Java Application Development using JNI on Android

Chang Pei Shan, Sea Chong Seak, and Wong Hon Loon

***Abstract*—**In this paper, we sharing experience on Java Android development by using JNI which is a Java interface to converge with non-Java code. Nowadays, the Android platform widely used for mobile devices can be applied to embedded systems. Therefore, Android developers around the world are creating and compiling Android application by using Java language which provided by Android SDK for embedded systems operated via Android platforms. However, many Android developers still interested to implement their application by using library from other programming language such as C/C++. It is the reason JNI comes in to the picture and plays an important role to provide an interface which enables the Java able to interact with libraries of others platform. This paper describes the difficulties faced and solutions when doing implementation of Android application development by using Java JNI.

***Index Terms*—**JNI, Android Application, NDK, native codes, secure algorithm, OPT code, cryptography, authentication

I. Introduction

Today, Tablet PC and Smartphone are become part of our life as these smart devices have become more powerful and unable to replace by other electronic items yet. The most popular mobile operating systems are Apple's iOS, Microsoft's Windows Mobile and Google's Android [1]. In this paper, we focus development experience on the Android platform which is the most famous used embedded OS nowadays; hence, the applications inside Android OS are important and widely used by the entire mobile user.

Android provides an open development terminal which is Android SDK for developers writing applications that commonly called apps to extend the functionality of the devices [2]. There are currently over millions apps available for Android users. Android Market is the online apps store operates by Google. The Android applications are mostly written in Java by developer and controlling the device via Google-developed Java libraries [3]. However, mobile devices having a common issue which have limited storage and constrained battery life [4], as a result the developers should pay more attention to the efficiency of the program when they develop Android applications. It is because Java is slow when handling hardware resource and complex operation such as calculation in image processing or authentication process. Due to this, it is necessary to use native codes to overcome the memory management and performance constraints in Java. Therefore, Android OS is supporting JNI (Java Native Interface) with provide Android NDK which is a toolset that let developer embed components to enable native code languages is workable in Android applications [5]. The native code languages are such as C and C++. JNI is the Java Native Interface. It defines a way for managed code which written in the Java programming language to interact with native code. It's vendor-neutral, has support for loading code from dynamic shared libraries [6], and while decrease the execution times is reasonably efficient.

The rest of paper we organized and structured as follows. In Section II, we look into background of Android and the JNI interface. Session III we briefly introduce related work which involved our design and implementation follow. In Section IV, we describe the architecture design of our implementation by using JNI and the difficulties we faced through the development process. In section V, we show the solution we used to solve the issues when implement JNI in our design. Benefits of JNI to our implementation work will be explained in session VI. Finally, we describe the lesson learnt and future work in the conclusion section.

II. Background

A. Android

Android Inc was founded in Palo Alto of California, U.S. by Andy Rubin, Rich miner, Nick sears and Chris White in 2003. Later Android Inc. was acquired by Google in 2005 [7]. Android is a software environment built for mobile device by leveraging its Linux Kernel to interface with the hardware [8]. The Android platform is a device-independent platform which means that developers can create application for any devices such as smartphone, TV, ebook reader, gps and etc. Android is a software stack for mobile devices that includes an operating system, middleware, and key applications [9].

The major five components of the Android operating system as shown in Figure 1 included Linux kernel based OS, a rich UI, end user application such as maps and camera, Android runtime which including Dalvik Virtual Machine and core library, also the application framework which for phone functions [10][11]. However, the component of the underlying OS are written in C or C++, end user application are built for Android in Java and built-in application are written in Java [9].

Android applications are written in the Java programming language. All the code in a single Android package file is considered to be one application [4]. Each application runs in its own Linux process and performs a different role in the overall application behavior, and each one can be activated individually. Furthermore, each managed piece of code executes in a virtual machine (DVM). As a result each application is sand-boxed from the other applications running at any given time.

Manuscript received June 18, 2015.

