DOI: 10.3969/j. issn. 1673-5188. 2016. S0. 005
http://www.cnki.net/kcms/detail/34.1294.TN.20160622.1830.002.html, published online June 22, 2016

SeSoa: Security Enhancement System with Online Authentication for Android APK

DONG Zhenjiang[1], WANG Wei[1], LI Hui[2], ZHANG Yateng[2], ZHANG Hongrui[2], and ZHAO Hanyu[2]
(1.Cloud Computing and IT Research Institute, ZTE Corporation, Nanjing 210012, China;
2.Beijing University of Posts and Telecommunications, Beijing 100876, China)

Abstract

Android OS provides such security mechanisms as application signature, privilege limit and sandbox to protect the security of operational system. However, these methods are unable to protect the applications of Android against anti-reverse engineering and the codes of such applications face the risk of being obtained or modified, which are always the first step for further attacks. In this paper, a security enhancement system with online authentication (SeSoa) for Android APK is proposed, in which the code of Android application package (APK) can be automatically encrypted. The encrypted code is loaded and run in the Android system after being successfully decrypted. Compared with the exiting software protecting systems, SeSoa uses online authentication mechanism to ensure the improvementof the APK security and good balance between security and usability.

Keywords

software protection; anti-reverse; Android; authentication

1 Introduction

Because of its true openness, Android [1] has become one of the most popular mobile platforms in the world since its appearance in the market a few years ago. Naturally, the number of available applications is increasing rapidly and now over 1,800,000 Android applications can be chosen by the Android users according to the statistical result of Google Play [2]. However, the open source code of Android and a huge number of applications pose a vital security challenge for the Android system. More and more attackers aim to its flaws, not only the flaws of the OS itself, but also the flaws of the applications, which are hard for the developers of Android system to fix.

Among many security threats of Android system, the Android applications' source codes face the risk of being decompiled and may be acquired by illegal organizations through reverse engineering techniques. This threat may result in that the program is slightly modified, re-packaged and released in the form of new applications; even worse, attackers may tamper with the code, mix malware to obtain the user's sensitive data, steal his bank account numbers and passwords stored on the phones, or to increase the charges by triggering some pay-needed operations. It is difficult for the Android existing security mechanisms to detect and prevent this kind of threats, which may make users suffer from certain direct or indirect economic losses.

To avoid the above mentioned threats, researches have proposed some mechanisms to protect the software against anti-reverse. One of the solutions is code obfuscation [3], which can be used to transform the original code into a form that makes reverse engineering harder and more time consuming and at the same time still possess the same function as the original software has. In general, the code obfuscation techniques include renaming of the identifiers of the variables, constants, classes, methods, etc. in the program. Such an obfuscation tool is ProGuard [4], which is integrated into the Android software development kit. In [5], a modification of Tiny C Compiler (TCC) , a simple compiler, is proposed to modify certain unconditional branching instructions to conditional branching and to make confusing conversion on the critical data of the software. This modification aims at misleading automated reverse engineering tools [6] to detect the original code. However, code obfuscation just makes a simple change of the names of classes or variables. If only this method is used to protect Android apps, as long as the .dex files are found, they can be decompiled into .smali or .java, and then be reversed.

To protect the software more effectively, new tools as DexGuard [7] and Allatori [8] appear, which also support a number

This work was supported by National Natural Science Foundation of China (61370195) and ZTE Industry-Academia-Research Cooperation Funds.

of other features such as control flow randomization, string and class encryption, and temper detection. In 2014, Piao [9] reveals that DexGuard has some weaknesses and an attacker is able to analyze the hex code of a Dalvix executable file. Piao suggests to store the core execute class file through obfuscation on the server; in this way, when an application needs to execute core routines, it must request these routines from the server, download it and maybe decrypt it. Our analysis shows that Piao's solution requires the availability of server all the time while the protected Android application is running. To overcome this shortage, a security enhancement system with online authentication for Android application package (APK) called SeSoa is proposed in this paper, in which Android applications are only authenticated by the authentication server before being run for the first time after they are downloaded and installed, which reduces the burden on the server.

The rest of this paper is organized as follows. In Section 2, related work is discussed and the existing security problems related with application protection in the Android security model and the security goals in SeSoa are analyzed. In Section 3, a novel solution of protecting software in Android platform is provided. In Section 4, the security of the proposed solution is analyzed. And finally, the conclusions are presented in Section 5.

