Smart Card Based Protection for Dalvik Bytecode – Dynamically Loadable Component of an Android APK

By

Muhammad Shoaib

2011-NUST-MS-CCS-047

Supervisor

Dr. Abdul Ghafoor Abbasi

School of Electrical Engineering and Computer Science

National University of Sciences and Technology

Islamabad, Pakistan.

July 2015

Approval

It is certified that the contents and form of the thesis entitled “**Smart Card Based Protection for Dalvik Bytecode – Dynamically Loadable Component of an Android APK**" submitted by **Muhammad Shoaib** have been found satisfactory for the requirement of the degree.

Advisor: **Dr. Abdul Ghafoor Abbasi**

Signature: ____________________

Date: ____________________

Committee Member 1: **Dr. Adnan Khalid Kiani**

Signature: ____________________

Date: ____________________

Committee Member 2: **Mr. Qaiser Chaudhary**

Signature: ____________________

Date: ____________________

Committee Member 3: **Ms. Rahat Masood**

Signature: ____________________

Date: ____________________

Abstract

The global smartphone market is growing at a brisk pace and Android, an open source platform of Google has become one of the most popular mobile operating systems. Android apps generate lot of revenue which is increasing every year. The reverse engineering of Android applications is much easier owing to the use of open source platform. Therefore, it becomes important to protect applications running on Android from attacks like Break Once Run Everywhere (BORE), tampering, illegal use and distribution. The goal is to minimize software flaws by using anti-reverse engineering techniques. In this paper, we present a protection scheme based on obfuscation, code modification and cryptographic protection. The combination of these protection techniques in a single solution can effectively counter reverse engineering on the Android platform. Our approach makes it tough for an attacker to get the business logic performed by an Android application such that the resources and time expended in reverse engineering would be more than are required in building a new application.

Publications

In this thesis, we have made the following contributions:

1. In our paper titled **Software Protection Techniques: A Survey**, authors Muhammad Shoaib and Abdul Ghafoor Abbasi, to be published in Journal of Computer Science and Technology (JCST) we have surveyed, reviewed and categorized current designed and implemented solutions for software protection. An overview of different protection techniques has been discussed by primarily focusing on the use of encryption and obfuscation for software protection and their secure execution. The motivation behind this paper is to study the existing protection techniques, look at their pros and cons and then present a comparative study by evaluating their effectiveness against static and dynamic analysis, tampering and piracy. The study shows that no protection scheme is entirely perfect. Therefore a proposition for future research is presented by suggesting a new software protection technique.

2. In our paper titled, **Smart Card Based Protection for Dalvik Bytecode - Dynamically Loadable Component of an Android APK**, authors Muhammad Shoaib, Abdul Ghafoor and Noor Yasin, published in proceedings of 7th International Conference on Computer Science and Information Technology (ICCSIT) held in Barcelona, Spain on 22nd -24th Dec 2014, we present a software protection scheme for Android applications that utilizes the benefits of encryption, obfuscation and code modification to prevent against attacks like reverse engineering, BORE, illegal use and distribution and tampering. The goal is to give a solution which is capable of restricting the process of reverse engineering and illegal distribution for a reasonable amount of time and hence ensures that desired level of protection of Android applications is achieved.

Certificate of Originality

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at NUST SEECS or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at NUST SEECS or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics which has been acknowledged.

Author Name: **Muhammad Shoaib**

Signature: _________________

Acknowledgment

I thank Allah for giving me perseverance to pursue my master's degree and then my parents and family who have always motivated me to be diligent and hardworking and helped me believe in my abilities to sore high.

Special thanks to my supervisor Dr. Abdul Ghafoor Abbasi who has always inspired me with his dedication, commitment and enthusiasm to work. His guidance, motivation and mentorship by far had been the most encouraging factors to complete my research work.

I would like to appreciate and thank my committee members, Dr. Adnan kiyani, Mr. Muhammad Qaiser Chaudhary and Ms. Rahat Masood who had always given me their precious time and guided me through my work. Their critique has always helped me in improving my work.

I am glad to have a company of friends who has always provided me all kind of encouragement throughout the span of my thesis work. I thank my friends Noor Yasin, Yasir Mehmood and Muhammad Kazim who had been my companions; they have been critically analyzing and helping me with my work. I have been part of the KTH-AIS Lab at SEECS which provided me a competitive, comfortable and friendly environment and necessary facilities for carrying out my research work.

Muhammad Shoaib

Dedication

Dedicated

to

my loving Parents

for supporting me all the way

Table of Contents

List of Figures

I. CHAPTER 1: INTRODUCTION & MOTIVATION

A. *Software protection*

In last few decades, the usage of software applications has increased tremendously. Software applications and their components are playing very important and critical role in socio-economic development and have become a necessity in almost all important fields of life. Ranging from mobile devices to computers, point-of-sale electronic devices to Photostat machines, physical access control to logical access control, telecom sector to e-commerce based web-applications and to software defined systems; Software applications are all around us including online payment systems, enterprise solutions, management information systems, critical health care systems and so on. With such large scale applicability, the software industry generates huge revenue and has become one of the largest contributing factors in a nation's economy and businesses. Worldwide IT spending in the year 2013 exceeded $2.1 trillion as per International Data Corporation (IDC) predictions. This represents a growth rate of 5.7% which is well above the worldwide GDP of 3 percent [1].

The importance of software applications and their unprecedented growth have made software components the prime target of attackers. Software applications are vulnerable to various attacks. The trend of exploiting these attacks for illegal use and distribution of software has increased over the years. Thus software piracy has emerged as a huge threat to the software industry. Business Software Alliance (BSA) reported that software piracy has resulted in a loss of 63.4 billion to software industry. More alarming is the fact that this loss is on the rise and increasing every year as it surged from $58.8 billion in 2010 to $63.4 billion in 2011 [2].

B. *Motivation for Attacks:*

Attacks on software are for variety of reasons. Apart from economic gains attained through software cracking, programmers take it as a challenge and do it for fun for their personal satisfaction. Similarly attacks can also result owing to a disgruntled employee working against the company. There is also a group of hackers who attack software because they are against the concept of Digital Rights Management (DRM) and intellectual property rights and advocate for free and open source software. Another reason behind these illegal activities is that the open source code debuggers and reverse engineering tools are easily available and anyone with basic knowledge and tools can actually end up cracking the software. Thus software cracking has increased probably because of the ease with which an attacker can analyze software binaries and executable, and then modify them for illegal or malicious activities.

We usually see registration keys or licensing restrictions in order to use different software and games. In order to defeat the entire licensing and registration process, most of the attackers replace the single conditional branch statement with an unconditional jump to the desired location. Thus, if an attacker succeeds in breaking the initial protection mechanism then it would be easy to create tampered versions and distribute it illegally to exacerbate the piracy problem.

Protecting and breaking software is a long struggle between vendors and crackers. Users/Clients demand mechanism to ensure that the software they acquire by paying money is authentic i.e. its source is verified. They want that software doesn't contain any malicious content and is not vulnerable. Similarly the software vendors/developers also have few concerns. They want for business reasons only authentic and legitimate users of the software with a valid software license. They also want protection against the attacks like Reverse Engineering, illegal use and tampering and Break Once Run Everywhere (BORE) [3]. In order to address clients' concerns a lot of research has been done in academia and in industry. They have come up with the solutions like Antivirus, Firewalls, Intrusion Detection & Prevention Systems (IDPS) and Digitally Signed Software. These solutions are always evolving and every now and then we get updated versions and updates for such software. This shows that clients' side concerns are usually taken care of. On the contrary the vendor/developer's concerns are not properly catered with and mostly ignored. We need solutions that can protect the software against vulnerabilities exploited by the attackers and reverse engineers.

C. *Android Applications and Piracy:*

In recent years Android platform, which is developed by the "Android Open Source Project" has become one of the most popular systems for mobile devices and the market for Android applications has rapidly grown in variety and financial volume. This platform is designed in a way that developers can upload and publish an app on Android market without a review from Google and users can easily download and install new applications. Since android apps are java based and open source it is quite easy to decompile them and reverse engineer. The Android's smartphone mobile app revenues reached nearly $6.8 billion by the end of 2013, almost doubling its revenues from the previous year [4]. This results in an increasing demand to safeguard intellectual property of developers and protect Android applications files from piracy.

D. *Android App Protection*

Android app protection is relatively a new topic. Android applications can be protected in three ways:

- Anti piracy
- Anti tampering
- Anti reverse engineering.

The ***anti piracy technique*** for application protection includes techniques like licensing, Digital Rights management (DRM) and software watermarking. Google has introduced license verification library, a tool library that protects apps from being stolen by third parties. Operators like Amazon and Verizon have introduced their own DRM options to protect their apps from being pirated or copied. Software watermarking is also an effective tool for anti piracy.

Anti tampering in Android is mainly achieved via a signature mechanism which verifies integrity of the APK file. We can also verify the integrity of classes.dex file in an APK to make sure that source code has not been tampered with. Encryption can also be used to prevent tampering if key exchange and key storage is carefully dealt with.

In order to protect the software from malicious attacks, ***anti reverse engineering*** techniques are used for defense. Since Android apps are java based so protection techniques based on obfuscation that are used to prevent reverse engineering of java bytecode have been researched and applied to dalvik bytecode as well.

