3 primary parties: Sender, Receiver, and Server.

X3DH has 3 phases: Sender registers his identity key and prekeys to a server, second is Receiver retrieves Sender’s prekeys bundle from the server uses it to start a session then send an initial message to Sender, and last one is Sender receives and decrypts Receiver's message.

5.3 IMO

Crypto Surveillance[32] found 5450 source code files when searched for java files in IMO packages shown in Figure 5.10.

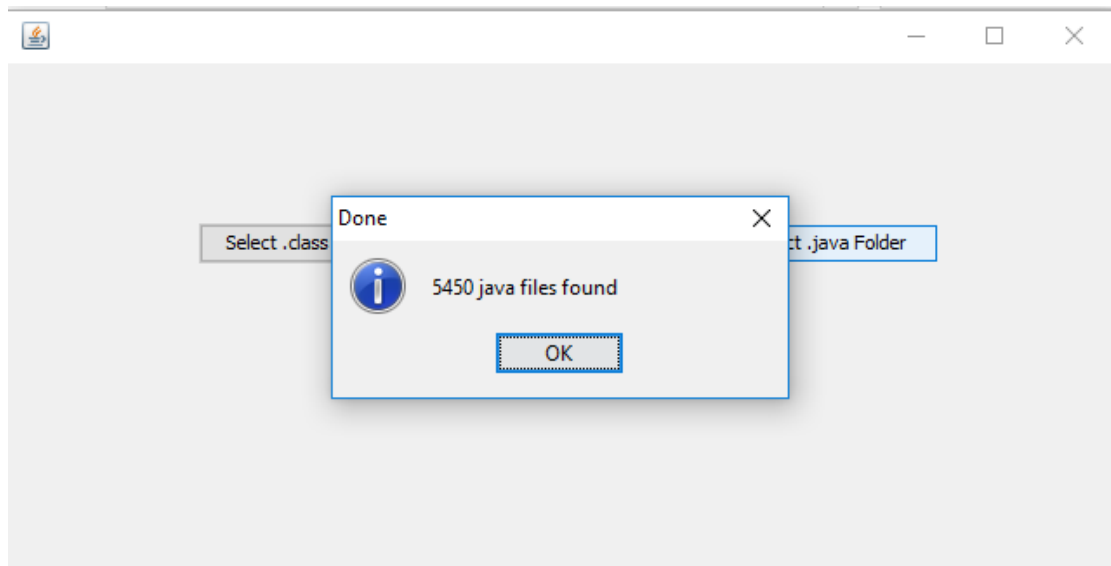


Figure 5.10: All Java Files in IMO Package

Only 11 files shown in Figure 5.11 found that were potentially having crypto code in them. The highlighted words found out in the code were mainly Cast, AES, UTF-8 and SSL. In cryptography, CAST-128 (Created in 1996) is a symmetric key block cipher used in a number of products. Additionally It has been approved to use for Government of Canada by the Communications Security Establishment[37]. but here in the IMO package its not used in cryptographic context, it was just an exception message of type "java.lang.ClassCastException".

```

final class c
{
    @VisibleForTesting
    static final c h;
    View a;
    MediaLayout b;
    TextView c;
    TextView d;
    ImageView e;
    TextView f;
    ImageView g;

    static {
        h = new c();
    }

    static c a(final View a, final MediaViewBinder mediaViewBinder) {
        final c c = new c();
        c.a = a;
        try {
            c.c = (TextView)a.findViewById(mediaViewBinder.c);
            c.d = (TextView)a.findViewById(mediaViewBinder.d);
            c.f = (TextView)a.findViewById(mediaViewBinder.e);
            c.b = (MediaLayout)a.findViewById(mediaViewBinder.b);
            c.e = (ImageView)a.findViewById(mediaViewBinder.f);
            c.g = (ImageView)a.findViewById(mediaViewBinder.g);
            return c;
        }
        catch (ClassCastException ex) {
            MoPubLog.w("Could not cast from id in MediaViewBinder to expected View type", ex);
            return com.mopub.nativeads.c.h;
        }
    }
}

