SAVED: Secure Android Value addED services

Antonio Grillo, Alessandro Lentini, Vittorio Ottaviani,
Giuseppe F. Italiano, and Fabrizio Battisti

Department of Computer Science, Systems and Production
University of Rome "Tor Vergata",
Via del Politecnico 1, 00133 Rome, Italy
{grillo,lentini,ottaviani,italiano}@disp.uniroma2.com

Abstract. The availability of free Software Development Kits for recent mobile device platforms challenges many developers in realizing applications for the growing Smartphone market. In many cases such applications may interoperate in their working environment using mechanisms similar to the inter-process communication (IPC) and made available by the mobile operating system. Unfortunately, mobile devices lack in flexible solutions for making these communications secure. In this paper we propose a framework to secure the message exchange with the services installed on Google Android mobile devices. VASs realized by different providers are discovered, used and composed by an Application Frame designed for realizing complex goals. We implemented a prototype of our proposed framework on a real device and we performed extensive testing to measure the overhead introduced by the cryptographic operations required to protect the inter process communication.

Keywords: IPC security, Value Added Services, digital certificate, service interoperability.

1 Introduction

Today's smartphones are widespread mobile devices that combine advanced features in managing personal, phone and business data. One of the most interesting features of smartphones lies perhaps in the possibility to install third party applications; as a consequence, one can use his/her own smartphone truly as a PC: accessing social networks, paying bills, checking bank accounts, etc… Smartphone applications are commonly installed and stored in memory, and in modern devices all the application's data are kept safe from the OS by using a sandbox approach. Such approach prevents other applications to access unauthorized data insulating each application from the others [4], [5], [6]. However, users and manufacturers may suffer from mobile malware infections, call-ID spoofing attacks, spam, and problems with third-party applications or accidents that can cause malfunction of network capacity, disclosure of sensitive business data and more in general privacy problems such as loss of personal data.

M. Griss and G. Yang (Eds.): MOBICASE 2010, LNICST 76, pp. 415–427, 2012.

In this paper we propose a new framework based on Google Android Operating System (OS) for the realization of several value added services (VAS). We call this framework SAVED (Secure Android Value addED services). SAVED enables secure communication between services and applications using such services via Inter Process Communication (IPC)/Remote Procedure Call (RPC). Each VAS is realized through an Android Service. The access to such a service requires the execution of an authentication and authorization phase among the involved parties. Once this initial phase is completed, the application sets up a secure communication with the service using a symmetric encryption scheme.

2 State of the Art

Android is a multi-process system, in which each application (and parts of the system) runs in its own process. Most security between applications and the system is enforced at the process level through standard Linux facilities, such as user and group IDs that are assigned to applications [1]. The Android system requires that all installed applications be digitally signed with a certificate whose private key is held by the application's developer. The Android system uses the certificate as a means of identifying the author of an application and establishing trust relationships between applications. The Android approach grants security of application's data, and prevents access to all services developed by others. Every service publishes in its personal manifest file the permissions required to use the service. One of the permission settings in the manifest file is *Protection level*. The *Protection level* field configures the security policies required by the service; if the level is set to *signature* the service will communicate only with these applications with which it shares the same developer certificate.

The main advantage of the approach followed in the Android design is that developers have to focus their attention only on the application, while the OS grants that all the applications that are not allowed to access the services are prevented from doing so. This simplification comes at an extra cost: only developers sharing certificates and private keys can use services already developed in new applications. This is a huge limitation compared to the growing size of the mall market and the number of organizations and developers enrolled in publishing applications and services. According to a very recent survey on the smartphone OS market published by Gartner [2], both Android and Apple were the only two OSs vendors among the top five to increase their market share: in particular, Android moved to the fourth position displacing Microsoft Windows Mobile for the first time and the Android market has grown 4.4 times in size, going from 10,000 to 20,000 applications in the first four months of the 2010.

The approach of Android prevents third parties to start using the framework's VAS. Developers can use each others' services sharing certificates and credentials: in this case, the applications can interact but the security of the whole framework is granted from a single digital signature; if the developer's digital signature is stolen a

hacker could sign his/her own applications, thus getting complete access to all data of the framework.