Code obfuscation is a technique in which a program is automatically transformed in such a way that its functionality remains the same while it is more difficult to reverse engineer [5] [6]. Obfuscation is a useful and cost effective technique and it doesn't require any special execution environment. Moreover it is believed to be more effective on Android system [7] [8]. Some possible code obfuscation methods on the Android platform are identifier mangling, string obfuscation, dead code insertion [9], clone methods, reordering expressions and loops, changing the arrays and loop transformations [10] that aim at increasing the complexity and control flow of the applications so that it becomes tough for a reverse engineer to get the business logic behind decompiled code of an Android application [11].

Obfuscation has also been used in different code protectors for protection of Android applications. There are a number of code obfuscators like proguard, dexguard, allatori, etc that obfuscate the code by removing unused code and by renaming classes, fields, and methods to semantically obscure names [12]. These obfuscators are used to create apps that are optimized, faster, more compact, and more difficult to crack.

E. *Contributions:*

We have seen that anti reversing techniques that have been implemented for defense of APKs are mostly based on obfuscation. The reason cryptographic protection is not generally used is that the existing execution environments do not support execution of encrypted files. Therefore the encrypted file has to be decrypted before execution and the attacker can intercept the code when it is decoded for execution into the internal memory. Therefore, we have to customize/tweak the existing execution environment to enable it to run the encrypted software. In this paper we have proposed a solution based on obfuscation, code modification and cryptographic techniques for software protection. The cryptographic protection prevents static analysis while obfuscation and code modification delays dynamic analysis. A combination of both can effectively make reverse engineering hard even for an experienced attacker.

F. *Organization of Thesis:*

This thesis is organized as follows:

Chapter 2 describes the literature survey and related work regarding software protection in general and android application protection specifically. It gives an overview of android architecture various tools and techniques that can be used for protection and reverse engineering.

Chapter 3 describes the research methodology followed during the thesis to achieve the desired goal. A hybrid of research methodologies was used to achieve the required results including deductive research and conceptual research methods.

Chapter 4 gives an overview of the proposed solution and then components of the design are explained with an approach to get maximum inherent security and ultimate delay of the reverse engineering of APKs. The shortcomings, restrictions and special requirements if any for every module are discussed as well.

Chapter 5 gives implementation details, evaluation and analysis of the proposed solution.

Chapter 6 finally gives conclusion on the android protection method and suggests future research directions.

II. CHAPTER 2: LITERATURE SURVEY

In this chapter we will look into the android structure and its components, the android APK build process and will show how an android APK can be reversed engineered using a few tools. We will look into various techniques and tools that have been used to prevent reverse engineering of Android apps. This chapter will give the readers an understanding of the Android APK and protection techniques which will be helpful in understanding concepts presented later in this dissertation.

A. *Attacks on Software*

Owing to the widespread acceptability of various software applications, different attacks on software have emerged. We will define these attacks below:

1) *Reverse Engineering*

The process of taking a software program's binary code, analyzing it and recreating it so as to trace it back to the original source code is called reverse engineering. It is just like disassembling parts of a machine or vehicle, understand its functions and then make appropriate changes and adjustments to get a better performing module. In context of software engineering, reverse engineering can be used both for productive and non-productive purposes. Analyzing software to extract design and implementation information can be useful in a scenario where original source code is lost and it becomes imperative to reverse engineer the module, get the business logic and improve it for further use. Similarly a reverse engineer can have malicious intentions as well in order to steal the business logic of the software.

2) *Break Once Run Everywhere (BORE)*

It is a class of attack that if successfully executed on one instance of software can be applied to crack all other instances of the same software. This is owing to the fact that all copies of software lack diversity and have the same binary code resulting in development of a generic reverse engineering attack. If successful, BORE can result in tremendous financial loss to a software developer/company.

3) *Tampering, illegal use & distribution*

Tampering is modifying software binary to attain various objectives ranging from adding malware or malicious content with the code to bypassing the mechanism of license key to crack to software. Once cracked, software can be illegally used, distributed and even modified to make a copy utilizing the same business logic and hence violating the Intellectual Property Rights of the developer.

B. *Protection Against Attacks*

Security organizations as well as researchers around the globe are working in three directions to achieve protection against the attacks on software. These include solutions based on:

1) *Cryptographic and obfuscation techniques*

The first approach for software protection is by using the techniques of obfuscation and cryptography. The details of these techniques are given below:

a) Obfuscation

Code obfuscation is a technique in which a program is transformed in such a way that its functionality remains the same, while it is more difficult to reverse engineer using static and dynamic analysis. The objective of obfuscation is to change the physical appearance of code, to make it unintelligible or hard to understand. A program can be obfuscated according to the security needs i.e. for more security a number of obfuscation transformations can be applied, but the performance of program will also be compromised with every transformation. Thus there is always a tradeoff between achieving desired security and the program efficiency and performance. Obfuscation is a useful technique and provides lot of benefits. It gives protection against static and dynamic analysis, such that it becomes very difficult and time consuming for the attacker to reverse engineer the obfuscated program. Obfuscation introduces diversity, by creating multiple instances of the program, which can be helpful to counter global attacks. Since obfuscation is a software based protection method, it incurs no extra costs of maintenance or update, and provides a cost effective solution. As obfuscation transformations can be applied on high level code providing greater flexibility and platform independence. Obfuscation techniques can be further classified into the following types:

- Layout Obfuscation
- Control Flow Obfuscation
- Data Obfuscation
- Preventive Obfuscation

b) Cryptographic Protection

Encryption can be used to protect software as it thwarts the attackers to gain access to the source code and thus prevent reverse engineering. The main challenge in this technique is the inherent problem of existing implemented execution environments that do not support execution of cryptographically protected software. Therefore, the software has to be decrypted on the fly with a decryption key prior to its execution.

2) Digital Rights Management (DRM)

Digital Restrictions Management (DRM) is any technology that is built into a hardware or software product with the aim of limiting its illegal use after purchase. It is designed to prevent customers from using digital technology in ways that are conflicting to the business plan of product or service provider. In essence, DRM removes usage control from the person in possession of digital content and puts it in the hands of a computer program. E-books with copyrights, games and software requiring license keys, CDs and DVDs restricting number of copies or preventing against ripping software are just a few examples of digital rights management:

3) Software Security

Software security, as the name suggests is building secure software by design. It is based on the principle that building secure software is better than protecting a vulnerable one. Software security aims at applying good software engineering practices during the design and implementation phase. It includes knowing, understanding and analyzing the common threats, designing secure software and then testing against the attacks [13]. Software security is relatively a new technique but applying best practices of security at design and architectural level can make the software inherently secure.

C. *Android Architecture*

When an Android program is compiled, all of its parts are packaged into one file called Android Application Package file (APK). An APK is a zipped file formatted package with .apk extension and contains:

- Dalvik Executable file (.dex file containing dalvik bytecode)
- Resources & Assets
- Certificates
- Manifest file

1) Dalvik Executable File (DEX)

The dex file contains dalvik bytecode i.e. the program code for an Android application and is executed by the Dalvik Virtual Machine (DVM). Apart from executing dalvik bytecode, DVM also provides the ability to execute code which is not part of the dalvik bytecode i.e. by calling native functions within shared objects. The dalvik bytecode is comparable to java bytecode but is designed specifically for Android applications. Owing to limitations of resources on mobile devices, dalvik bytecode is more efficient and compact than java bytecode.

2) *Resources and assets*

Resources and assets of an APK file include bitmap files, sounds and other static data that is used by Android application.

3) *Manifest File*

Every application has an AndroidManifest.xml file in its root directory which presents essential information about the application to the Android system.

4) *Certificates*

An APK file, in order to be installed on any device, has to be digitally signed with a certificate whose private key is held by the developer. Self Signed certificates can also be used without the need of a Certificate Authority. The signing process adds no security and merely identifies the application developer. It happens automatically when we use Eclipse with the Android Development Toolkit (ADT) plugin.

The Android system comes with an optimizer and verifier tool called “dexopt” [14]. When an Android application is installed on any device, dexopt will optimize and verify the dalvik bytecode for efficient execution on underlying architecture. This process is called optimization and the resultant dex file is called “odex”.

D. *Sandboxing*

When it comes to Android security, a key concept is “secure sandbox”. The Android application Sandbox isolates app data and code execution from other apps. This is achieved by assigning a unique user ID (UID) to each app running as a process. By default no application has the permission to interfere with another app’s resources and private data. Only processes with same UIDs can share resources. In order to allow an application to interfere with another application's sandbox, permissions must be explicitly declared up front before the app is installed and cannot be changed after installation.

The only way to break out of application sandbox is by compromising the Linux kernel which is the operating system Android is based upon. This is called rooting the device where each app will have a root level access and can modify other apps data as well as the kernel. Rooting an Android device renders all the security mechanism null and void.

E. *Android Application Build Process:*

An Android application needs to go through many steps and requires different tools for building an APK file which is ready for deployment. The first step of the build process, as shown in Fig. 1 is compilation of ".java" files into ".class" files using Java compiler. Class files contain java bytecode and at this step obfuscation can be applied to this java bytecode. In the next step java bytecode is transformed into dalvik bytecode using a utility "dx" which comes with Android Software Development Kit (SDK). The application of "dx" utility on class files results into a single dex file called "classes.dex". It is possible to apply further obfuscation on dalvik bytecode at this stage using Proguard or any other obfuscation tool. The ApkBuilder then constructs an APK file from the "classes.dex" file and adds further resources like images and ".so" files. The ".so" files are shared objects which contain native functions that can be called from within the DVM. In the last step "jarsigner" adds developer's signature to the APK. The signed application can finally be deployed on an Android device.