2 Related Work

2.1 Problems in Android Security Model

Because of the open-source feature, Android requires strict security specification and robust security architecture. The security feature is reflected in the system security design structure, including all aspects in the application framework layer and the core layer. At the application layer, there are signature mechanism and application access control mechanism to protect the application security. At the kernel level, Android obtains its security goals based on the security features of Linux kernel systems. Therefore, the resources of different processes are well isolated by the sandbox mechanism, the unique memory management mechanism and the efficient and safe inter-process communication mechanism. Such security mechanisms provided by the Android system achieve the security goals to some extents, but they, especially Android signature and sandbox mechanisms, fail to protect the software of application from anti-reverse.

Firstly, Android system tries to protect the security of the application using the signature mechanism, which means that the applications installed in Android system are required to be signed by some institutes. In fact, the signature mechanism is not just used to identify the application's developer, it is also used to detect whether the application has been illegally tampered with. If the answer is yes, the program's signature cannot pass the verification of Android system, so the result is that the tempered application does not run correctly. Android system allows application with self-signed, which means that the applications can be signed by the developers themselves. If the signed APK package is decompressed, a folder named META_INF will be found, in it is the signature information of the applications. The folder contains MANIFEST.MF, CERT.SF and CERT.RSA files, multiple RSA file appears if multiple certificate signing is used. MANIFEST.MF is the main APK summary information, such as the APK information, application properties and the hash values of all files. CERT.SF is the signature file obtained using SHA1withRSA signature algorithm, which contains a summary of the application signature value. Attackers often get the correct signature information for the software by analyzing the file and tamper the software illegally, regenerate the signature and rewrite these files to pass the verification of the Android system.

Secondly, since Android is a multi-process system, isolating the resources for each application is a basic requirement for security. The sandbox mechanism is adopted in Android system to make sure that every application's process is a secure sandbox running in its own instances with a unique ID (uID) assigned to it. With such a mechanism, each application is running in a separate Dalvik virtual machine (DVM), with a separate address space and resources. As DVM is running on the Linux process and dependent on the Linux kernel , Android uses DVM and Linux file access control to achieve the sandbox mechanism. Any application that wants to access for system resources or other application permission must be declared in its manifest files or shared uID. But this sandbox isolation technology on Android also makes the code of its application face the threat of being decompiled. The reason is that the Java language application needs to be compiled into a binary byte code, which is the intermediate code running on DVM and can more easily be decompiled based on the Java decompile technology, be reverted from the original code to the logic results and get the identifiers names or other information.

In addition, due to the open source feature of the Android platform, the decompile technology for the applications on Android platform has been fully studied, thus through a number of mature disassembly tools, it is not difficult to get the Smali code of the software and then the source code through the reverse analysis. This makes the study on how to protect the code effectively in Android platform very important.

2.2 Goals of Application Protection

Generally, a software protection system should meet the following secure requirements.

1) Anti-tamper, preventing the application from being modified by some attackers
2) Preventing dynamic debugging, i.e., preventing attackers from getting the source code of application by using dynamic debugging tools;
3) Preventing decompilation, i.e., preventing attackers from getting the source code of application by using decompile

tools.

3 Proposed Solution

3.1 SeSoa Overview

To meet the above-mentioned requirements, a new solution called security enhancement system with online authention (SeSoa) is proposed. The main idea is using an authentication server to verify whether the APK is enhanced by the authorized party and integrity of the APK. The SeSoa consists of the operator, authentication server (AuS), security enhancement server (SeS) and Android client equipment. The main security reinforcement software is running on the SeS to generate templates packers, encrypt the APK's core files, regenerate signature of APK through implement the script file, and repackage the new APK. The authentication server is used to verify the integrity of the downloaded APK at the Android client side and source of APK. The Android client is mainly responsible for running the environmental monitoring, integrity verification and decryption of encrypted part of code and loading the original APK code. **Fig. 1** shows the SeSoa system and its operational flows.