```

Figure 5.11: All Crypto-related files in IMO Package

```

package com.google.android.gms.internal;

import java.util.*;

final class aes<FieldDescriptorType extends aeu<FieldDescriptorType>>>
{
    private static final aes d;
    final agj<FieldDescriptorType, Object> a;
    private boolean b;
    private boolean c;

    static {
        d = new aes((byte)0);
    }

    private aes() {
        this.c = false;
        this.a = agj.a(16);
    }

    private aes(final byte b) {
        this.c = false;
        this.a = agj.a(0);
        if (!this.b) {
            this.a.a();
            this.b = true;
        }
    }

    private static int a(final ahf ahf, int f, final Object o) {
        final int n = f = ael.f(f);
        if (ahf == ahf.j) {
            aey.a();
            f = n << 1;
        }
        return f + b(ahf, o);
    }
}

```

Figure 5.12: "AES" class in IMO application

The class shown in Figure 5.12 is declared in "com.google.android.gms.internal" package. GMS stands for Google Mobile Services[38], which is the set of apps that come pre-installed with any android device. The code in the class as shown in Figure 5.12 is the encryption code of google which is used for google's own mobile services. so it can easily justify that there is not even single line of cryptographic code exists in IMO mobile application.

5.3.1 Results (IMO)

As its proved in previous chapter that IMO hasn't implemented any sort of cryptography or security in its main and core part i.e. messaging or communication. Unlike Signal and Whatsapp, who have adopted recent trends in cryptography to make their messaging and communication secure.

5.4 CONCLUSION AND FUTURE DIRECTIONS

This section has concluded the research work by providing a brief overview of the research conducted. It has given a sketch of the findings from this research. Furthermore it has set future directions for the researchers in the fields of Information Technology, Information Security and Programming.

5.4.1 Conclusion

Applictaions properties result			
	Signal	Whatsapp	IMO
Cryptographic-Library	SpongyCastle	SpongyCastle	nil
Algorithm	AES-256	AES-256	nil
mode of encryption	CBC with padding(PKCS#5) and CTR without padding	CBC mode with PKCS #5 padding scheme and CTR mode without padding	nil
Hashing Algo	SHA 256	SHA 256	nil
padding scheme	PKCS #5 (mode CBC)	PKCS #5 (mode CBC)	nil
keys and their sizes	Root-Key(32bytes), Chain-Key(32bytes), Message-Key(80bytes)	Root-Key(32bytes), Chain-Key(32bytes), Message-Key(80bytes)	nil
Initialization Vector	16 bytes Random IV	16 bytes Random IV	nil
Encoding format	base64	base64	utf-8

Table 5.3: Comparison of Application

5.4.2 Future Directions

This research has provided open research areas for future researchers as there is still room for further research in this field. Following future directions are provided to the researchers as a result of this research work.

- 1) Need to identify new code analyzing techniques to ensure that application is secure and upto the latest standards of cryptography.
- 2) Apply the same practice to application developed for Desktop, Web and other mobile platforms.

References

- [1] Pierre Lestringant, Frédéric Guihéry, and Pierre-Alain Fouque. Automated identification of cryptographic primitives in binary code with data flow graph isomorphism. In Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, pages 203–214. ACM, 2015.
- [2] Re-tools. <https://resources.infosecinstitute.com/top-8-reverse-engineering-tools-cyber-security-professionals/{#}gref>. Accessed: 2018-11-05.
- [3] Recommended-algos. <https://sushi2k.gitbooks.io/the-owasp-mobile-security-testing-guide/content/0x04g-Testing-Cryptography.html>. Accessed: 2018-11-05.
- [4] poly1305. <https://en.wikipedia.org/wiki/Poly1305>. Accessed: 2018-11-05.
- [5] Sha-256. <https://www.movable-type.co.uk/scripts/sha256.html>. Accessed: 2018-11-05.
- [6] Mobile future. [comScore Inc. 2012 Mobile Future in Focus, February 2012](#). Accessed: 2018-11-05.
- [7] Anshul Arora, Shree Garg, and Sateesh K Peddoju. Malware detection using network traffic analysis in android based mobile devices. In Next generation mobile apps, services and technologies (NGMAST), 2014 eighth international conference on, pages 66–71. IEEE, 2014.

