Analyzing Security of Korean USIM-Based PKI Certificate Service

Shinjo Park[1(✉)], Suwan Park[1], Insu Yun[2], Dongkwan Kim[3], and Yongdae Kim[1,3]

[1] Graduate School of Information Security, KAIST, Daejeon, South Korea
peremen@kaist.ac.kr
[2] Department of Computer Science, KAIST, Daejeon, South Korea
[3] Department of Electrical Engineering, KAIST, Daejeon, South Korea

Abstract. This paper analyzes security of Korean USIM-based PKI certificate service. Korean PKI certificate consists of public key and password encrypted private key on disk. Due to insufficient security provided by single password, Korean mobile operators introduced USIM-based PKI system. We found several vulnerabilities inside the system, including private key's RSA prime number leakage during certificate installation. We also suggest possible improvments on designing secure authentication system (Preliminary work of this paper was published previously [1]. This work was responsibly disclosed to the vendor and associated government organizations.).

1 Introduction

PKI certificates are used in several places, including nationwide identification service in several countries. One of the implementation is smartcard-based PKI. PKI certificate on smartcard based national ID card is secure due to card operating system with access control, interfaces to block access of private key and only provides interfaces for signing functionality, but card reader infrastructure costs a lot initially. Korean implementation chose public and encrypted private key on disk, and this caused problem later since it created single point of failure on certificate encryption password and OS security. Most users set certificate password as what they use in other web sites. When malware steals them, attacker can easily decrypt certificate by that password. To mitigate private key leakage program, Korea introduced USIM-based PKI service.

Advancement of mobile technology enabled certifiate storage in USIM card. USIM card is a smartcard conforming to ISO/IEC 7816 with telecom functions. Unlike traditional smartcard, mobile phone already has the reader built-in. Wide penetration of mobile phone enabled USIM-based certificate service in some countries, including Estonia, Finland, and Korea. Estonia started mobile PKI service in 2007 [2], aiming 300,000 users (about 25 percent of population) by 2017 [3]. All Korean mobile operators started commercial USIM-based PKI service in July 2014. They were certified by KISA (Korea Internet and Security Agency) before starting service [4]. Korean PKI service was originally developed

K.-H. Rhee and J.H. Yi (Eds.): WISA 2014, LNCS 8909, pp. 95–106, 2015.
DOI: 10.1007/978-3-319-15087-1_8

for authentication in online banking, but it is now widely used in about 800 sites for the purpose of financial transactions, government services, foreign trading, etc.

In this paper, we analyze the security of commercial implementation of Korean USIM-based certificate service, and find design and implementation flaws. By intercepting PKI certificate installation, attacker can eavesdrop RSA private key in both PC and mobile phone. Other implementation flaws such as not validating SSL certificate, inappropriate debugging message, ineffective code obfuscation makes current implementation more vulnerable.

Section 2 introduces existing work related to USIM-based PKI service in Estonia. In Sect. 3, we present preliminaries of USIM-based PKI system including system overview and possible attack models. Our security analysis is presented in Sect. 4. Section 5 details possible attack scenarios. Discussion and conclusion is in Sects. 6 and 7.

2 Related Work

Formal security analysis for Estonian Mobile ID has been presented by Peeter Laud et al. [5] Prerequisites of Estonian Mobile ID are mobile ID enabled USIM card and activation of the card using smartcard based Estonian ID card. Mobile ID could be used for both identification and signing, with two separate PIN numbers for each purpose. During identification and digital sign process, message set by institution and control code generated by combination of the nonce values of institution and certifiate management authority are shown in both user and USIM application. It provides verification of current action and visual indicator of channel security. Messages between network operator and USIM are based on SMS, and encrypted using a symmetric key. They used protocol analyzer ProVerif, with formal language based on π-calculus. The paper also presents scenarios when each components are controlled by adversary, and provides possible protocol modification on each scenario. In general, despite some weaknesses, Estonian Mobile ID is equal or more secure than smartcard based authentication.