1) Input APK

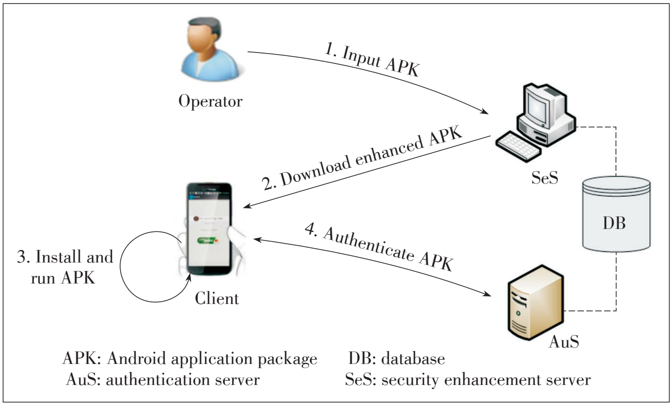
The operator of SeSoa submits the original unprotected APK to SeS. SeS completes all the functions of security enhancement: 1) using the encryption technique to encrypt the dex code in the application, so that the plaintext code no longer appears on the client, which ensures that the code is protected from static analysis; 2) adding the safety function code to make sure that the application can be protected from dynamic debugging; 3) adding the code supporting authentication function so that the client could run the particular APK only after the completion of the authentication by the AuS.

2) Download enhanced APK

The enhanced APK can be downloaded by every Android user to the Android mobile terminals.

3) Install and run APK

The users install and run the enhanced APK.



▲**Figure 1. System framework and operational flows.**

4)Authenticate APK

The enhanced APK needs to complete the online authentication function when it is run for the first time. Only after it passes the authentication by the AuS can the enhanced APK get the security parameters, which allow it to continue running and move into the APK original function module.

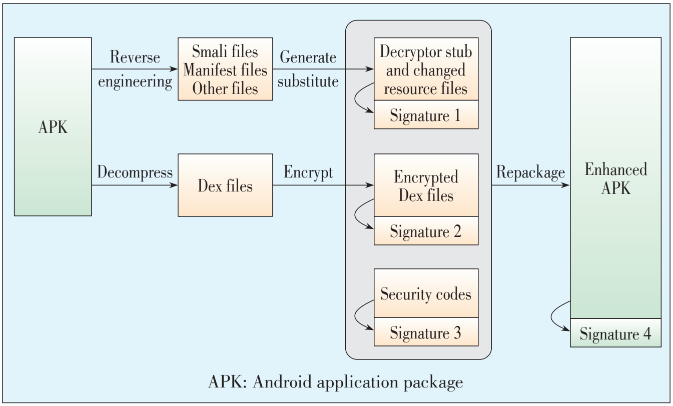
In addition, the SeS and AuS are logical concepts, which can be separated or implemented on one equipment. However, they can share the same Database to reduce the complication of management for data consistency.

3.2 Security Enhancement Process

The security enhancement process is implemented on SeS (**Fig. 2**). At the first step, the original APK file needs two pretreatment functions. One is to gain information about the APK and its implementation details, including the Smali and Manifest files using reverse engineering tools such as APKTool [10]. The other is to decompress the APK to get the .dex file. After the pretreatment process, the files are ready to be used in the following steps.

The two components of the encrypted application and a decryptor stub are generated at the second step. Unlike Proguard and Dexguard using obfuscation to make the code harder to be understood, SeSoa encrypts the .dex file and decrypts it before it is loaded into memory. In this way, the .dex file will not be stored as the plain text in the file system, which means that the others must decrypt it first if they want to obtain the code. To make the encrypted .dex file run normally in the Android system, the decryptor stub has to be implemented to fulfill the fetch of the encrypted application into memory, decrypt the application and yield the original .dex file, which can be loaded into the DVM and executed. Because the decryptor stub should be run first, so we have to change the manifest file to ensure that the application can be run normally. These changes include replacement of the component names, modification of the primary activities property, addition of some new services and activity components, etc.

In order to attack such a security enhancement scheme, a reverse engineering tool has to gain access to the decrypted .dex



▲**Figure 2. The security enhancement process of SeSoa.**

file. This can only be done after the .dex file is successfully decrypted and loaded into the memory. To protect from such kind of threats, some other security mechanisms such as anti-dynamic debug function are needed. More importantly, the client also has no information about the source of the APK through this method, so the authentication mechanisms should be applied to make sure that the APK is from the authorized party. All these security mechanisms besides encryption/decryption are included in the security code module in Fig. 2.

At the third step, the signature of the resource files (signature 1), the signature of the original .dex files (signature 2) and the signature of security codes (signature 3) are generated separately according to different parts of data. These signatures are used to protect the APK from being tampered.