P. CHANG. Author is with Information Security Lab, MIMOS Berhad, Technology Park Malaysia, 57000 Kuala Lumpur, Malaysia.

C. SEA. Author is with Information Security Lab, MIMOS Berhad, Technology Park Malaysia, 57000 Kuala Lumpur, Malaysia.

H. WONG. Author is with Information Security Lab, MIMOS Berhad, Technology Park Malaysia, 57000 Kuala Lumpur, Malaysia.

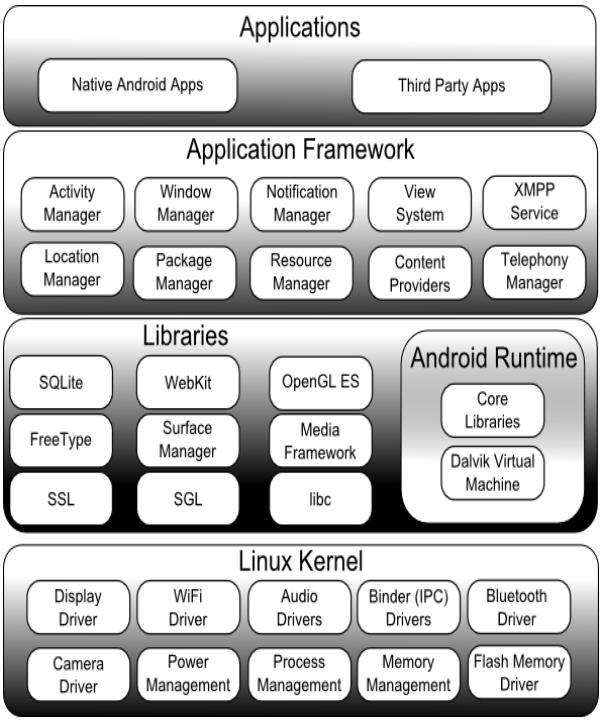

Fig. 1. Android Architecture

B. *Android NDK and JNI*

Android NDK was first released in June 2009 [5]. It is a toolchain that provide a secure environment to native library to be loadable inside Java development. It enables Android application developers to manage devices in detail beyond the limit of the framework and to reuse legacy code written in C/C++ language easily [12]. By using NDK, the Android applications able to improve the performance by handling hardware resource easier and enable to develop in higher speed application. Before using NDK, developers are required to first familiarize with Java JNI programming because JNI is the prerequisites of using NDK tool.

JNI technology is the part of Java Virtual Machine (JVM) implementation, Java offers JNI as an interface connecting the world of Java to the native code [13]. It is a bridge to build up the communication gap between two totally different languages which is Java and native codes. JNI technology enables the procedure and the class libraries which other languages such as C/C++ able to access Java environment by using the Java method and objects. While, Java also can facilitates interoperate the application which involved varies functions and libraries of native codes. As a result, JNI allows developers to take advantage of the power of the Java platform, without having to abandon their effort in developed the legacy code [1]. Figure 2 indicates the relations among native code such as C/C++, JNI interface, Java codes and Android system.

As a two-way interface, the JNI allows Java application to invoke native codes which is native libraries and native application [3], yet vice versa. When a native function is called, one of the arguments passed to the function is the JNIEnv interface pointer which is a gateway to access all the Java fields and predefined JNI functions. The JNIEnv interface pointer links to thread-local data and is organized like a C++ virtual function table. Therefore, JNIEnv cannot

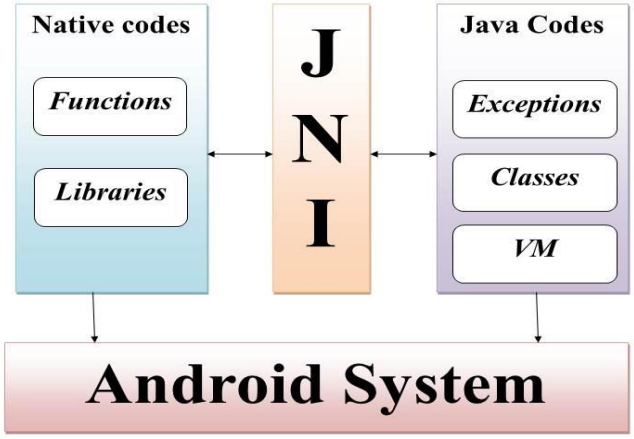

Fig. 2. Relations among Native Code, JNI Interface, Java Codes and Android System

be shared and only accessible by Java thread. The other arguments passed to function when native function is called are the instance or static method information.