3 Preliminaries

Unlike typical PKI implementation where private key is stored in secure storage, Korean nationwide PKI service chose certificate storage based on files in disk. Certificate of multiple users $(U_A, U_B, \cdots)$ could be stored in one storage. Certificate of U_i contains public key PK_{U_i} and encrypted private key $E_P(SK_{U_i})$ with password P. By default, PK_{U_i} and $E_P(SK_{U_i})$ are stored in fixed location of U_i's hard disk. Also, P might be shared among other online services of U_i. Therefore, if attacker can steal $E_P(SK_{U_i})$ and P, then she can impersonate U_i.

To protect key pair and password from external attacker, Korean PKI implementation (typically as web browser plugin) includes endpoint protection software (anti-keylogger, firewall, etc.) by default. Additional mitigation suggested

by Korean government is to store PK_{U_i} and $E_P(SK_{U_i})$ in removable storage, but it does not increase security. Because of that, Korean government decides to move the storage to USIM card, which provides hardware-based access control by design.

According to Korean patents related to USIM-based PKI service, secure channel exists between smartphone and certificate management server, and communication between smartphone and USIM card is done in plaintext [6]. To access USIM secure storage during key pair installation and cryptographic operation, its password P_{USIM} is used instead of P. Thus, it implies that decryption of $E_P(SK_{U_i})$ is done either in PC or smartphone application. On the other hand, additional vulnerability is introduced by certificate management application.

3.1 System Description

Workflow of Korean USIM-based PKI system consists of installation of existing certificate, and certificate usage. Available installation methods are copying existing certificate, and direct key pair generation on USIM card. Although directly generating key pair on USIM card is the most secure method, it is not feasible for daily usage because current USIM-based PKI is not available on mobile applications, and using separate key pairs for different application is not possible. Therefore, most users will retain their PK_{U_i} and $E_P(SK_{U_i})$ on disk, along with PK_{U_i} and SK_{U_i} installed in USIM card. Figure 1 shows communication process during certificate installation and usage.

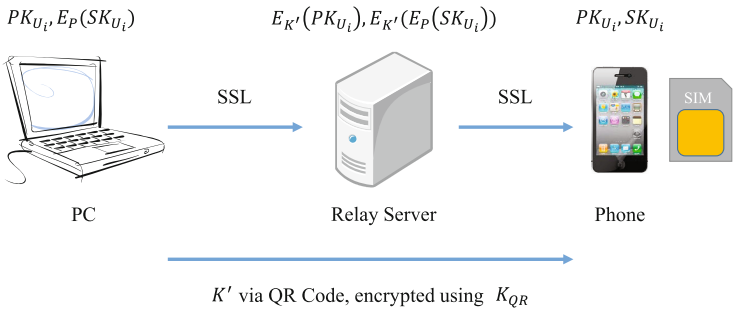


Fig. 1. Key pair installation of Korean USIM-based PKI

On PC side during key pair installation, PC application checks whether $E_P(SK_{U_i})$ belongs to U_i by decrypting it inside the PC. If the key pair is successfully decrypted, additionally encrypted $E_{K'}(PK_{U_i})$ and $E_{K'}(E(SK_{U_i}))$ are sent to relay server via SSL. QR code is generated simultaneously, containing K', address of relay server and session information, encrypted using SEED-CBC with K_{QR}. They are valid for 3 min. Smartphone captures QR code, downloads, decrypts and installs the key pair stored in relay server.

To use certificate stored in USIM card, user select Secure Token in certificate location dialog. Another window asks user's mobile phone number, to send push

message and execute certificate management application. The PC application will display list of public keys in USIM card. After selecting certificate, P_{USIM} is typed on PC. Relay server sends encrypted P_{USIM} using SEED-CBC with K_{PIN}, along with cleartext to be signed to mobile phone. If P_{USIM} entered from PC is correct, then the text is signed by SK_{U_i} in USIM card, and sent back to the user. Before performing cryptographic operations, a confirmation dialog is displayed on smartphone by mobile application.