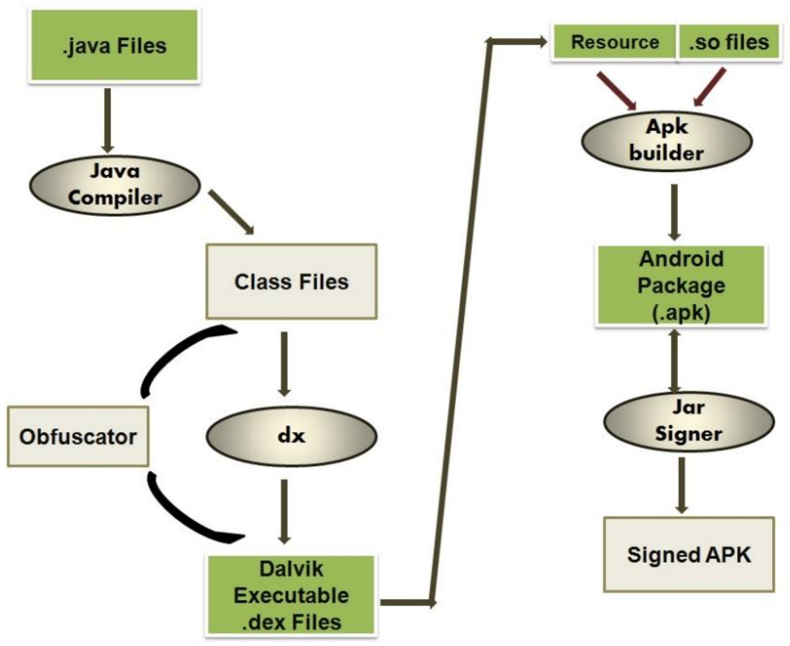


Fig. 1. Android application build process.

F. *Reverse Engineering an Android APK:*

Since android is based on an open source platform (JAVA) and we already stated that it is easy to decompile an APK using some tools and basic knowledge.

1) *Analysis Tools*

There are many tools to interact and analyze Android applications. Smali/baksmali [15] assembler and disassembler are useful tools for reverse engineering of Android apps. JAD [16] is a tool that is used to extract source code from class files. Androguard [17] is python based tool which is used for reverse engineering of APK files. IDAPro [18], DexDump [19] and Dexter [20] are also useful code analysis tools which are quite effective for reverse engineering of Android apps.

We give an example of decompiling an APK using the following tools:

- APK Extractor
- Dex2Jar
- JD-GUI

In the first step we use APK Extractor [21] to extract the contents of an APK.

As a result of extraction we can get access to the App resources, DEX file and the Android Manifest file. As we know the DEX file contains all the code in the form of dalvik bytecode, so we need to decompile into java .class files. Dex2Jar [22] is used to convert the resulting files to JAR file which include the class files. We then use JD-GUI [23] to decompile the class files to readable java code. The process of reverse engineering is shown below:

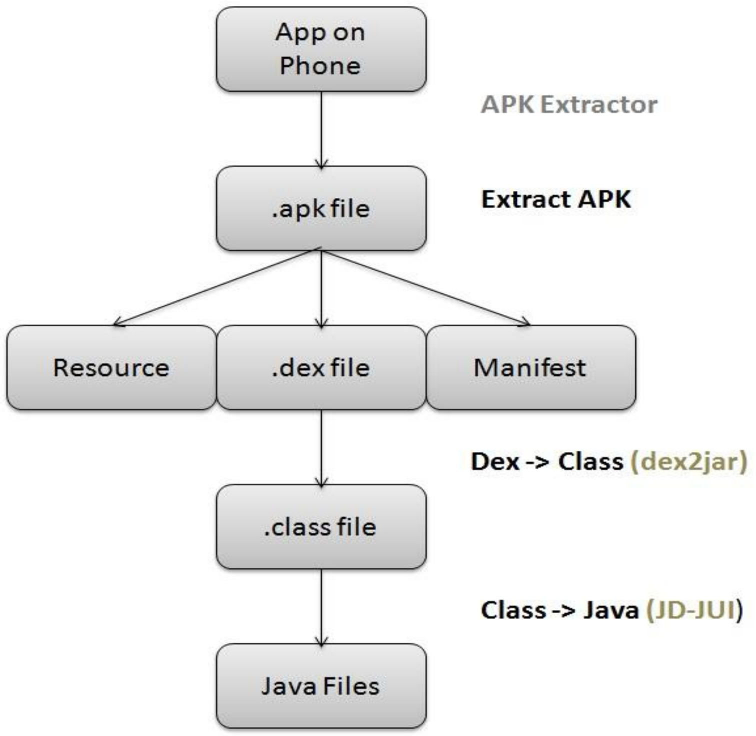


Fig. 2. Decompiling an APK.

III. CHAPTER 3: RESEARCH METHODOLOGY

This chapter presents the research methodology that we have adopted during our research work to achieve our goals. Our research was carried out in two different parts. In the first part a detailed analysis of security situation of android application files was done, different attack vectors were studied and also methods to overcome/counter these attacks were analyzed. In the second segment of our research we focused on design and implementation of a protection scheme that can effectively counter the various Reverse Engineering attacks against android application files (APK files). For completion of our objectives we have adopted different research methodologies during our research. Different research methods are combined to conduct the research via a hybrid approach. This chapter illustrates integration of various research methods into a single one and the effectiveness of using a hybrid approach for achieving results. In this chapter we define a complete methodology to identify problem statement, develop a hypothesis and form a prototype based on the hypothesis.

A. *Intro (Research definition)*

The word research is derived from the "recherche", which means "to go about seeking". Research has been defined in a number of different ways:

A broad definition of research is given by Martyn Shuttleworth - "In the broadest sense of the word, the definition of research includes any gathering of data, information and facts for the advancement of knowledge" [24].

Another definition of research is given by Creswell who states that - "Research is a process of steps used to collect and analyze information to increase our understanding of a topic or issue". It consists of three steps: Pose a question, collect data to answer the question, and present an answer to the question [25].

Research is a work done on systematic basis to increase knowledge and then use this knowledge for developing new theories and applications. It involves investigating a hypothesis, suggesting new interpretations of data and posing questions for future to explore. Thus research is not an individual activity rather it’s an act of community. Researchers around the globe build on the knowledge others have acquired over the years and provide a roadmap for future research. Every research is done with an objective in mind and achieving that objective is a motivating factor for research work. There are different research methods that can lead to achievement of our goals but we follow the solution and research method which is most effective and efficient.

B. *Research types*

The various research types are defined below:

Descriptive Research is also called survey research and describes the state of the art research related to a specific subject area. It employs the methods of survey and questionnaires to collect data and record observations but cannot be used for novel contributions.

Analytical Research is a type of research which is done by analyzing already collected data, evaluating it and drawing conclusions from it. Analytical research involves analyzing a particular solution and suggesting ways of improving it.

Applied Research deals with the study of problems faced by industry, business organizations and society and developing a solution to these practical problems. This type of research is required for real world problems and aims at developing new technologies and solutions.

Fundamental Research is also called pure research. It is a systematic study to gain knowledge and understand core concepts related to a phenomenon. It involves observation of facts to develop a theory and add to the broad base of organized knowledge that works as a foundation for applied research.

Quantitative Research, as the name suggests relates to quantifying data for mathematical analysis. It aims at testing hypothesis and making case and effect relationships. In this type of research, the researcher gathers data through surveys to make statistical relationships and test his hypothesis. The focus of this research is to narrow down a problem and draw conclusions in connection with mathematical expressions.

Qualitative Research aims at exploring issues and understanding phenomenon. It uses unstructured data for analysis and answering questions.

Qualitative research methods originated in social and human sciences and involve the use of various empirical materials like case study, interviews, group discussions, attitude or opinion tests, program evaluation, etc.

Conceptual Research aims at developing new ideas and concepts on the basis of existing theories. It focuses only on theory, investigates ideas and reinterprets them. It involves current review on literature and critically analyzes a research to give one's own proposition. Thus conceptual research formulates new concepts in order to improve existing ideas.

Empirical Research as the name suggests is completely related to practical work. It aims at verifying facts on basis of observation and experiment and then draws conclusions. In order to carry out this research, initially a working hypothesis is formulated which is then evaluated and proved on the basis of extensive experimentation. Empirical research gives the most powerful support to a given hypothesis.

C. *Research Methods vs. Research Methodology*

The tasks a researcher carries out during course of his research study are known as research methods. It includes all the procedures and methods used to initiate and then complete the research. On the contrary research methodology is a broader term that encompasses research methods too. It gives basic guidelines of how to carry a research and defines principles for carrying a systematic research. Researchers in order to carry out an effective research need to understand what research methods are most suitable for a certain scenario. They need to understand the logic behind using a specific research method and its applicability in the real world. A researcher can formulate his own research methodology on the basis of problem being studied and research methods being used. Thus research methodology provides a much wider scope and covers not only research methods and techniques but also the logic behind using a specific research method for a particular problem.

D. *Thesis Research Methodology*

Thesis research methodology is based on the research objectives that are targeted. Our research was conducted in two major phases. One was to analyze the in depth security of android application files (APK) and identify the vulnerabilities and possible attacks against these applications. The existing methods and techniques for protection of APKs were also studied. The second objective was to design a protocol for protection of android APKs against attacks like reverse engineering and tampering. We proposed a protection technique build on existing protection methods and implemented a prototype solution. To achieve our objectives and goals we had to adopt a mixed research methodology approach and use more than one research methodologies as described above. For instance we used conceptual research method to study the security architecture of android APK files and adopted empirical research to implement a solution for security of APKs.