Our approach wants to promote the framework scalability and grant secure access to services developed by other users without the need to share private data. We propose to insert a new layer that handles security of inter-process communications; in such layer, trustability is granted directly by the security policy of the framework, and each application can require access and publish services interacting with the framework like in a PKI environment. Thanks to SAVED framework it is possible to face different kinds of threats:

- *Service Spoofing:* the application refers to a service by simply using an interface that establishes the name, the package and the methods signatures; if the original service is replaced on the mobile device, applications that exploit that service are unaware of the substitution.
- *Memory Dump:* starting from Android 1.5, a new API has been introduced to generate a memory dump programmatically. The static method *dumpHprofData(String fileName)* of the *Debug* class generate a dump file that can be converted with the *hprof-conv* tool of the Android SDK and, subsequently, analyzed with different memory analysis tools (e.g., Eclipse MAT, JProfile, etc.). If a fake application execute the dump periodically and export the dump data using a connection (e.g., HTTP connection), it is possible to steal the data exchanged among applications and services.

3 The Framework

SAVED (Secure Android Value addED services) is a framework that grants secure communication between services without requiring private data sharing. Our intent is to improve interoperability between applications and services facing the limits of the Android's native approach. The purpose of SAVED is to allow applications to use services developed by others, to add new VAS to the framework or even to create new applications using already existing VAS. All the interactions performed using the proposed frameworks will be performed in a secure way. SAVED adds supplementary security at the process communication level: each application is accredited to the framework which grants privileges to access in a secure way shared services and facilities. Single process security provided using sandboxes with the Android approach is also preserved in SAVED. In our framework we defined two main entities:

- ***Application*****,** which provides graphical user interface, and all the logics implementing the task to be realized. Applications are implemented extending the *Android.Activity* class.
- ***Value Added Service (VAS),*** which provides to the applications developed using the framework all the certified services. VASs are implemented as remote services extending the *Android.Service* class. The ProxyCA and the ProxyTSA are two special VAS in the framework; these VASs allow the communication with a Certification Authority and a Timestamping Authority, respectively.

In order to realize Applications participating to the framework, developers have to extend specific interfaces and include particular resource packages. When a new VAS is realized, it is required to export its class package. Such class packages will be imported from the Applications that will use the services provided by the VAS. The packages imported will be used to perform inter-process communication. Including such packages and extending the interfaces will provide the supplementary security layer that will grant a secure communication between entities and prevent the access to the services to those applications that are not allowed.

Moreover, we tried to address some best practices to create components participating to the framework enforcing the required security needs. Some examples follow:

- *Activation code:* when the Application/VAS is installed on the device an unlock code should be required to the user; the Application/VAS will remain locked (preventing all interactions) until the user will insert the proper activation code of every entity;
- *Use of standard certificate*: each component should have a proper X509 digital certificate signed from a valid Certification Authority (CA), such certificate will be saved in a keystore inside the component memory area; the component will be responsible to take care of managing correctly the keystore itself to grant a secure saving of the other's certificates;
- *Model View Control Pattern*: VAS and Applications will take care of implementing independently graphical user interfaces to be shown to the end user;
- *Mutual Authentication:* each entity needs to implement a mechanism to grant mutual authentication. The mutual authentication should be ensured by mutually exchanging and verifying the digital certificates. Using a handshake schema (e.g., TLS handshake) the involved entities exchange their digital certificates, check the certificates validity through the ProxyCA, and mutually authenticate themselves (see Fig. 1).

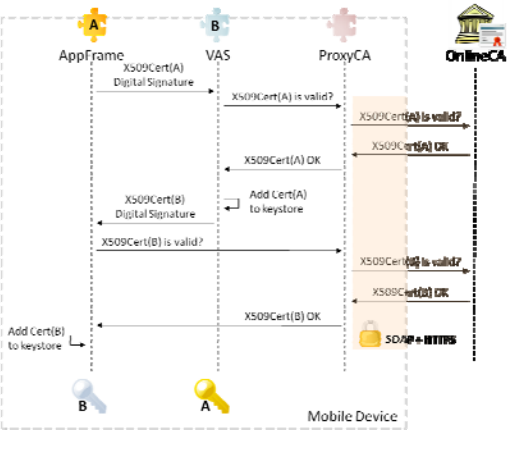

Fig. 1. Mutual Authentication phase

- *Session Authentication:* once the entities are mutually authenticated, a session key (i.e., SK) is shared. According to our approach the SK is generated by both the Application and the VAS using parameter defined by the two parties (i.e., CTRL_A and CTRL_B). Adopting a key agreement protocol (e.g., Diffie-Hellman protocol) the involved entities agree on secret SK that will be used to encrypt subsequent communications (see Fig. 2).