Finally, all the files in the application are repackaged, and need to be resigned as any normal applications should do, as the Android system requires that developers must sign applications with their private key/certificate unless the application cannot be installed on user devices. Furthermore, because of the online authentication function, the data (application ID, secret K, CKSUM, Addr) are sent and stored in the database on AuS for further use.

3.3 Application Load Process

After APK is downloaded and installed in the Android client side, the decryptor stub program is loaded and run at once. The decryptor stub calls the online authentication module to communicate with AuS, and only those applications that pass the authentication verification can get the correct address of the required Dex. Then the Key used for encrypting .Dex is retrieved and used to decrypt .Dex file, and the successfully decrypted .Dex file is the original .Dex file of the application, which can be loaded into DVM and executed. **Fig. 3** shows this application load process.

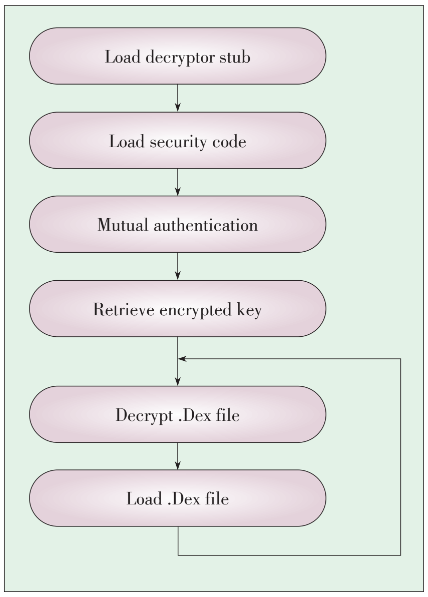


Figure 3. Application load process.

In fact, the process is not as easy as what is described above because we have to make sure that the whole process about decryption is secure and other applications cannot get the plain text of .dex file through the process. DVM also has limited instruction set, so there is no direct way to access the bytecode with an instruction, which makes it impossible to execute the bytecode stream during the running time. To circumvent this restriction, "Java Native Interface" (JNI) of the DVM is used, JNI is intended to allow execution of native code [11]. The mutual authentication process is another complicated process, the detailed authentication protocol and processes in SeS and Android client are described in the following section.

3.4 Online Authentication Process

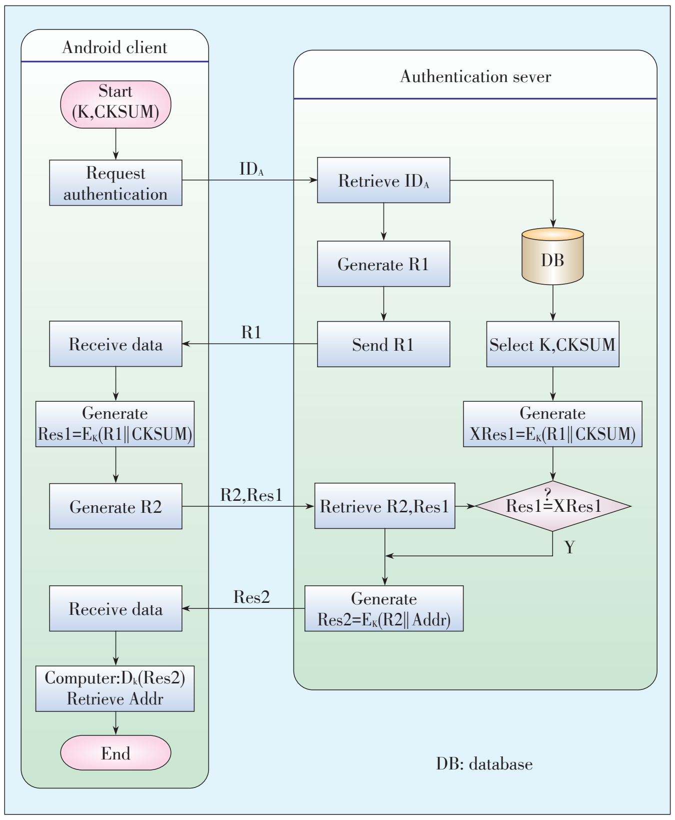
The online authentication process is used to achieve the mutual authentication between the Android client and authentication sever when an app is installed and run for the first time. Considering the performance of the implementation on Android side, only the symmetric cryptography is used in the scheme. **Table 1** lists the notations used in this section.

Fig. 4 shows mutual authentication procedure of the proposed Android application security enhancement scheme. This procedure is triggered by the application at the Android side after the decryptor stub starts and finishes generating the check sum of the application (CKSUM), as well as retrieving the K from the application.