E. *Research Objectives*

The objectives with which a research is carried out are called research objectives. We had two major objectives:

- Analyze the security of Android application files.
- Design, implement and validate a framework for protection of android APK files.

F. *Research Approach*

To achieve our set of objectives we followed a hybrid of research approaches. We have used deductive and conceptual research approaches to study the literature and review it critically. We started by studying the theories related to android architecture. The protection methods that have been used against attacks like reverse engineering, BORE and tampering were studied with main focus on improving the protection against these attacks vectors. Protection techniques like obfuscation and encryption were examined in detailed by studying various survey papers, conference papers and journals. All this research and study led us to the development of our research hypothesis. A hypothesis is a proposition or supposition based on reasoning and limited evidence and it serves as a starting point for our investigation and future research goals. Hypothesis forms the basis of a phenomenon that is to be tested and verified during the course of our research.

G. *Hypothesis*

The hypothesis of our research was:

Do the security solutions based on obfuscation or encryption prevent the android APKs from reverse engineering?

Android is based on java which is an open source platform so can we have a solution which provides run-time security against static and dynamic attacks.

Can we use a combination of solutions to provide a better and more reliable security model for android APKs protection?

H. *Observations*

In order to support the hypotheses that have been formulated, the following observations have been made regarding android bytecode protection.

- Android is based on an open source platform and the protection techniques like obfuscation which are usually used to protect java bytecode are also used for dalvik bytecode protection.
- Cryptographic protection and encryption is usually not used for protection of binaries as the existing platforms and environments do not support runtime execution of encrypted files.
- A combination of obfuscation and encryption with modification of execution environment can be used to provide a better security against reverse engineering attacks.
- Moreover runtime bytecode modification can further make the application secure making it very tough and time consuming for the attacker to reverse engineer and APK.

I. *Implementation and Verification of Prototype:*

The last step for deductive research approach was to develop and verify the prototype. Our design focused on protection of reverse engineering of android APKs by using a combination of protections techniques like obfuscation, encryption and bytecode modification. Our prototype includes obfuscation and encryption. We are not using smartcard for our simulation rather the password for decryption is entered at runtime from a dialog box. The APK consists of manisfest file, certificates, images and classes.dex file. The main code and business logic of an android app lies in classes.dex file and we are only protecting this file while rest of the data remains unencrypted. The overall research methodology adapted during the thesis is shown below:

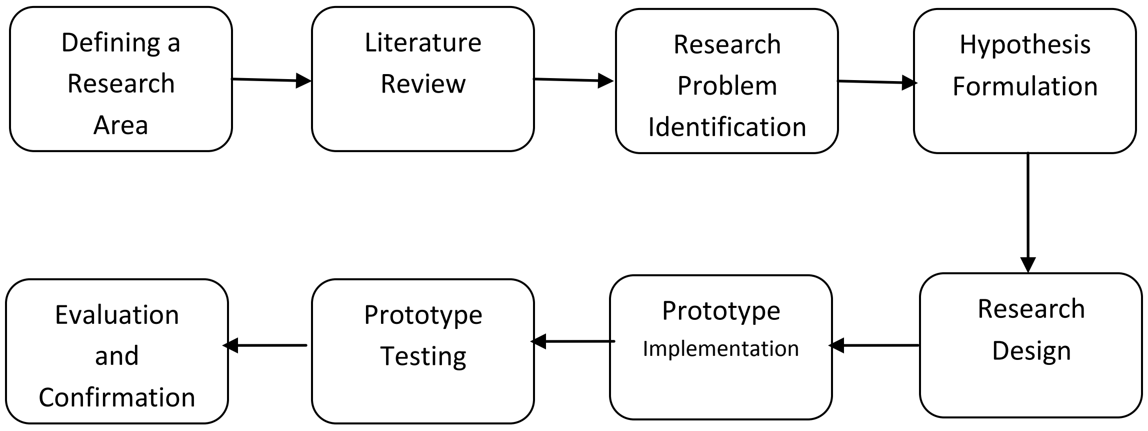


Fig. 3. Thesis Research Methodology steps.

IV. CHAPTER 4: DESIGN

In this chapter we have first given an overview and then detailed description of the proposed solution. Various components of the design are explained with an approach to get maximum inherent security and ultimate delay of the reverse engineering of APKs. The shortcomings, restrictions and special requirements if any for every module are discussed as well.

A. *Design Overview*

In the first step of our design, the application developer publishes the app on the application repository. On receiving a download request from the user, app repository redirects it to Identity Management Server (IDMS) which requires the user to register on the IDMS. The user then has to access the app repository to download the app containing encrypted dex file. The app can only be downloaded upon authentication from the IDMS. Once the file is downloaded it can be executed in a customized environment using a Dexfile loader. The encrypted file is decrypted on the fly using a key provided by the user via smartcard. An overview of design is given below in Fig. 4:

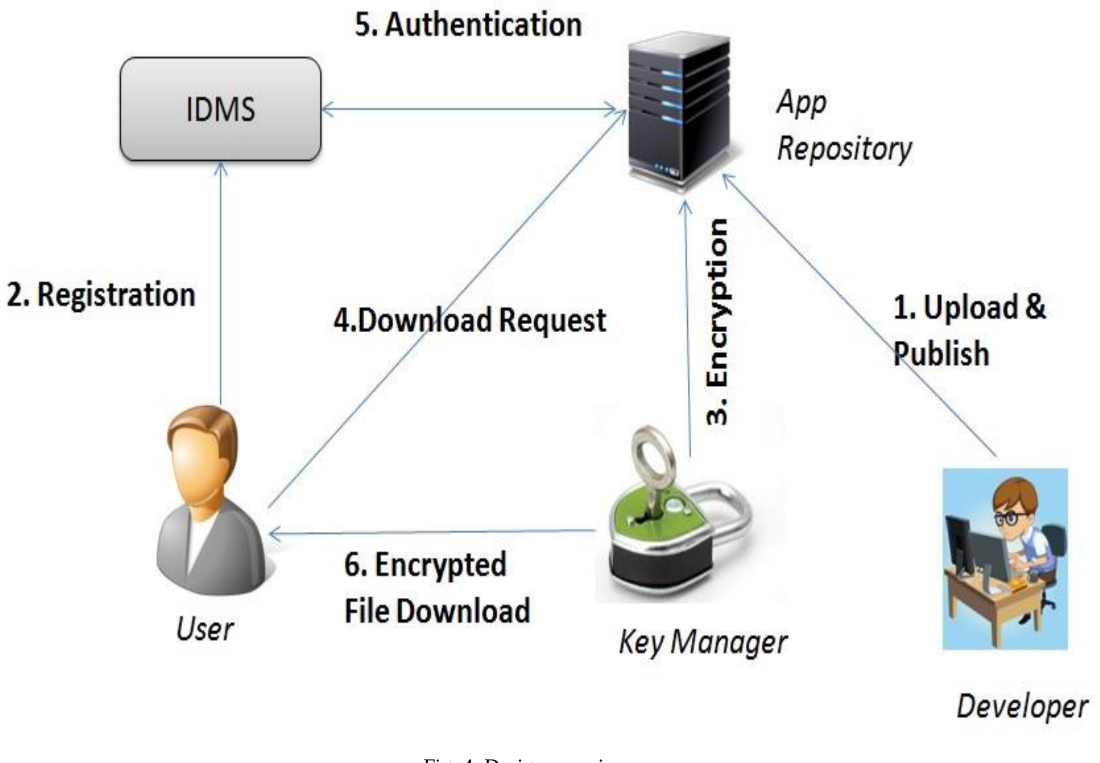


Fig. 4. Design overview.

B. *Registration*

The user first registers with the registration server (IDMS) using registration web page [26]. This registration information is sent using SSL protocol to the

Web Server. Once the registration process is complete, a group ID is generated for the user which acts as a passphrase for generating a Groupkey for dex file encryption/decryption later. This ID is same for a group of users with size n and is changed for every subsequent group. For example if we take a group size of 10 (i.e. n=10), then GroupID will remain same for 10 users and will change for next group of users. Thus generated key is same for a group that consists of n users.

In order to download the app, the user sends a download request to app repository. The app server verifies that user is registered at the IDMS and allows the download to proceed after authentication as shown in Fig. 5.

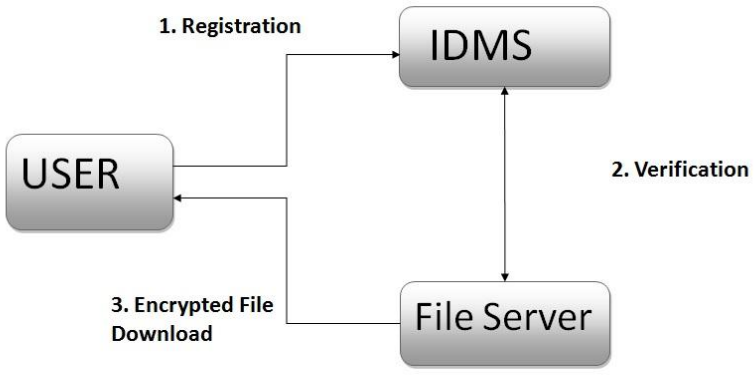


Fig. 5. Registration, Authentication & file download

C. *Encryption & Decryption*

Our protection technique has two main parts:

- Packer (Encryptor)
- Unpacker (Decryptor Stub)

1) Packer

A perfect obfuscator would result in transformation of an application in such a way that it is impossible to analyze it and extract any information from it. But obviously it is not possible to generate such an ideal transformation. A close enough result can be achieved by using a technique often used by malware i.e. packing [27]. A packer takes an executable or binary file, encrypts it so it cannot be analyzed by the attacker/analyst unless decrypted. We know that the main code of an Android application is stored in a dex file and we can use encryption to protect the dex file. In Android, encrypted dex file can easily be generated using AES [28]. The GroupID which was generated at registration step is used as a passphrase for generation of key using Password Based Encryption (PBE) [29]. This key is used to get a .dex file with an encrypted dalvik bytecode. The application containing encrypted dex file is then stored on the server. The PBE generated Groupkey is stored by key manager using public keys of the registered users. Thus each user can use his private key to extract the GroupKey which will be used for decryption later. In this way the symmetric key can be distributed to multiple recipients of the group who have registered at IDMS and downloaded the application with encrypted dex file.