3.2 Threat Model

Memory Hacking on PC. Operating systems provide APIs for accessing other processes' memory region and control execution to use in debuggers. Malware writers, on the other hand, can use the same APIs to access sensitive data of other application. This technique is called memory hacking. To use memory hacking to steal private information, malware needs to be installed inside victim's computer. Numerous bugs of application and user's unawareness of possible malware makes it easy to install the malware.

SSL MitM Attack. Content manipulating proxy like Paros [7] can hijack HTTPS communication. Paros presents faked SSL certificate to the client, which may use different domain name and/or signing authority than the real one. Without prior register of faked CA root certificate, web browsers will display certificate warning by default and refuse to continue operation unless explicitly told to do so. Applications may use certificate pinning to explicitly allow pre-defined certificate, and reject all other certificates. On Android application, certificate validity are checked by default, unless application author explicitly disabled the check.

For navigating bank web pages, security of HTTPS session depends on that of web browser. Modern web browsers have EV certificate (Extended Validation Certificate) feature to present visual indicator of certificate validity. This is still vulnerable when user initially enters HTTP URL first, then redirected to HTTPS. In most cases, web contents are the same even using different protocol, SSLstrip [8] uses that property to change all HTTPS requests to HTTP when user initially used HTTP.

To perform SSL MitM attack, we use rogue AP (Access Point) for Wi-Fi or ARP spoofing and activate web proxy behind the scenes. Users do not know what is happened behind, or are hidden from what is actually going on.

Effect of Android Rooting. Numerous vulnerabilities in mobile operating system allows user or malware to gain root permissions, which is typically blocked in general. With rooted phones, Android security measures are not effective since root user can see everything inside sandbox, deactivate operating system functionalities and mobile vaccine. Application handling sensitive information can check whether the phone is rooted or not. Common methods are checking existence of `su` binary and root permission management application, vendor-specific warranty bits indicating rooting. Some methods could be bypassed, allowing rooted phone to execute particular application.

3rd-party Malware. Malware can steal private information by collecting user information using Android API, monitoring system logs to check existence of private information, and accessing outside of sandbox on rooted phones. If an application presents its private information using insecure way like printing it inside Android system log, malware can steal that information. Developers can access Android logging facility by `android.util.log` series of API [9], and read the logs using logcat [10] utility or GUI tools from PC. According to Google, it is adviced to remove all logging API calls before releasing the application. Android logging API uses system-wide log buffer to collect all application logs, and logcat application shows combined logs of all running applications. Since logcat is terminal-based application, there are some GUI log management applications on Play Store. Although Google blocked system wide log access for apps since Android 4.1, collecting logs via ADB on PC or rooted device is still possible.

Repackaging. Android applications are written in both Java and native codes. Android Java code is compiled to DEX (Dalvik EXecutable) which could be converted back to Java code using dex2jar [11] and Java decompiler, or smali [12] code (Dalvik assembly) using smali/baksmali combination. Repackaging tool like apktool [13] assists extracting and modifying smali code, creating repackaged APK of application. Repackaged application is visually the same as original one. They are distributed through various channels, and when victim executed particular application, malware injected by attacker is executed.

4 System Analysis

4.1 PC Application

It is easy to write memory hacking application since address of private data is always the same, due to absence of ASLR (Address Space Layout Randomization). By using memory hacking, attacker can obtain user's information including P, PK_{U_i}, $E_P(SK_{U_i})$ and SK_{U_i}. The K_{QR} and IV_{QR} to encrypt K', server address and session information is hard-coded inside application. If attacker knows K_{QR}, she can hijack the session in behalf of victim. If SSL MitM proxy changes relay server's certificate, the application refuses to generate QR code, probably due to usage of certificate pinning or strict certificate check.