III. Related Works

A. *The Implementation Environment*

We used Android version 4.4.2 KitKat on Eclipse platform 4.2.1, Android Development Toolkit 23.0.3.1327240. We also used Android NDK r9.

B. *Implementation*

In this paper, the related works for implement the C/C++ libraries in Java through JNI are to develop network security architecture to generate an OTP code by mobile application for authentication purpose as shown in Figure 3. The most important component in the client application is the security functions, which implements all the security related operation for example hashing HMAC, key decryption and key encryption. The implementation should be hidden and kept secret. Therefore this component has to be implemented in C++ to prevent the mobile application from being reverse-engineered.

We aimed that both Android and iOS will be using the same approach and share the same security function modules. Both JNI and security functions resides in the same C++ file. For reusability purpose, the security function will need to be separated from JNI. The JNI/Objective C++ should only serve as an interface between the app and C++ without any real security implementation.

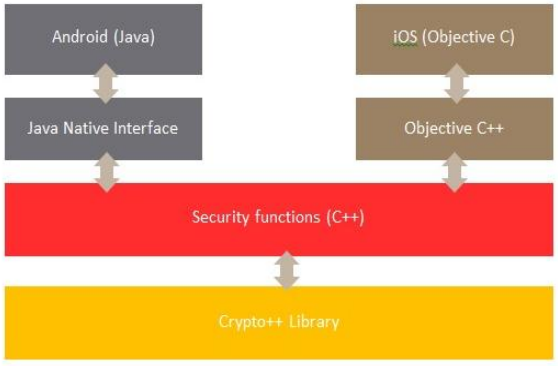

Fig. 3. Client Architecture for both Ios and Android

The functions of each component in our architecture are listed inside Table 1 as below.

TABLE I
FUNCTIONS OF COMPONENTS FOR IMPLEMENTATION

Component	Description
Android	Android Program Implemented In Java
Ios	Ios Program Implemented In Objective C
Java Native Interface (JNI)	A Programming Framework That Enables Android Java Program To Call The C++ Component. This Component Should Only Serves As An Interface Without Real Security Implementation To Protect The Design Algorithm
Objective C++	Same As JNI But Different Programming Language As This Framework Is For Ios.
Security Functions	Contains Implementation Of Security Related Function And Operation. This Component Has To Be Protected.
Crypto++ Library	A Third-Party Open Source Crypto Library.

IV. THE ARCHITECTURE DESIGN

To confirm both Android and iOS will be using the same approach, we aware most of the native support on advanced cryptographic operations, such as elliptic curve cryptography, varies across mobile platforms having an existing C++ security function, hence, JNI is needed to implement for this architecture design. In Figure 4, it is the authentication flow by using OTP code which generated by client.

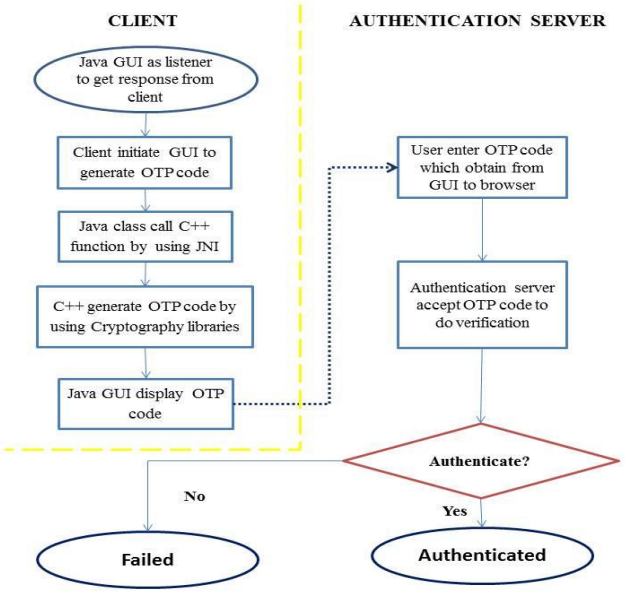

Fig. 4. Authentication Flow by Using OTP Code From Client

From the figure, the client has a Java GUI which is display on mobile device application as the listener to wait the response from user. Then, once the user initiate the listener, the Java class will call the C++ function by using JNI and generate the OTP code with the C++ cryptography libraries. The Java GUI will show the OTP code after the process. Lastly, the OTP code is a verification ticket to enter inside browser to complete the authentication process.