2) *Unpacker*

The unpacker or Decryptor stub is an important component of our design and it must be executed when starting an application. It performs some key functions like fetching the dex file, decrypting and loading it into the memory and executing it. The loading of dex file is generally achieved by using reflection which loads dex file from a certain location in a currently running process. The problem with this simple approach is that data has to be decrypted on the system prior to execution and this decrypted data can easily be copied by the analyst, rendering the whole protection scheme futile. The unpacker in our solution can be used to load encrypted dex file and decrypt it on the fly for execution.

Our main aim is to make hard the analysis of unpacker and protect dex file containing dalvik bytecode. This can be achieved by obfuscating the bytecode and encrypting dex file using a cryptographic function like AES. Android does not allow loading an encrypted dex file. In order to overcome this problem, we first decrypt the encrypted dex file into a bytearray and then load from it. Loading a dex file from bytearray into the Dalvik Virtual Machine (DVM) is also not possible in Android by using standard procedures. Therefore we access the private method "private static int openDexFile(byte[] fileContents)" of DexFile to achieve this task [30]. Private methods in Java can be accessed either by calling setAccessible (true) on method before invoking it or using Java Native Interface (JNI). In this way we do not have to store the decrypted dex file on local storage rather it is present only in the volatile memory and for rest of the time it remains encrypted. The symmetric key is also stored and entered via a smartcard by the user and is not stored in the application which makes protection scheme quite secure. The file encryption and decryption process is shown in the Fig. 6 below:

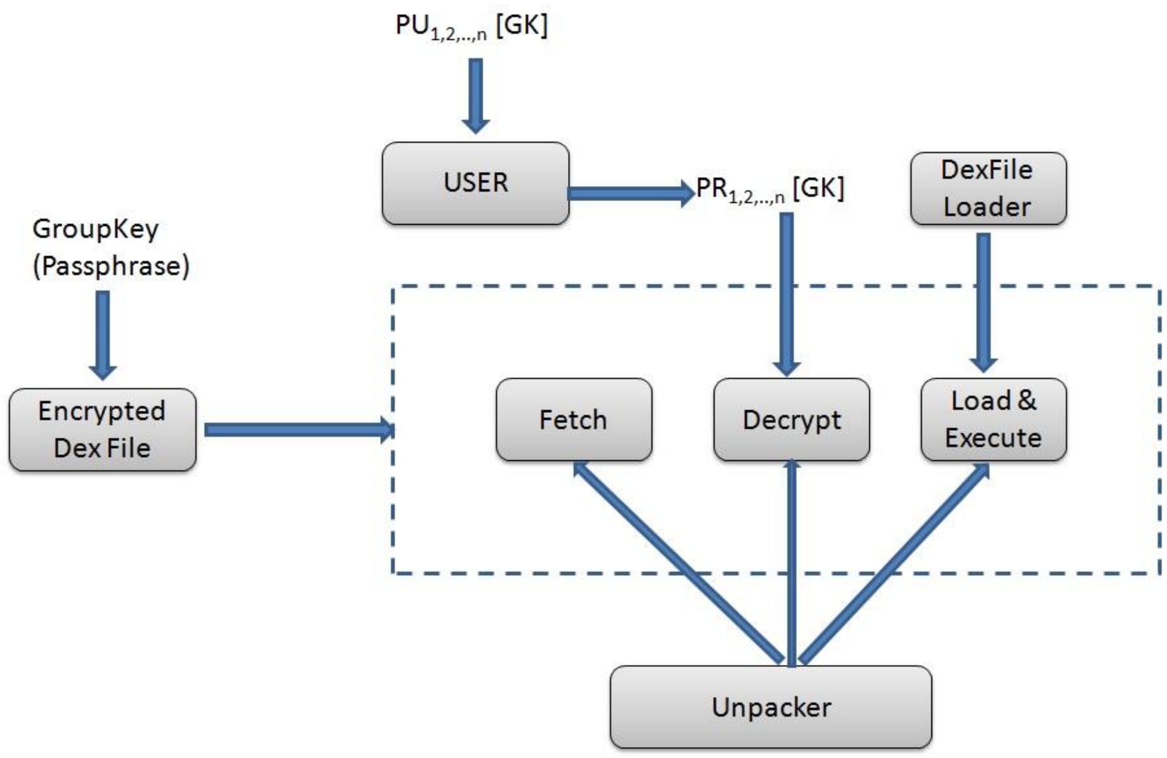


Fig. 6. File encryption & decryption process

So far the only place where the bytecode is present unencrypted is in the process memory. An expert attacker can root the device and trace process memory to get useful information. We know that the DVM as well as the dalvik bytecode present in the process memory cannot be modified at run time but we can circumvent that restriction by using native code that can be called from within the DVM using JNI to modify the dalvik bytecode. Thus, if the attacker roots the device and accesses process memory still it won’t be easy to analyze the intercepted code owing to obfuscation and code modification.

V. CHAPTER 5: IMPLEMENTATION & ANALYSIS

In this chapter we give an overview and detailed description of how the proposed solution is implemented. It includes how the protection techniques are incorporated in the implementation of proposed design described earlier. Techniques like obfuscation and encryption are utilized for protection of dalvik bytecode while a code modification technique at runtime is also used to provide added security. The security of resultant dalvik code is also analyzed for testing and improvement.

A. *Android Security Model*

The android security model is designed based on layers that provide flexibility as well as protection for users and developers resulting in secure applications.

Users are given information on how the applications work and what permissions they have on their device.

B. *Security Architecture*

The Android operating system's goal is to protect user data, system resources and provide applications' isolation. The following security features are provided are provided to achieve these goals:

- Robust security through the Linux kernel at OS level
- Application sandboxing
- Secure interprocess communication (IPC)
- Application signing
- Application and user-defined permissions

The core of the Android security model is the Linux kernel which has been around for a very long time. Linux is a very robust kernel and continuously being improved. It is widely used in industry and trusted by many professionals. This kernel provides the Android OS with a user-based permissions model, process isolation, mechanism for secure IPC, and the ability to remove parts of the kernel.

The figure below shows the different components and considerations of the Android software stack. Each part of the stack operates under the assumption that everything below it is properly secured [31].

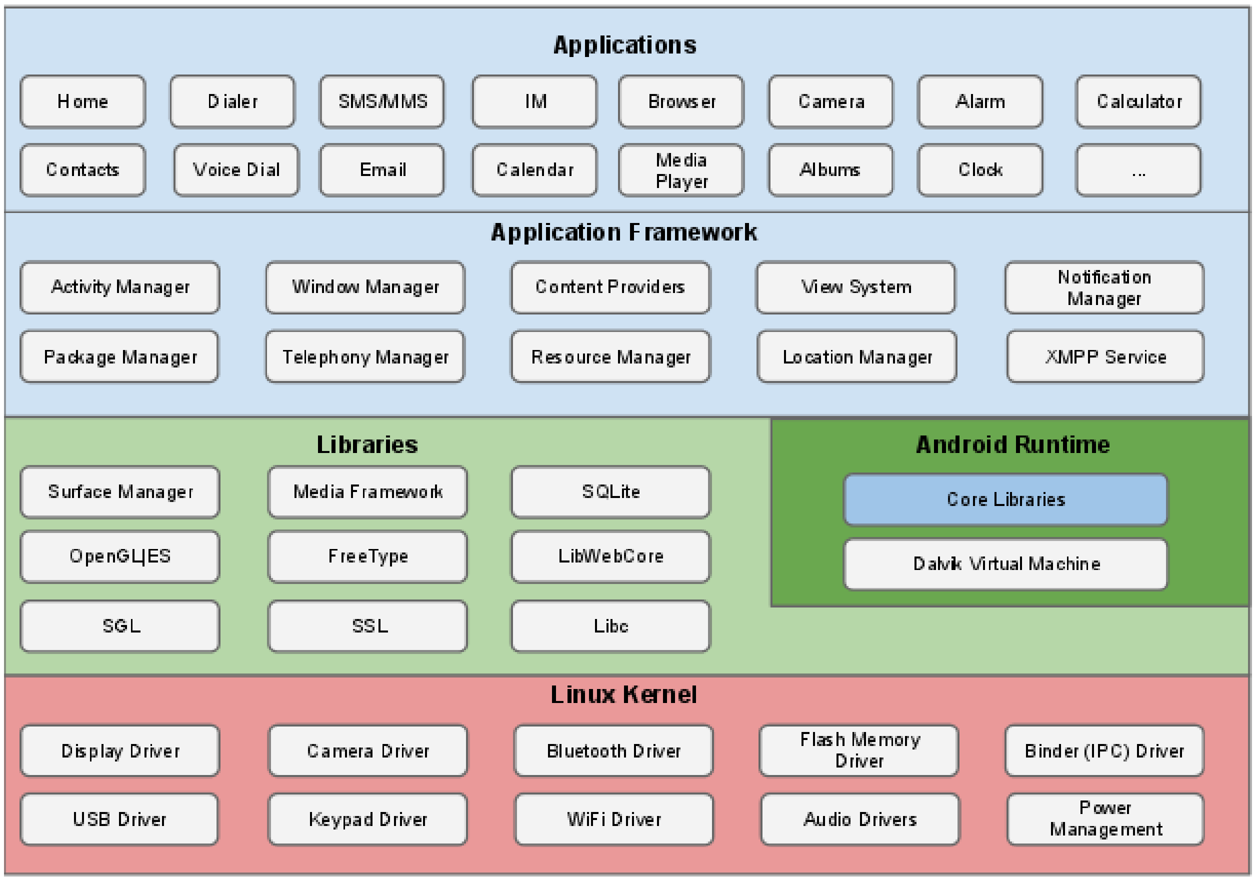


Fig. 7. Android Security Architecture

C. *Attack Vectors*

Android application attack surfaces can be categorized as:

- The application code/business logic
- Application permissions
- Data at rest and in transit
- Client side logic
- Server side logic

Since, in this research we are dealing with standalone applications so our focus will be on the applications code/business logic. The developers are usually under the impression that if they obfuscate their code it cannot be decompiled and thus they implement the business logic into their applications. A determined and capable attacker can decompile, de obfuscate and analyze an application to successfully reverse engineer it. Therefore a better approach is to keep the business logic at the server side. But there are situations when we have to develop standalone applications with business logic incorporated in them. In our research we have given a design that can delay the reverse engineering of an application's business logic by utilizing a hybrid of protection techniques.

D. *Implementation overview*

In our implementation we have used a combination of techniques in order to implement a secure solution. In order to implement a proof of concept application we have used an emulator for android SDK. Although the real application requires smartcard and NFC enabled smart phones, we are using a dialog box authentication requiring a user input.

E. *Obfuscation*

Obfuscation is a technique in which program code is transformed in such a way that it is hard to understand and reverse engineer but the program remains functionally equivalent. Obfuscation in our code is implemented by using a built in utility called ProGuard [32] which shrinks the code and obfuscates it by replacing names by meaningless character sequences. It can also perform optimizations at the bytecode level, inside and across methods using techniques like control flow analysis, data flow analysis, partial evaluation etc. we can also use obfuscator like DexGuard [33] which is slightly more efficient than ProGuard. Similarly there are other obfuscation techniques like identifier mangling, string obfuscation, dead code insertion, clone methods, reordering expressions and loops that can be used for code obfuscation purposes.

F. *Encryption & Decryption*

Android application's code is contained in a file called classes.dex. Therefore we have used AES to encrypt this file. Android provides the library to use AES encryption. The component which uses the encryption is called packer. An important component in our solution is unpacker which loads encrypted dex file and decrypts it on the fly for execution.

Loading an encrypted dex file is not allowed in android. The problem is tackled by decrypting first the encrypted dex file into a bytearray and then loading from it. Loading a dex file from bytearray into the Dalvik Virtual Machine (DVM) requires to access the private method **“private static int openDexFile(byte[] fileContents)“** of DexFile to achieve this task. **SetAccessible (true)** is called on the private method before invoking it. In this way the decrypted dex file is present only in the volatile memory and for rest of the time it remains encrypted. The symmetric key is also not stored in application and is entered by the user.

G. *Dynamic code modification*

In order to modify Dalvik bytecode during runtime, we have to use native code. This is done using JNI, which enables us to execute native code within current process context. This means we can access the process memory where dalvik bytecode is running. The advantage of using native code is that we can access arbitrary memory locations and modify the dalvik bytecode at runtime.

H. *Analysis*

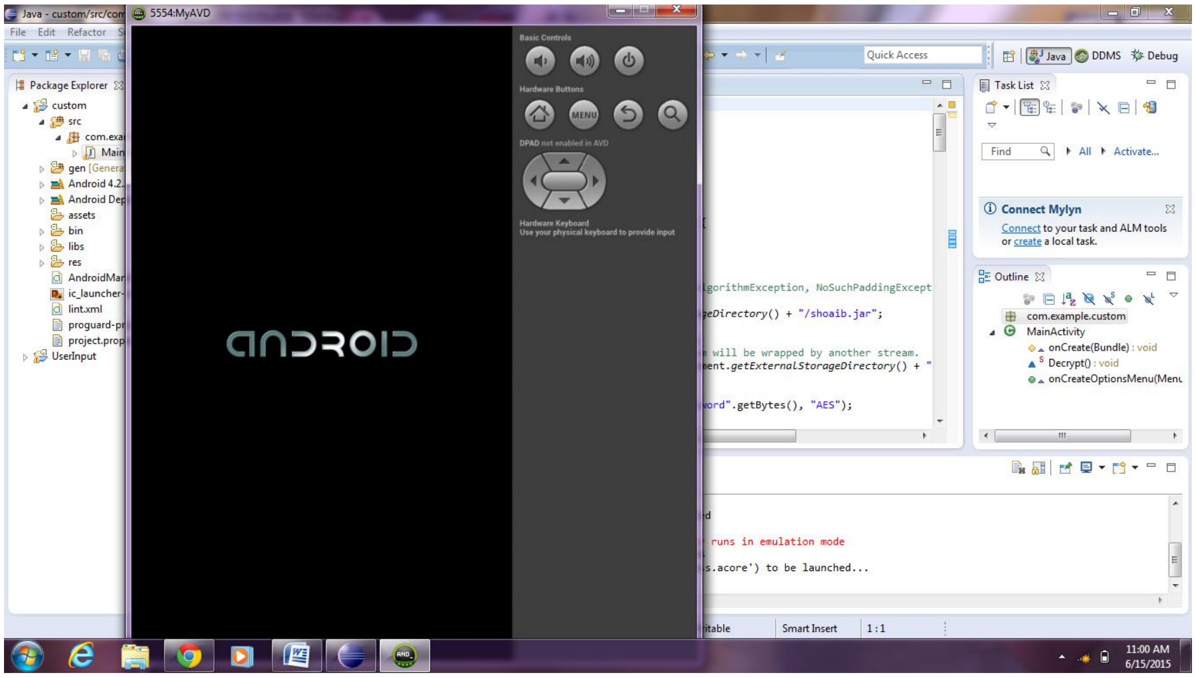
We have used obfuscation and encryption that provide protection against static analysis. The dynamic analysis is countered by runtime modification of the dalvik bytecode. Since the dalvik bytecode is altered at runtime an attacker must analyze the native code as well for reverse engineering purposes which makes the task even harder.

I. *Implementation Screenshots*

Some important implementation steps and their screenshots are shown below.

1) *Android Emulator:*

Instead of using a real smartphone an emulated android environment is used for testing and deploying the APK.



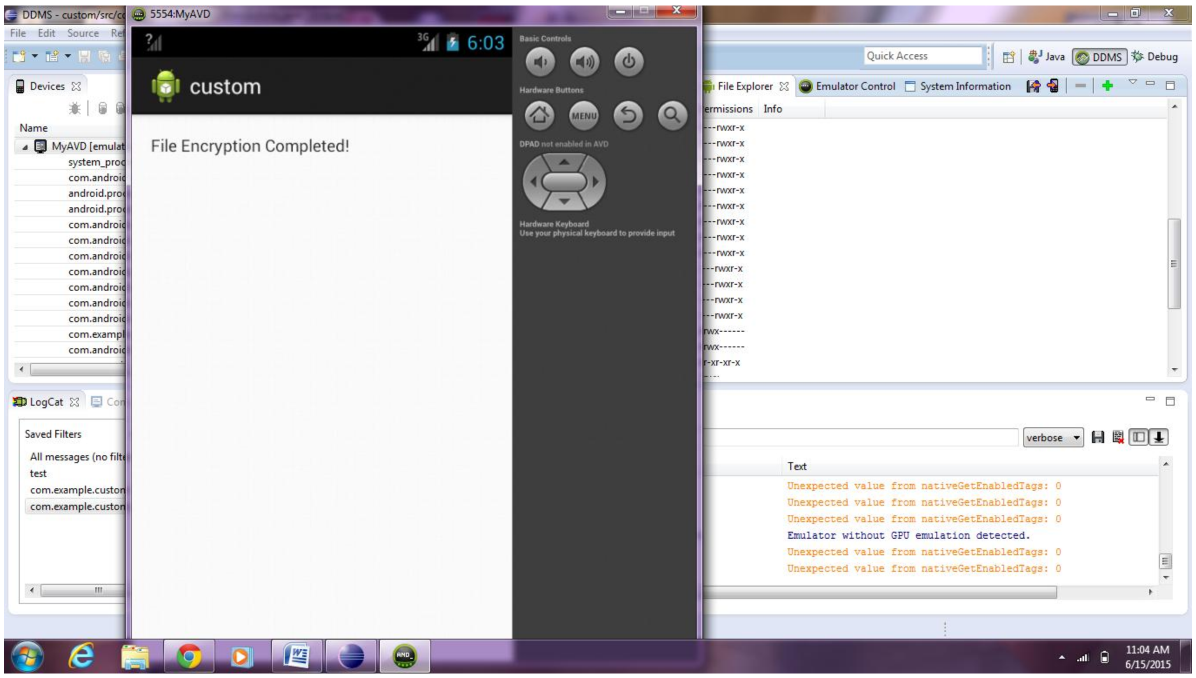
```
lgorithmException, NoSuchPaddingExcept
geDirectory() + "/shoaib.jar";

 will be wrapped by another stream.
ment.getExternalStorageDirectory() + "

word".getBytes(), "AES");
```

```
d
 runs in emulation mode
s.acore') to be launched...
```

Fig. 8: Android Emulator

2) *File Encryption Process:*

The dex file is encrypted using AES and 16 Byte key for this proof of concept application. Once the file is encrypted we cannot deploy it, extract it or display its contents without the decryption key.