4.2 Mobile Application

The application is obfuscated in custom method involving transformation of Dalvik bytecode. Figure 2 shows the method to decode obfuscated application during runtime. Unencrypted bootstrap executes native library to decode obfuscated part before executing the real application. By recollecting memory area using memory map on `/proc/PID/maps`, we can reassemble the decrypted ODEX (Optimized DEX) of application. During Android application execution, Dalvik optimizes functions in device framework before storing application's DEX inside Dalvik cache on memory. This optimized ODEX could be converted back to

DEX file using deodex tools, with framework files of the device where ODEX is generated. Decoded DEX file is not obfuscated; original names of class, method, variable names are untouched.

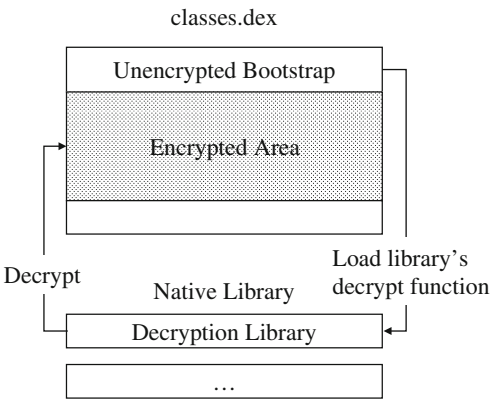


Fig. 2. Obfuscation method of application [1]

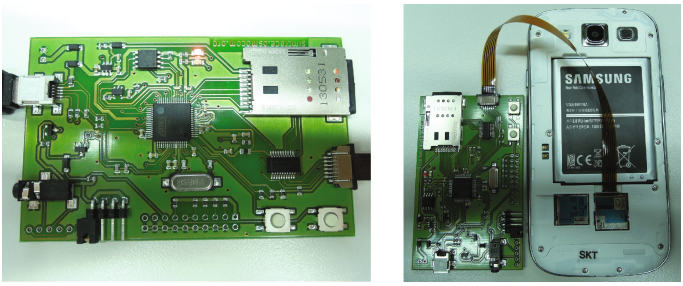
During key pair installation, Android application uses SD card for temporary storage of PK_{U_i} and $E_P(SK_{U_i})$. Using application repackaging to retain temporary files, or create race condition to copy key files from SD card before its removal, it is possible to hijack PK_{U_i} and $E_P(SK_{U_i})$. Vulnerable version of application printed out every USIM APDU (Application Protocol Data Unit) used for communication between phone and USIM, to Android system log. We implement small malware to read the logs and extract information from it.

The application also lacks some security measurements, or degraded platform security deliberately. Application integrity is not checked; attacker can distribute malicious repackaged app to steal private information. While the application refuses to run in rooted device, it could be easily circumvented by renaming `su` binary to other name, and removing root permission management application like SuperSU. The application communicates with relay server via HTTPS channel, whose SSL certificate verification is explicitly turned off. This enables attacker to perform MitM attack on SSL without knowledge to the user. On our experiment with rogue AP, we successfully collect private information inside SSL session.

4.3 USIM Application

To analyze communication between mobile phone and USIM card, we can modify application to print out logs of USIM communication, or use hardware based sniffing device like SIMtrace [14]. There is no unified API for accessing USIM card inside mobile operating system: Apple iOS provides private API (not accessable for general developers), AOSP (Android Open Source Project) and their

derivatives do not include API at all. Example of USIM access API implementation in Android is SEEK for Android [15], which is often included in stock ROM of devices. Device manufacturer and mobile network operator may provide their own private API for accessing USIM card and its applications. SIMtrace allows eavesdropping of phone-USIM channel without knowledge about APIs for accessing USIM, and modification of target application.



(a) SIMtrace hardware (b) Phone connection

Fig. 3. SIMtrace hardware and phone connection

Figure 3a shows SIMtrace device. Original USIM card from mobile phone goes to SIMtrace's USIM card slot (upper right of figure), and mobile phone is connected to SIMtrace using FPCB (Flexible Printed Circuit Board) cable (lower right of figure). By connecting device to computer and using SIMtrace application, we can eavesdrop channel between phone and USIM card. Figure 3b shows SIMtrace connected to Samsung Galaxy S III (SHW-M440S) for demonstration. In this mobile phone, FPCB cable is easy to route since there is no other objects blocking the cable. Different FPCB cable placing is required when USIM slot is underneath battery, or USIM card is connected to phone via tray.