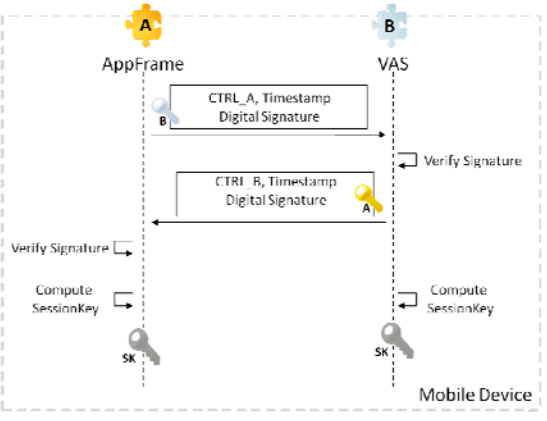

Fig. 2. Session Authentication phase

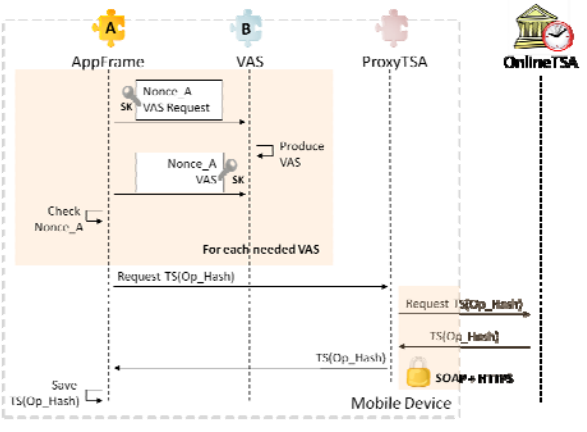

Fig. 3. Session Encryption phase

- *Session Encryption:* Every VAS allows access to its functionalities only to "trusted" Applications; trusted Applications have performed successfully the Mutually Authentication and the Session Authentication phases. In order to enforce the uniqueness of each interaction with VAS a random value (i.e. Nonce_A) is used; the confidentiality is granted by encrypting the exchanged data with the SK.

  The Application composes the results of different VAS in order to realize a complex goal. At the end of this phase, the Application interacts with a

timestamping authority through the ProxyTSA in order to securely keep track of the creation time of the realized goal (see Fig. 3). The sensitive data of the operation are summarized applying an Hash function (i.e., Op_Hash) and these data are sent to the Timestamping service.

Mutual Authentication, Session Authentication and Session Encryption represent the secure core of SAVED framework and should be carefully performed in order to join the framework.

4 The Framework Implementation

We developed a prototype of the SAVED framework on an Android 1.5 platform. The main features of the proposed framework are encapsulated into the jar files that contains two kind of files (i.e., .aidl, .Stub) for the inter process communication.

AIDL (Android Interface Definition Language) is an IDL [3], [4] with which it is possible to generate automatically the source code that allows two Android applications to exchange information using IPC. AIDL/IPC interface based mechanism is similar to Common Object Model (COM) or Common Object Request Broker Architecture (CORBA). In order to implement an AIDL/IPC service it is required to perform some steps:

- Create an .aidl file to define the interface (*YourInterface*.aidl). The interface defines the access methods and the fields available to a client.
- Add the .aidl file to the makefile and implement the methods of the interface creating a class that extends the *YourInterface.Stub* (.Stub file is automatically generated by the tool) and implements methods declared in the .aidl description file.
- Publish the interface to clients rewriting the Service.onBind (Intent) method; this method will return an instance of the class implementing the interface.

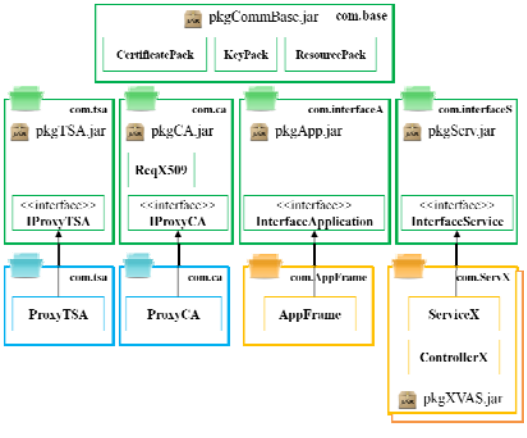

Fig. 4. SAVED framework main packages

This IPC mechanism needs a way to share complex information, such as non-primitive types, between two entities. In order to achieve this goal Android provides Parcelable class able to serialize and deserialize complex types.