File Encryption Completed!

```
Unexpected value from nativeGetEnabledTags: 0
Unexpected value from nativeGetEnabledTags: 0
Unexpected value from nativeGetEnabledTags: 0
Emulator without GPU emulation detected.
Unexpected value from nativeGetEnabledTags: 0
Unexpected value from nativeGetEnabledTags: 0
```

Fig 9: File Encryption

3) Entering Decryption Key

The decryption key is entered at runtime through a dialogue box in order to get the decrypted dex file. The entering of key via dialog box is just for this proof of concept application. In a real environment we have to use NFC enabled smartphone and use smart card to store the key and reenter it for decryption.

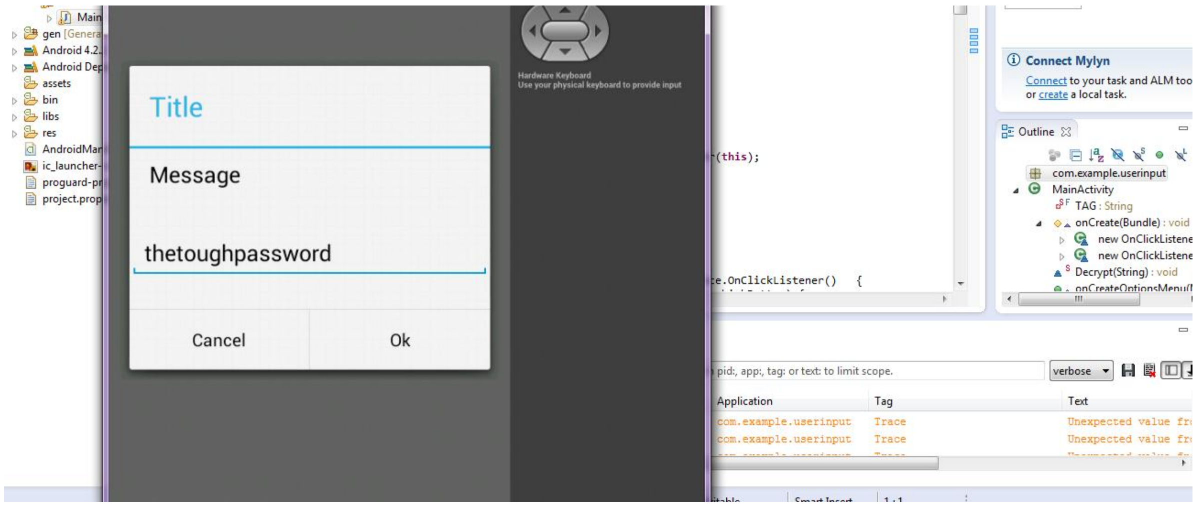


Fig 10: Entering password for file decryption.

4) *File Decryption:*

Finally the file is successfully decrypted after entering the correct decryption key.

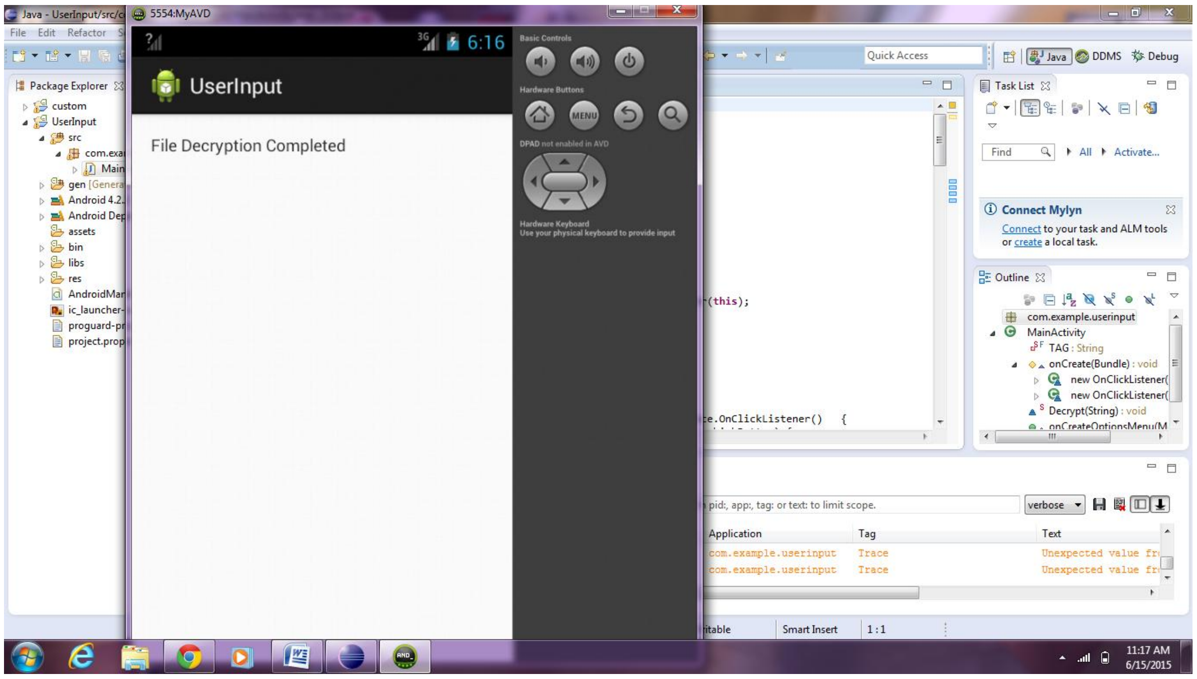


Fig 11: File Decryption

VI. CHAPTER 6: CONCLUSION & FUTURE DIRECTION

Software security is an ongoing research and android application security is relatively a new topic. Over the years developers have been trying to protect their codes and binaries from attackers. Attacks like Reverse Engineering, tampering and BORE have evolved and techniques like obfuscation and encryption have been used to counter these attacks.

There is no such thing as perfect security and usually the aim of protection technique is to sufficiently delay the reverse engineering time so that attacker has to utilize more time and resources. As a matter of fact every protection scheme can be bypassed if it is assumed that the attacker has infinite resources. Therefore, if the resources required to reverse engineer an application are more than the value of application, the protection scheme should be deemed successful.

A. *Advantage and contribution of research:*

The protection technique proposed in this research is for the applications where we have to deliver the app code and resources to the users in order to run the application in offline mode. By applying the approach proposed here we are making it difficult for the attackers to reverse engineer and access the bytecode. Once we modify the execution environment, the dex file is decrypted on the fly into a bytearray from which it is loaded and then executed. The advantage of this approach is that code is not decrypted into the internal memory and is only present unencrypted in the volatile memory. Another advantage of our solution is that the decrypted dex file has not to be stored in optimized form and thus can be deleted to prevent unauthorized access. The proposed technique also provides some degree of protection in case the device is rooted and process memory is accessed. Obfuscation and the use of JNI for code modification together make analysis hard. Apart from dalvik bytecode the attacker also has to analyze the native code which further delays the reverse engineering process. An important technique that we have used in this research is that decryption key is provided via smartcard by the user. Thus we do not have to store the key with the application. The decryption key is secured by encrypting it with users' public key so only authorized users can get access to the key. Cryptographic protection using Groupkey ensures that an attacker has to be a registered user which may also be used in future for tracking the attacker.

B. *Limitations and Special Requirements:*

The protection scheme presented in this research has some limitations as well. It cannot be used at large scale and is ideal only for a company or small group of people. Our proposed scheme does not protect the resources and assets of an Android app and protects only the dex file containing dalvik bytecode. The size of dex file should not be very large for this scheme to be applicable. This technique requires the Android device to be compatible with smartcards.

Software protection is a challenge and protection of Android apps is still a relatively new research. Android is basically built on an open source platform owing to which reverse engineering an Android app is easy as compared to applications based on some other platform. Therefore, when dealing with highly secure systems and information, it is better to put all the business logic and code on the server side. Using a real time service or remote server to deliver content is the best practice to prevent source code from reversing.