We have difficulties to design an architecture flow with creating a universal code base using a programming language which can be compiled in different mobile platforms to simplify the development lifecycle on mission-critical modules. For different OS implementation, all the secure functions are written inside a C++ file and use its own language to create a wrapper to invoke the C++ functions when it is called. Therefore, the routine of wrapper using should be synchronized while programming language is not the same.

As in Android, we were facing issue on the method to store and obtain some variables by through JNI function. Normally we able to obtain and store IMEI parameter through Java API which is existing method for Java users as show in Figure 5.

```
// Obtain IMEI with Java API
const char* getIMEI(JNIEnv* env, jobject thiz) {
    // Obtain the IMEI
    jmethodID mid = env->GetMethodID(env->GetObjectClass(thiz),
            "getSystemService", "(Ljava/lang/String;)Ljava/lang/Object;");
    jobject telephony_manager = env->CallObjectMethod(thiz, mid,
            env->NewStringUTF("phone"));
    mid = env->GetMethodID(env->GetObjectClass(telephony_manager),
            "getDeviceId", "()Ljava/lang/String;");
    jstring imei_string = (jstring) env->CallObjectMethod(telephony_manager,
            mid);
    const char *imei = env->GetStringUTFChars(imei_string, 0);
    return imei;
}
```

Fig.5. Obtain IMEI with Java API Example

Lastly, problems to simplify some steps of the designs which are keep repeating in different scenarios.

V. SOLUTIONS

In our architecture design to tally the approach of iOS and Android, we build a hierarchy file which is a wrapper for us to follow the flow and function correctly. It is significant to create as the interface for communicates with the Crypto C++ libraries as show in Figure 6. Besides, we have to confirm the input and output parameters are defined correctly in each JNI functions. Therefore, the approaches, flows, input and outputs parameter of the design architecture will be exactly the same for iOS and Android although the method to obtain the output may different inside the functions.

```
@interface MiTCKWrapper : NSObject

- (id) init;
- (BOOL) methodOneWithParam1: (NSString *) param1;
- (BOOL) methodTwoWithParam1: (NSString *) param1;
- (BOOL) methodThree;
- (BOOL) methodFour;
- (BOOL) methodFiveWithParam1: (NSString *)param1;
- (BOOL) methodSixWithParam1: (NSString *) param1 andParam2: (NSString *)param2;
- (BOOL) methodSevenWithParam1: (NSString *) param1;
- (void) methodEight;
- (NSString *) methodNineWithParam1: (NSString *) param1;
- (NSString *) methodTen;
- (void) methodEleven;
- (int) methodTwelve;
```

Fig.6. Wrapper as Interface to Call Native Codes

For the storing variable issue, we created a solution by open file to store the particular value inside and read it when we needed for our design as show in Figure 7.

```
const char* SECRET = "/data/data/my.mimos.isluan.mitck/files/secret";

bool saveSecretFile(const char* code) {
    FILE* file = fopen(SECRET, "w+");
    if (file != NULL) {
        fputs(code, file);
        fflush(file);
        fclose(file);
        __android_log_write(ANDROID_LOG_DEBUG, "MIMOSTCK-JNI",
                "Secret written");
        return true;
    } else {
        __android_log_write(ANDROID_LOG_DEBUG, "MIMOSTCK-JNI",
                "Secret failed to write");
        return false;
    }
```

Fig.7. Store Variables by Using File Example

With this solution, we can simplify steps as define the functions in public function, hence, it allows calling by any JNI functions in the same architecture.

VI. Benefits Of JNI

One of the main features to use JNI in our implementation is because of secure key management. It is because we involved multiple encryption, decryption and cryptography formula to create the signature key which is the OTP code. Then, those applied security formula such as C and C++ are compiled directly into assembly codes which are harder to be reverse-engineered than Java bytecodes.