Fig. 4 simplifies the package diagram of SAVED. The picture shows on top the following core .jar files:

- ***pkgApp.jar*** contains the interface *InterfaceApplication* that must be implemented by every class that want to participate SAVED as an Application;
- ***pkgServ.jar*** contains the interface *InterfaceService* that needs to be implemented by every class that want to be a VAS in the framework;
- ***pkgCA.jar*** carries the *IProxyCA.aidl* with his relative .Stub file; these files allow the communications between the entities of SAVED and the ProxyCA. Moreover, the jar file contains the parcelable class ReqX509 that is mandatory for the communication;
- ***pkgTSA.jar*** packages the *IProxyTime.aidl* with his relative .Stub file to grant communication with the Proxy TSA;
- ***pkgCommBase.jar*** contains the three base parcelable files that grant the communication between the Application and the VASs, namely *CertificatePack.java*, *KeyPack.java* and *ResourcePack.java.*

In order to grant to an Application to contact and receive services from all the VAS inside the framework, and so assemble the services offered from the VAS to create complex applications, it is required to install the ProxyTSA and the ProxyCA Android packages (apk); these entities are shown in the lower left half of Fig. 4.

ProxyCA is one of the underlying VASs that exist in the framework. All entities must submit to the ProxyCA the digital certificates they receive from their communication partners. The service contacts a web service that works as an online Certification Authority, inserts the certificate in a XML file and through a secure HTTP connection (i.e., HTTPS) asks for the certificate verification. The web service checks the certificate validity and answers with an XML response.

ProxyTSA is another basic VAS of SAVED. As the ProxyCA the ProxyTSA takes in account the communication with an external partner, the timestamping web service. All the communications between the proxy and the timestamping web service are managed through XML messages on HTTPS.

A Building a Value Added Service

- Create a new Android project with a class that extends the native Service class;
- Import in the project:
- pkgServ.jar,
- pkgCommBase.jar,
- pkgCA.jar;
- The main class of the project must implement *InterfaceService* interface class and consequently all his methods;

- Create the graphical user interface;
- Create the IServiceX.aidl in the project as described previously;
- Create and export pkgXVAS.jar containing IServiceX.aidl and the corresponding .Stub file generated automatically;
- Service class must implement, all the standard methods of the Android native Service class, and the .aidl interface with all the methods defined through the description language;
- Release the service as an .apk file for the installation on the device.

B Building an Application

- Create a new Android project which contains a class that extends the native Activity Android class;
- Import in the project:
- pkgApp.jar,
- pkgCommBase.jar,
- pkgCA.jar,
- pkgTS.jar;
- The main class of the project must implement the *InterfaceApplication* interface with all his methods;
- Create a graphical user interface to allow the user to interact with the Application;
- Import from each VAS you want to use in the Application the corresponding jar file (i.e., pkgXVAS.jar)
- Use each service in a proper way, taking care of managing and releasing correctly the connection with the involved VAS. Note that early versions of Android platform serialize the access to the services.
- Release the Application as an .apk file for the installation on the device.

Assume we are in a scenario where we have one Application and one VAS, each one with its own digital certificate signed by different CAs. Note that in this scenario, none of the entities “knows” the public key or the certificate of the counterpart. If the two entities wish to cooperate, they need to authenticate each other. After contacting the ProxyCA to verify the communication partner trustability (cfr. the Mutual Authentication phase), an asymmetric cryptography session to exchange the session key can be started (cfr. the Session Authentication phase). Finally, the session between the involved parties is encrypted using symmetric cryptography (cfr. the Session Encryption phase). The need to switch from asymmetric to symmetric cryptography is due to the performance overhead of asymmetric cryptography: indeed, the switch from asymmetric to symmetric cryptography improves the performances of the whole framework reducing the effort due to encryption/decryption operations.

5 Use Cases

The framework described in the previous sections, can be used to develop complex Applications or VAS interfacing basic services in a secure, certified and non-repudiable way. In this section, we will detail two use cases.

A Payment VAS

An explicative use case, which requires security and non-repudiability of the operations can be the access to money transfer services and management of the related information.

A payment gateway (e.g., PayPal[tm] [8], or Google checkout [10]) eases the transfer of information between payment portal and frontend processor. A new trend (e.g., PayPal Mobile Checkout [9]) for payment gateway is to provide its service to the growing population of mobile users.