Appendix

Algorithm / Code

Encryption of dex file:

```
public class MainActivity extends Activity {
   //@SuppressWarnings("unchecked")
         @Override
       protected void onCreate(Bundle savedInstanceState) {
           super.onCreate(savedInstanceState);
           setContentView(R.layout.activity_main);
         try {

         //static void encrypt() throws IOException,
NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException {

       final String libPath = Environment.getExternalStorageDirectory() +
"/classes.dex";
       // Here you read the cleartext.
       FileInputStream fis = new FileInputStream(libPath);
       // This stream write the encrypted text. This stream will be wrapped
by another stream.
       FileOutputStream fos = new
FileOutputStream(Environment.getExternalStorageDirectory() +
"/encrypted.dex");

       // Length is 16 byte
       SecretKeySpec sks = new SecretKeySpec("thetoughpassword".getBytes(),
"AES");
       // Create cipher
       Cipher cipher = Cipher.getInstance("AES");
       cipher.init(Cipher.ENCRYPT_MODE, sks);
       // Wrap the output stream
       CipherOutputStream cos = new CipherOutputStream(fos, cipher);
       // Write bytes
       int b;
       byte[] d = new byte[8];
       while((b = fis.read(d)) != -1) {
           cos.write(d, 0, b);
       }
       // Flush and close streams.
       cos.flush();
       cos.close();
       fis.close();
   }
         catch (Exception e) {
                     e.printStackTrace();
                 }


   }

/*  @Override
```

```
    /*protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        try {
            final String libPath = Environment.getExternalStorageDirectory()
+ "/shoaib.jar";
            final File tmpDir = getDir("dex", 0);

            final DexClassLoader classloader = new DexClassLoader(libPath,
tmpDir.getAbsolutePath(), null, this.getClass().getClassLoader());
            final Class<Object> classToLoad = (Class<Object>)
classloader.loadClass("com.example.custom.MyClass");

            final Object myInstance  = classToLoad.newInstance();
            final Method doSomething = classToLoad.getMethod("doSomething");

            doSomething.invoke(myInstance);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }*/


    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is
present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

}
```

Decryption Via Dialog Box

```
public class MainActivity extends Activity {

   //@SuppressWarnings("unchecked")
   private static final String TAG = "MyActivity";
   @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

           AlertDialog.Builder alert = new AlertDialog.Builder(this);

                alert.setTitle("Title");
                alert.setMessage("Message");

                // Set an EditText view to get user input
                final EditText input = new EditText(this);
                alert.setView(input);
                alert.setPositiveButton("Ok", new
DialogInterface.OnClickListener()   {
                public void onClick(DialogInterface dialog, int whichButton)
{
                String value = input.getText().toString();
                try {
                             Decrypt(value);
                             Log.d(TAG,value);
                       } catch (InvalidKeyException e) {
                             // TODO Auto-generated catch block
                             e.printStackTrace();
                       } catch (NoSuchAlgorithmException e) {
                             // TODO Auto-generated catch block
                             e.printStackTrace();
                       } catch (NoSuchPaddingException e) {
                             // TODO Auto-generated catch block
                             e.printStackTrace();
                       } catch (IOException e) {
                             // TODO Auto-generated catch block
                             e.printStackTrace();
                       }
                //     catch (Exception e) {
          //           e.printStackTrace();
          //       }
                }});


                alert.setNegativeButton("Cancel", new
DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog, int
whichButton) {
                            // Canceled.
                       }
                       });

                        alert.show();      }
/*  static void encrypt(String Value) throws IOException,
NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException {

    final String libPath = Environment.getExternalStorageDirectory() +
"/shoaib.jar";
    // Here you read the cleartext.
    FileInputStream fis = new FileInputStream(libPath);
    // This stream write the encrypted text. This stream will be wrapped by
another stream.
    FileOutputStream fos = new
FileOutputStream(Environment.getExternalStorageDirectory() +
"/encrypted.jar");

    // Length is 16 byte
    SecretKeySpec sks = new SecretKeySpec(Value.getBytes(), "AES");
 //   String stringKey = Base64.encodeToString(sks.getEncoded(),
Base64.DEFAULT);
    Log.d(TAG,Value);
    // Create cipher
    Cipher cipher = Cipher.getInstance("AES");
    cipher.init(Cipher.ENCRYPT_MODE, sks);
    // Wrap the output stream
    CipherOutputStream cos = new CipherOutputStream(fos, cipher);
    // Write bytes
    int b;
    byte[] d = new byte[8];
    while((b = fis.read(d)) != -1) {
        cos.write(d, 0, b);
    }
    // Flush and close streams.
    cos.flush();
    cos.close();
    fis.close();
    Decrypt(Value);

}*/
static void Decrypt(String Key) throws IOException, NoSuchAlgorithmException,
NoSuchPaddingException, InvalidKeyException {

    final String libPath = Environment.getExternalStorageDirectory() +
"/encrypted.dex";
    FileInputStream fis = new FileInputStream(libPath);

    FileOutputStream fos = new
FileOutputStream(Environment.getExternalStorageDirectory() +
"/decrypted.dex");
 //   byte[] encodedKey     = Base64.decode(Key, Base64.DEFAULT);
 //   SecretKeySpec originalKey = new SecretKeySpec(encodedKey, 0,
encodedKey.length, "AES");
 //  Log.d(TAG,Key);
    SecretKeySpec sks = new SecretKeySpec(Key.getBytes(), "AES");
    Cipher cipher = Cipher.getInstance("AES");
    cipher.init(Cipher.DECRYPT_MODE, sks /*originalKey*/);
    CipherInputStream cis = new CipherInputStream(fis, cipher);
    int b;
    byte[] d = new byte[8];
    while((b = cis.read(d)) != -1) {
        fos.write(d, 0, b);
    }
    fos.flush();
    fos.close();
    cis.close();
}

/* @Override
    /*protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        try {
            final String libPath = Environment.getExternalStorageDirectory()
+ "/classes.dex";
            final File tmpDir = getDir("dex", 0);
            final DexClassLoader classloader = new DexClassLoader(libPath,
tmpDir.getAbsolutePath(), null, this.getClass().getClassLoader());
            final Class<Object> classToLoad = (Class<Object>)
classloader.loadClass("com.example.custom.MyClass");

            final Object myInstance  = classToLoad.newInstance();
            final Method doSomething = classToLoad.getMethod("doSomething");

            doSomething.invoke(myInstance);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }*/


    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is
present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

}
```

References

[1]. International Conference on. IEEE. International Monetary Fund. (2013) World Economic Outlook. www.imf.org

[2]. Business software alliance. http://globalstudy.bsa.org/2011. Last Accessed: 2013-10-28.

[3]. Arxan Technologies. Protecting .NET Software Applications, Arxan Best Practices, White Paper, http://www.softwaremag.com/pdfs/whitepapers/protecting-NET-software-applications-wp.pdf?CFID=14965377\& CFTOKEN=16715129

[4]. Abi research. https://www.abiresearch.com/press/android-mobile-app-revenues-will-reach-68-billion-Last Accessed: 2013-09-18.

[5]. G. Naumovich and N. Memon, "Preventing piracy, reverse engineering, and tampering. *Computer,* vol.36, no. 7, pp. 64-71, July 2003.

[6]. P. Sivadasan, P. SojanLa, and N. Sivadasan, "Jdatatrans for array obfuscation in java source codes to defeat reverse engineering from decompiled codes," in *Proc. 2nd Bangalore Annual Compute Conference,* Bangalore, India, January 2009, pp. 13. ACM.

[7]. A. Venkatesan, "Code obfuscation and virus detection,"M.S. Project, Dept. Comp. Science, San Jose State University, California, USA, 2008.

[8]. S. Schrittwieser and S. Katzenbeisser, "Code obfuscation against static and dynamic reverse engineering," In *Information Hiding*, pp. 270-284. Springer Berlin Heidelberg, 2011.

[9] P. Schulz, "Code protection in android," *Insititute of Computer Science, Rheinische Friedrich-Wilhelms-Universitgt Bonn,* Germany, 2012.

[10]. D. Kundu, "JShield: A Java Anti-Reversing Tool," PhD thesis, San Jose State University, California, USA, 2011.

[11]. S. Ghosh, S., S.R. Tandan, S. and K. Lahre, "Shielding android application against reverse engineering," ESRSA Publications, *International Journal of Engineering Research and Technology*. Vol. 2. No. 6, June-2013.

[12]. Piao, Yuxue, Jin-Hyuk Jung, and Jeong Hyun Yi. "Server-based code obfuscation scheme for APK tamper detection." *Security and Communication Networks* (2014).

[13]. McGraw, Gary. Software security: building security in. Vol. 1. Addison-Wesley Professional, 2006.

[14]. DexOpt. https://android.googlesource.com/platform/build/+/donut-release/tools/dexpreopt/dexopt-wrapper/

[15]. Smali. https://code.google.com/p/smali Last Accessed: 2013-09-18.

[16]. JAD. http://varaneckas.com/jad

[17]. Androguard. https://code.google.com/p/androguard

[18]Idapro. http://hex-rays.com/idapro

[19]. Dex dump 1.1.0. http://dex-dump.soft112.com

[20]. Dexter. http://dexter.dexlabs.org

[21]. Apktool. http://code.google.com/p/android-apktool

[22]. Dex2jar. http://code.google.com/p/dex2jar

[23]. JD GUI. http://jd.benow.ca

[24]. Shuttleworth, Martyn (2008). "Definition of Research". *Explorable*. Explorable.com. Retrieved 14 August 2011.

[25]. Creswell, J. W. (2008). Educational Research: Planning, conducting, and evaluating quantitative and qualitative research (3rd ed.). Upper Saddle River: Pearson.

[26]. A.G. Abbasi, S. Muftic, "Cryptonet: Integrated secure workstation," In *International Journal of Advanced Science and Technology,* vol. 12, pp. 1-10, November, 2009.

[27]. obfuscation using code packing. http://www.foocodechu.com/?q=node/55/ Last Accessed: 2013-03-13.

[28]. Cipher. http://developer.android.com/reference/javax/crypto/Cipher.html/ Last Accessed: 2013-02-11.

[29]. http://nelenkov.blogspot.com/2012/04/using-password-based-encryption-on.html Last Accessed: 2014-06-25

[30]. Dexfile https://android.googlesource.com/platform/libcore-snapshot/+/ics-mr1/dalvik/src/main/java/dalvik/system/DexFile.java/ Last Accessed: 2014-01-15.

[31]. Gunasekera, Sheran. "Android Security Architecture." *Android Apps Security*. Apress, 2012. 31-45.

[32]. Proguard. http://proguard.sourceforge.net/

[33]. Dexguard. http://www.saikoa.com/dexguard/