- [8] Java-devices. https://www.google.com.pk/search?q=According+to+Oracle,+Java+currently+installed+on+more+than+3+billion+devices.&rlz=1C1CHBD_enPK799PK799&tbm=isch&source=lnms&sa=X&ved=0ahUKEwjy_9Wsm9PeAhUhSY8KHTZ1APMQ_AUICygC&biw=1366&bih=657&dpr=1#imgrc=uLe3ca6qRjDmXM:, . Accessed: 2018-11-05.
- [9] Android-java. <https://www.developer.com/java/j2me/java-mobile-programming-for-android.html>, . Accessed: 2018-11-19.
- [10] Android-sdk. <https://stuff.mit.edu/afs/sipb/project/android/docs/sdk/index.html>, . Accessed: 2018-11-05.
- [11] Android-runtime-art. https://en.wikipedia.org/wiki/Android_Runtime. Accessed: 2018-11-05.
- [12] Google-play. <https://play.google.com/store?hl=en>. Accessed: 2018-11-05.
- [13] Softwares-powered-by-java. <https://www.linkedin.com/pulse/12-examples-popular-software-powered-java-doug-purcell>. Accessed: 2018-11-05.
- [14] Ravindar Reddy Ravula. Classification of Malware using Reverse Engineering and Data Mining Techniques. PhD thesis, University of Akron, 2011.
- [15] Nugache-worm. <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Worm:Win32/Nugache.F>. Accessed: 2018-11-05.
- [16] Sam Stover, Dave Dittrich, John Hernandez, and Sven Dietrich. Analysis of the storm and nugache trojans: P2p is here. USENIX; login, 32(6):18–27, 2007.
- [17] Chien-Chung Chan and Santhosh Sengottayan. Blem2: Learning bayes' rules from examples using rough sets. In Fuzzy Information Processing Society, 2003. NAFIPS 2003. 22nd International Conference of the North American, pages 187–190. IEEE, 2003.

- [18] Firmware. <https://en.wikipedia.org/wiki/Firmware>. Accessed: 2018-11-05.
- [19] Hex-rays(ida). <https://www.hex-rays.com/products/ida/>. Accessed: 2018-11-05.
- [20] Radare2. <https://rada.re/r/>. Accessed: 2018-11-05.
- [21] Binary-ninja. <https://binary.ninja/purchase/>. Accessed: 2018-11-05.
- [22] Mallodroid. <https://github.com/sfahl/mallodroid>. Accessed: 2018-11-05.
- [23] Androguard. <https://github.com/androguard/androguard>, . Accessed: 2018-11-05.
- [24] Crypto-verification-kit. <https://docs.microsoft.com/en-us/windows/desktop/seccrypto/cryptography-tools>, . Accessed: 2018-11-05.
- [25] Murphi verification system. <http://seclab.stanford.edu/pcl/mc/mc.html>. Accessed: 2018-11-05.
- [26] Cryptolint. <https://sgros-students.blogspot.hk/2017/03/cryptolint.htm>, . Accessed: 2018-11-05.
- [27] dex2jar. <https://www.nccgroup.trust/us/about-us/newsroom-and-events/blog/2010/october/decompiling-android-apps-undx-dex2jar-and-smali/>. Accessed: 2018-11-05.
- [28] Jd-gui. <http://jd.benow.ca/>. Accessed: 2018-11-27.
- [29] Luyten. <https://github.com/deathmarine/Luyten>. Accessed: 2018-11-27.
- [30] Angr framework. <https://github.com/angr/angr>. Accessed: 2018-11-05.
- [31] Juanru Li, Zhiqiang Lin, Juan Caballero, Yuanyuan Zhang, and Dawu Gu. K-hunt: Pinpointing insecure cryptographic keys from execution traces. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pages 412–425. ACM, 2018.

- [32] Crypto-surveillance. <https://gitlab.com/Haseebjaved/crypto-surveillance>, . Accessed: 2018-11-05.
- [33] Spongycastle. [SpongyCastle : https://github.com/rtyley/spongycastle](https://github.com/rtyley/spongycastle). Accessed: 2018-11-05.
- [34] Github. <https://github.com/>. Accessed: 2018-11-05.
- [35] Java-cryptographic-extension-(jce). https://en.wikipedia.org/wiki/Java_Cryptography_Extension, . Accessed: 2018-11-05.
- [36] Metaprogramming. <https://en.wikipedia.org/wiki/Metaprogramming>. Accessed: 2018-11-12.
- [37] Cast-128. <https://en.wikipedia.org/wiki/CAST-128>. Accessed: 2018-11-05.
- [38] Gms. <https://www.android.com/gms/>. Accessed: 2018-11-05.