With SIMtrace connected to the phone via FPCB cable and PC via USB, executing `simtrace` application and powering up the phone gives traces of USIM communication in GSMTAP format via local UDP socket. When an application sends command to USIM card, SIMtrace will send the packet containing APDU of particular application. Packets from SIMtrace and USIM communication logs of application were the same, making it easier to trace phone-USIM communication without modification on application or phone firmware.

5 Attack Evaluation

Following attacks were possible in each components:

– PC application: Memory hacking, SSL Hijacking (partially), Circumventing anti-keylogger

- Mobile application: Circumventing rooting check, Log sniffing, SSL Hijacking, No integrity check on message
- USIM application: Unprotected phone-USIM channel.

We implement custom C&C server to collect and display PK_{U_i} and SK_{U_i}, public and private exponents, phone number, P_{USIM}, P of victim. It also provides PK_{U_i} and SK_{U_i} download function, allowing attacker to directly impersonate victim without further processing.

On PC application, we implement memory hacking malware to steal and upload PK_{U_i}, $E_P(SK_{U_i})$, SK_{U_i} and P during key pair installation. Although Paros worked on bank web site, Internet Explorer 11 on Windows 7 displayed certificate warning before actually navigating the site. SSL MitM attack revealed $E_{K_{PIN}}(P_{USIM})$ and cleartext to be signed. Hardcoded K_{PIN} and IV_{PIN} inside application allows attacker to decode P_{USIM}. Cleartext to be signed could be changed in theory, but on our tested bank web site only hash of original message was visible. Anti-keylogger software is ineffective while malware is running with USB keyboard.

On Mobile application, we implement custom Android log stealing malware to steal PK_{U_i}, SK_{U_i} (during key pair installation) and P_{USIM} (all certificate operation). Some phones required workaround to avoid background application termination problem, by using a dummy thread to make application as active. If attacker wants to install key pair on her USIM card, only PK_{U_i} and SK_{U_i} are required. To use key pair on PC, attacker can re-encrypt SK_{U_i} using unrelated P', because Korean PKI implementation on PC expects encrypted SK_{U_i}.

There is no way to find out whether cleartext to be signed comes from legitimate source. Also, person with headset displayed before cryptographic operation do not clearly represent secure user operation. As Alma Whitten et al. suggested [16], the image could be replaced to represent cryptographic operation instead. By hijacking SSL session it is also possible to steal private information in mobile application.

Using SIMtrace, we found that communication channel between USIM and phone is not encrypted and showed the same messages as system log during accessing USIM card. An advanced attacker can create fake signing application from scratch by analyzing how messages are processed.

6 Discussion

During our research, several new implementations of USIM based certificate in Korea emerged. Unlike Estonia where one unified solution is used among all mobile operators, multiple vendors implemented their own solution to mobile network operators. Android application is different from what we have analyzed in previous section, but USIM-phone communication method uses the same telco-specific API and similar USIM APDU structure. If USIM-phone channel is not properly secured, fundamental problem of information leakage will not be solved. Although all services were certified by KISA, the certification is limited to USIM

itself, not including applications for service. Current certification process is broken since any vulnerable component in application chain makes whole process vulnerable.

Since fixing problems in higher level like Android and PC application is trivial, and most problem could be remedied by following secure coding guidelines, we more focus on lower level design to survive problems in higher level, for example, rooting of the device and vulnerable applications.

6.1 Mitigation

One mitigation to prevent memory hacking on PC is ASLR (Address Space Layout Randomization), where operating system changes memory layout of binaries for each execution. From attacker's point of view, location of sensitive changes between each execution. Another mitication is anti-debugging techniques like executable packing (e.g. Themida [17]), self-modifying code, code obfuscation. This can slow down the application analysis, but not completely prevent it.