1) The application on Android client side sends an authorization request message to the AuS, which contains its own identifier ID_A;
2) AuS receives the message and retrieves ID_A from the authentication request message, then it can use ID_A to select K and CKSUM from database, which are stored in the database by SeS. At the same time, the SeS generates a random number R1, and sends it to the Android client. After that, it calculates XRes1 using such symmetric algorithm as DES or AES with R1 and CKSUM as the input parameters and K as the key. XRes1 will be stored and used later.
3) When the client application receives R1, it can calculate Res1 with the same algorithm running on AuS, using K and

Table 1. Notations

Notions	Description
K	Symmetric key
CKSUM	Integrity check value
ID_A	Identification of application
Addr	Start address of main Dex program
R1/2	Random number
Res1/2	Response value
XRes1	Expect response value
E_K	Encryption using K
D_K	Decryption using K



▲Figure 4. Authentication scheme.

CKSUM as the algorithm's inputs. Then the app generates random number R2, and sends R2 and Res1 back to the SeS.

4) After SeS retrieves R2 and Res1, Res1 is compared with XRes1. If they are equal, it is shown that the application in Android is enhanced by SeS and the application has not been modified. If they are not equal, the app cannot get the K and correct CKSUM, and AuS will send error number to the client and stop the process. Secondly, SeS generates Res2 using encryption algorithm, Res2 equals $E_k(R2\|Addr)$. Finally, SeS sends Res2 back to the client.

5) At the Android side, Res2 can be retrieved from the message and decrypted to $R2'\|Addr'$ by the decryption algorithm using the same key K. Then R2' and R2 is compared, if they are equal, it means that this message is sent by the real SeS, and the application can regard addr' as the correct security parameter, which can be some code in the initial process or some key data needed by the application; otherwise, the application can react as what its security strategy allows.

4 Experiments and Evaluations

In this section, the test results of SeSoa on compatibility and performance are presented, which indicates that SeSoa is usable. The security of SeSoa is also analyzed to ensure that it can also meet the requirement for security.

4.1 Compatibility Analysis

Besides the security goals, SeSoa should also possess the capability of enhancing arbitrary application automatically. To test this feature, 50 applications in Android were downloaded and applied in our test. These applications were randomly selected from Android App Store, loaded and run in the 100 types of mobile phones with Android 4.0-4.4 platforms. For each application, we first evaluated whether it could be enhanced by SeSoa. If the answer is positive, we evaluated whether it could be loaded and run in the 100 types of smartphones. Experimental results are shown in **Table 2**.

The success rate for SeSoa to enhance the random applications is 82%, which is acceptable, although not very high. However, as the complexity of the applications increases, the success rate of the reinforcement system will decline. Therefore, it is necessary to further optimize the program.

4.2 Performance Analysis

In order to test whether SeSoa seriously affects the performance of the original application, ten test applications were randomly selected out of the above mentioned 41 apps that were successfully run on our 100 test mobile platforms. Because the running time varies on different platforms, the experiments presented in this section were conducted using Samsung Galaxy S4 smartphone, equipped Exynos 5410 dual quad-core processor and 2 GB RAM.

Table 3 shows the test results of performance. It can be seen that after being enhanced by the SeSoa, the installation package size of the applications increases about 10%, the average start-up time is increased by about 20%. Since the start-up time is generally less than 1 sec, this will not make users feel different compared with the start-up time of original APPs.

4.3 Security Analysis

1) Anti-tamper

We use the multiple signature mechanism to prevent the application from being tampered. In the security enhancement process, three signatures are generated at the third step. Therefore, at the client side, the signature of the resource files, the signature of the original .dex files and the signature of security code are verified during the application's start-up process, be-

▼Table 2. Test results for compatibility

App size	App numbers	Enhanced APP numbers	Success numbers for loading	Success numbers for runing	Platform compatibility
<1 MB	11	11	11	11	99.3%
1 MB-5 MB	25	23	23	22	98.6%
>5MB	14	10	9	8	96.4%
Sum	50	44	43	41	98.1%
Success rate	-	88%	86%	82%	-

▼ Table 3. Test results for performance

App no.	Original app size (KB)	Enhanced app size (KB)	Size change rate (%)	App startup time (ms)	Enhanced app startup time (ms)	Startup time increase rate (%)
1	560	581	3.8	284	365	28.5
2	740	784	5.9	415	467	12.5
3	813	897	10.3	396	452	14.1
4	1123	1260	12.2	675	813	20.4
5	1457	1634	12.1	561	697	24.2
6	2360	2523	6.9	741	879	18.6
7	2709	2942	8.6	759	896	18.1
8	3234	3558	10.0	737	834	13.2
9	4921	5489	11.5	891	1019	14.4
10	8913	9974	11.9	919	1131	23.1

fore the decryptor stub begins to decrypt .dex files. If any of the three verifications fail, the application will stop running. In this way, any tamper of the application will be detected.