Also, it able to protects our security algorithm for OTP code generation by using JNI. This means writing mission-critical modules in C and C++ can provide a stronger defense against network attacks aiming to disclose the underlying application logics.

From the development experience, we noticed native codes can execute in lower speed under correct usage when involved varies complex math calculation such as cryptography operations. Besides, JNI enables our application to do cross-platform development which involved iOS and Android. This is good for reusability purpose which uses the existing legacy library and could not afford to rewrite in certain language only such as Java code. As a result, we not only can fully utilize the existing native code and in some cases increased speed of mobile apps development by saving the developer time.

VII. Conclusion

In this paper, we aimed to implement a security mobile application which involved tons of complex cryptography functions for authentication purpose in iOS and Android platform.

At first sight, we plan to create different programming libraries to support different mobile platform development which is not the efficient way. After we research for the related requirement, we found out that the JNI function enables our implementation sharing the same native code libraries as cross-platform development which supports Android and iOS. It allows time saving in both development and apps execution, while still providing better performance in our design. Besides, by adding JNI to our implementation, it is adding the complexity of the security apps which become harder to be reversed-engineer.

In conclusion, we recommend that Android application developers use the native C/C++ library through JNI function when the implementation requires number of memory access, multiple complex formula calculations and stronger defense to against network attack.

Acknowledgement

This research was supported by Mr Ng Kang Siong who acts as our technology architecture reviewer for the network security design. Also, architecture programmer was supported and consulted by Dr Cheong Hoon Sin in our implementation for both iOS and Android platforms.

References

[1] Jae Kyu Lee, "Android Programming Techniques for Improving Performance", *Awareness Science and Technology (iCAST) Internation Conf*, pp386-389, Dalian, 2011

[2] Yeong-Jun Kim, "Benchmarking Java application using JNI and native C application on Android", in *Control, Automation and Systems (ICCAS), Jeju Korea*, pp 284 – 288, 2012

[3] Ki-Cheol Son, "The method of android application speed up by using NDK", *Awareness Science and Technology (iCAST) Internation Conf*, pp382-385, Dalian, 2011

[4] Yann-Hang Lee, "Efficient Java Native Interface for Android Based Mobile Devices", in *International Joint Conference of IEEE TrustCom*, pp 1202-1209, 2011
http://dx.doi.org/10.1109/trustcom.2011.162

[5] Sangchul Lee, "Evaluating performance of Android platform using native C for embedded systems", *Control Automation and Systems (ICCAS), International Conference*, pp1160-1163, Gyeonggi-do, 2010

[6] JNI Tips, http://developer.android.com/training/articles/perf-jni.html

[7] W. FRANK ABLESON, "Android.in.Action.2", Stamford, 2011

[8] David Ehringer, *The Dalvik Virtual Machaine Architecture*, http://davidehringer.com/software/android/The_Dalvik_Virtual_Machine.pdf

[9] Google, *Android 2.3 User Guide*, www.google.com/googlephone/AndroidUsersGuide- 2.3.pdf

[10] Android (Operating System), http://en.wikipedia.orglwiki/Android (operating_system)

[11] Android, http://www.engineersgarage.com/articles/what-is-android-introduction

[12] Java Native Interface, http://www.cs.iastate.edu/~yingcai/cs587x/notes/jni.pdf

[13] Android JNI/NDK, http://www.slideshare.net/dumura/android-jni-37395860

P Chang, was born in Malaysia in 1984. She received a degree of Bachelor Electrical and Electronic Engineer from Nottingham Trent University in 2008.
She has around 3 years working experiences in Intel Penang Design Center working as structural design automation engineer in VLSI design CAD tools and methodology application support require in state-of-the-art SOC development team. Also, she has 2 years working experience in Acer Taiwan working as mobile software project engineer in Android phone production line test software development which applied Qualcomm CDMA chipsets and tools support. For recent years, she was back to Malaysia and join in MIMOS Berhad as member, to works as senior engineer which supporting network security platform and as a Java programmer on Android development