To prevent SSLStrip like attack, HSTS (HTTP Strict Transport Security) [18] uses HTTP header information to tell the browser that it must use HTTPS for next visit. If web browser has HSTS information for particular web site, then it will always use HTTPS even user entered HTTP URL. Attacker still can strip down HSTS header when user visits web site for the first time. To prevent header strip attack, web browser vendors have whitelist of HSTS-enabled sites to force HTTPS.

Application repackaging could be mitigated by integrity check. If integrity check is implemented in Android Java code, attacker still can circumvent it by modifying application to return fake values for integrity checking routine. Implementing sensitive routine like integrity check in native code makes modification and repackaging more difficult. Android obfuscation tools like ProGuard [19] makes application analysis difficult, but not impossible.

Even with these mitigations, running attacker's code on PC or mobile phone is still possible. To solve root cause of problem caused by implementing security operations on PC or mobile platform, dedicated hardware based security is highly recommended.

6.2 Secure User Interface

First of all, sensitive user interface must be implemented using USIM application toolkit, or inside TrustZone container for better protection.

Figure 4 shows how USIM application toolkit is handled inside Android, other mobile platforms have similar structure. Application binary is contained inside USIM card, secured by card operating system. When mobile operating system is booted, USIM card tells whether toolkit menu is available or not. Mobile OS then shows USIM application toolkit "application" to access application inside USIM card. The "application" in mobile OS can not access application binary directly, the only interface is USIM toolkit terminal messages sent via RIL and baseband.

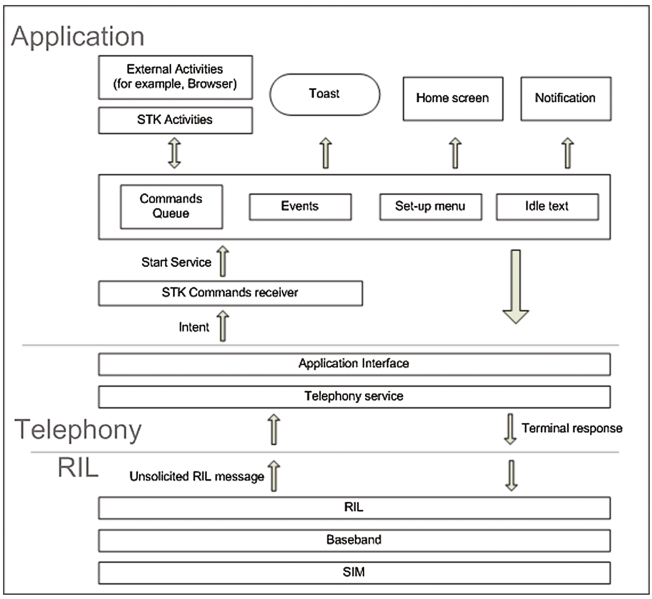


Fig. 4. USIM application toolkit architecture in Android [20]

Android application acts only as wrapper of USIM card's output and input messages. When the application is not tempered, it will only display messages from USIM and pass user's input values to it. USIM toolkit handling application is installed as system application, whose modification requires rooting of the device and may cause unexpected consequences on system operation.

To protect modification of preinstalled system application, we can use TrustZone [21] container to isolate application from other applications. TrustZone-based application environment like Samsung Knox [22] is not easily tempered by simple device rooting. Isolating USIM toolkit application will prevent from application modification to disguise and eavesdrop messages from USIM card. By this scenario we can provide secure display opened to application. Input event must be secured too, making no other party except USIM toolkit application can receive or monitor events to reconstruct user input values.

The application must show confirmation message starting from USIM card. If confirmation dialog is displayed by Android application, malware can press OK on the dialog without knowledge of user. If the dialog came from USIM card, then toolkit message must be passed through USIM card to actually press OK, which requires extra step to do this.

6.3 Secure Communication Channel

If communication method of phone and USIM is revealed to external attacker, it is analogous as sending password using HTTP in web application. By using

faked certificate application, it can collect necessary information to sign the plaintext on user's behalf, and hacker can use stolen identity to cause damages to user. Moreover, if confirmation steps are only implemented in Android side and USIM lacks any kind of verification, identity theft is more easy. Secure channel starting from USIM card can prevent these from happen by preventing USIM APDU leakage.