The other anti-tamper technique we used is online authentication. Only the application that has passed the verification of AuS can obtain the secure parameters to run the core code in the APK. Any modification of the APK will lead to the failure of generating the correct CKSUM and secret K, so the APK on the Android client can hardly calculate the correct response of R1, which can be regarded as the challenge of AuS in a challenge-response protocol.

2) Preventing dynamic debugging

In our solution, a monitoring process ring is used to monitor whether there are some system calls for Ptrace, which is used by debuggers and other code - analysis tools to analysis the code.

The mainstream debugging tools use Ptrace system calls in Linux system; if a process is calling Ptrace, there must be someone using such kind of tools to debug and that the application should be stopped. This process for monitoring the call of Ptrace is used by most application enhancement systems, but the risk for such a method is that attackers can stop the monitoring process and then debug the application again. To avoid this weakness, we designed the monitoring process ring mechanism.

Our monitoring process ring scheme has three processes: the parent process (the main process to be protected), the child process (generated by the parent process) and the grandchild process (generated by the child process). These processes compose a process ring and monitor each other in two levels. The first level of monitoring is to ensure that the process of the ring is not damaged, which means that these three processes will not be killed. This can be achieved by monitoring the pipeline. The father process and child process listen anonymous pipes between them, so do the child process and the grandchild process. The father process monitors the other processes by listening to the named pipe. Once a process is killed, the other end of the pipe will be aware of, and the related process will enter its protection module to end the parent process. The second level of monitoring helps ensure that the process ring not be debugged by monitoring the status of the other two processes, using two threads created by each process, the parent process will be ended in case that the process finds its child or grandchild process' status changed to suspended, which indicates that the process is debugged by the other process.

Our solution also monitors the communication of Java Debug Wire Protocol (JDWP) to avoid the application from dynamic debugging on the java layer. The strategy is to find out all Java debug tools using JDWP in their socket communications and to judge what kind of debug tool is connected to. If the debug tool belongs to Andbug, which is usually used by a debugger to reverse-engineer applications for the DVM on Android platform, then the process will be killed at once.

3) Preventing decompilation

The main idea for preventing decompilation is encryption. The .dex files are encrypted with symmetric cryptographic algorithm. The key used in the algorithm is hidden among the codes of the application. In addition, we implement the White Box Cryptography of AES and SMS4, which is used to encrypt the symmetric cryptographic keys so that the encryption algorithm can be run in an unsafe environment of Android system.

5 Conclusions

There are great demands for code protection to prevent Android applications from being reversed and tampered, because the source codes of the Android applications can be more easily recovered by the existing reverse engineering methods. In this paper, we propose a security enhancement system with online authentication for Android APK called SeSoa, in which multiple security mechanisms such as encryption, mutual authentication and monitoring process ring are used to protect Andriodapplications from tamper, dynamic debugging and decompile. To achieve the good balance between security and usability, the mutual authentication is only proceeded when an enhanced Android application is installed and run at the first time.

References

[1] Android Open Source Project. (2015, Oct.). *Android security overview* [online]. Available: https://source. android.com/security/index.html
[2] AppBrain. (2015, Nov.). *Number of android applications* [online]. Available: http://www.appbrain.com/stats/number- of-android-apps
[3] V. Oorschot, and C. Paul, "Revisiting software protection," in *Information Security*. Germany: Springer Berlin Heidelberg, 2003, pp. 1–13.
[4] Sourceforge. (2015, Oct.). *ProGuard* [online]. Available: http://proguard.sourceforge.net
[5] C. Coakley, J. Freeman, and R. Dick. (2005, Feb. 4). *Next-generation protection against reverse engineering* [Online]. Available: http://www.anacapasciences.com/