Estonian Mobile ID secures communication channel by preshared symmetric key based encryption on SMS [5]. Binary SMS is typically not passed to mobile operating system and directly handled inside radio layer and USIM application of the phone. This provides extra protection layer not provided by Korean USIM-based certificate services, and eliminiates framework hijacking problem by not using USIM communication channel on Android application.

7 Conclusion

To implement secure USIM-based authentication service, only securing certificate itself inside USIM card is not sufficient. There are numerous communication channel around smartphone and USIM card, including application layer protocols, Android application, and USIM to phone channel. Our analysis showed that Korean USIM certificate implementation lacks security measures, some of which is fixable, but some are fundamental problems requiring at least reprogramming of USIM card and changes on service architecture.

USIM card can host secured application with user interface, additional security on mobile phone by using TrustZone can further protect those application from external attackers. USIM application can survive application modification, since card operating system prevents direct access of application binary. We strongly recommend implementing secure user interface inside USIM, instead of Android application. We also recommend extending certification scope of Korean USIM-based certificate to the whole system including user interfaces and communication channels.

References

1. Park, S., Park, S., Yun, I., Kim, D., Kim, Y.: Security analysis of USIM-based certificate service in Korea. In: Conference on Information Security and Cryptography (2014)
2. ASi Sertifitseerimiskeskus, About SK - History. https://www.sk.ee/en/about/history/
3. Vaata Maailma, NutiKaitse 2017. http://www.vaatamaailma.ee/en/nutikaitse
4. KISA, Operational Programs (in Korean). http://www.rootca.or.kr/kor/hsm/hsm.jsp
5. Laud, P., Roos, M.: Formal analysis of the estonian mobile-ID protocol. In: Jøsang, A., Maseng, T., Knapskog, S.J. (eds.) NordSec 2009. LNCS, vol. 5838, pp. 271–286. Springer, Heidelberg (2009)
6. Raonsecure Inc., Digital Signature System Using Mobile Device (in Korean), Patent KR 10–2013-0 065 149, 30 December 2013

7. Paros. http://sourceforge.net/projects/paros/
8. Marlinspike, M.: SSLstrip. http://www.thoughtcrime.org/software/sslstrip/
9. Android Open Source Project, Android Developers: Log. http://developer.android.com/reference/android/util/Log.html
10. Android Open Source Project, Android Developers: logcat. http://developer.android.com/tools/help/logcat.html
11. dex2jar. https://code.google.com/p/dex2jar/
12. smali/baksmali. https://code.google.com/p/smali/
13. apktool. https://code.google.com/p/android-apktool/
14. OsmocomBB Project, SIMtrace. http://bb.osmocom.org/trac/wiki/SIMtrace
15. Secure Element Evaluation Kit for the Android platform. https://code.google.com/p/seek-for-android/
16. Whitten, A., Tygar, J.D.: Why Johnny can't encrypt: a usability evaluation of PGP 5.0. In: Proceedings of the 8th USENIX Security Symposium, vol. 99, p. 16. McGraw-Hill (1999)
17. Themida. http://www.oreans.com/themida.php
18. Hodges, J., Jackson, C., Barth, A.: HTTP Strict Transport Security (HSTS), RFC 6797 (Proposed Standard), Internet Engineering Task Force, November 2012. http://www.ietf.org/rfc/rfc6797.txt
19. Android Open Source Project, Android Developers: ProGuard. http://developer.android.com/tools/help/proguard.html
20. Android Open Source Project, Android Open Source: SIM Toolkit Application. http://www.kandroid.org/online-pdk/guide/stk.html
21. ARM Inc., TrustZone. http://www.arm.com/products/processors/technologies/trustzone/index.php
22. Samsung, Samsung KNOX. http://www.samsung.com/global/business/mobile/platform/mobile-platform/knox/