publications/protecting_software2005.02.09.pdf
[6] C. Kruegel, W. Robertson, F. Valeur, and G. Vigna, "Static disassembly of obfuscated binaries," in *USENIX security Symposium*, San Diego, USA, 2004, pp. 18–18.
[7] DexGuard. (2015, Oct.). *DexGuard, premium security software for android applications* [online]. Available: http://www.saikoa.com/dexguard
[8] Allotori. (2015, Oct.). *Allatori java obfuscator* [online]. Available: http://www.allatori.com
[9] Y. Piao, J. H. Jung, and J. H. Yi, "Server-based code obfuscation scheme for APK tamper detection," *Security and Communication Networks*, vol. 9, no. 6, pp. 457–467, 2014. doi: 10.1002/sec.936.
[10] APKTool. (2015, Oct.). *A tool for reverse engineering android apk files* [online]. Available: http://ibotpeaches.github.io/Apktool
[11] P. Schulz. (2012, Jun. 7). *Code protection in android* [Online]. Available: http://net.cs.uni-bonn.de/fileadmin/user_upload/plohmann/2012-Schulz-Code_Protection_in_Android.pdf

Manuscript received: 2016-01-10

Biographies

DONG Zhenjiang (dong.zhenjiang@zte.com.cn) received his MS degree from Harbin Institute of Technology, China. He is the leader of the Business Expert Team of Expert Committee for Strategy and Technology of ZTE Corporation and the deputy president of Cloud Computing and IT Research Institute of ZTE Corporation. His research interests include cloud computing, big data, new media, and mobile internet. He has led more than ten funded programs and published a monograph and more than ten academic papers.

WANG Wei (wang.wei8@zte.com.cn) received her BS degree from Nanjing University of Aeronautics and Astronautics, China. She is an engineer and project manager in the field of mobile internet at Cloud Computing and IT Research Institute of ZTE Corporation. Her research interests include new mobile internet services and applications, PaaS, and terminal application development. She has authored five academic papers.

LI Hui (lihuill@bupt.edu.cn) received her PhD in cryptography in 2005 from Beijing University of Posts and Telecommunications (BUPT), China. From July 2005, she has been working for School of Computer Science at BUPT as a lecturer and associate professor. Her research interests are cryptography and its applications, information security and wireless communication security.

ZHANG Yateng (526551337@qq.com) is a graduate student in School of Computer Science at BUPT. His research interests include smart phone security, application of cryptographic algorithms, and implementation of white-box encryption algorithm on mobile platform.

ZHANG Hongrui (zhanghongrui@bupt.edu.cn) is a graduate student in School of Computer Science at BUPT. He is conducting research on information security and software protection.

ZHAO Hanyu (hyzhao1990@163.com) is a graduate student in School of Computer Science at BUPT. He is conducting research on software protection in smartphone.

Call for Papers

ZTE Communications Special Issue on

Channel Measurement and Modeling for Heterogeneous 5G

While cellular networks have continuously evolved in recent years, the industry has clearly seen unprecedented challenges to meet the exponentially growing expectations in the near future. The 5G system is facing grand challenges such as the ever-increasing traffic volumes and remarkably diversified services connecting humans and machines alike. As a result, the future network has to deliver massively increased capacity, greater flexibility, incorporated computing capability, support of significantly extended battery lifetime, and accommodation of varying payloads with fast setup and low latency, etc. In particular, as 5G requires more spectrum resource, higher frequency bands are desirable. Nowadays, millimeter wave has been widely accepted as one of the main communication bands for 5G. As a result, envisioned 5G research and development are inclined to be heterogeneous, with possibly ultra dense network layouts due to their capability to support high speed connections, flexibility of resource management, and integration of distinct access technologies.

Towards the heterogeneous 5G, the first and foremost hurdle lies in the channel measurement and modeling in the broad and diversified 5G scenarios. This special issue is dedicated to providing a platform to share and present the latest views and developments on 5G channel measurement and modeling issues.

Schedule

Submission Deadline: November 1, 2016
Final Decision Due: December 1, 2016
Final Manuscript Due: December 15, 2016
Publication Date: February 25, 2017

Guest Editors

Prof. Shuguang Cui, Texas A&M University, USA. Email: cui@tamu.edu

Prof. Xiang Cheng, Peking University, China. Email: xiangcheng@pku.edu.cn

Paper Submission

Please directly send to cui@tamu.edu and xiangcheng@pku.edu.cn, using the email subject "ZTE-CMMH5G-Paper-Submission".