A developer who has implemented an application that require any form of money transfer (e.g., buying e-books, music, games, tickets, … or booking a service) needs to interface his/her application with all available payment gateways. A payment gateway aims to reach as many developers as possible in a simple and secure way, preserving its distinctive user interface. Using SAVED, developers can interface their e-commerce Application with the VASs exposed by different payment gateways (see Fig. 5).

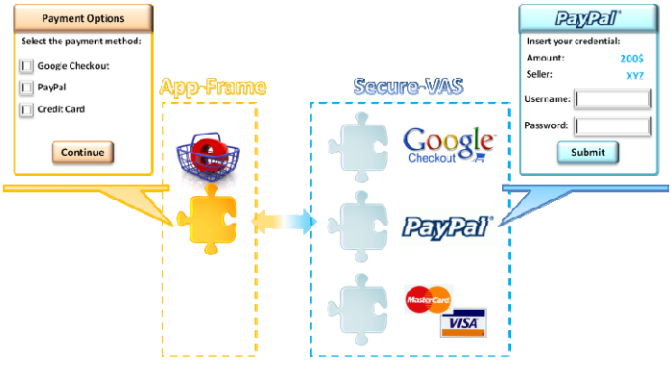

Fig. 5. The Payment VAS Use Case realized in the SAVED framework with three payment opportunities (i.e., Google Checkout, PayPal and Credit Card)

After inserting the payment gateway VAS in our framework, all the communications between SAVED Applications and VAS will be performed in a secure way. The VAS using the framework will transparently provide API ready to use in the Application.

B Event Certification

Producing digital evidence through a report can demonstrate other advantages of using Application and VASs belonging to the SAVED framework. To preserve in time and space a digital document, some accessory information are required to keep in time the authenticity of the digital document and the proof of the evidenced fact. The user probably could produce, as evidence, a document enriched by: a picture of the place where the fact happened or of the fact itself; some georeferenced information; a timestamp; a voice note; and a text note characterizing the digital document.

A modern mobile device, such as a Smartphone, is the most suitable tool to reach the use case's goal as it is equipped with all the hardware capabilities to perform the following services:

- GPS Position; that allows the user to get his/her current location on earth, the two GPS coordinates (latitude and longitude) will be used from the Application to be attached to the digital document; Android provides two classes to interact with the GPS device and access such coordinates:
  - LocationManager which handles the access to the location service of the system. The location service provides periodically updates of the current location of the device on the earth and alerts the user when he/she is in the proximity of some location on earth.
  - LocationListener is used to manage the alerts and the changes of latitude and longitude coming from the LocationManager class. The operating system calls the LocationListener methods automatically.
- Pick a photo; Android provides some classes and methods to access the camera installed in the device, to present a preview of the picture to the user, to get the picture clicking a button, to save the picture in the device filesystem and to handle the camera settings. The classes used to perform all these operations are:
  - SurfaceHolder.Callback, represents the user interface to show the preview of the picture the user is going to take.
  - Camera, which handles all the camera actions such as connect/disconnect, handle the settings and getting data to be converted in a manageable format.
- Getting an audio record; is the service providing to the user the possibility to get an audio record, using the device microphone; the class used to get access to audio/video recorder is the MediaRecorder class.
- Getting a text note; such functionality has not been implemented as a VAS, but like a small extension created using an EditBox.

The Application realized for this use case simplifies in a GUI the interaction with the VASs (see Fig. 6) and generates a data package containing a picture, a GPS location, a text note and an audio message. Such data package will be processed and sent to the TSAProxy that will certify the time in which the digital document has been created as it is connected with a Time Stamping Authority.

The VASs implemented to test the framework can be used in a plethora of different contexts; a lot of applications currently on the Android Market use one of the securized services in a non secure way.

Fig. 6. Screenshot of the Application that realizes the Event Certification use case

6 On a Real Device

The framework has been tested on an Android HTC Magic device. The device was equipped with Android 1.5 OS, 3.2 M-pixel camera, Integrated GPS Antenna, Wi-Fi: IEEE 802.11 b/g. Using Android ADB tool different .apk, created using Eclipse IDE, have been installed on the HTC Magic. The testing phase has highlighted a slower response of the Applications due to security operations, inter-process communications via AIDL interfaces and parcelable classes. We executed some performance tests using our prototype. We aimed at measuring the time computational overhead introduced by the use of SAVED, and thus we measured the time needed to execute security functions. In particular, we have considered the overhead related to each one of the phases described in Section 3.

Table 1. Time overhead for the framework phases

Phase		*Time (ms)*
1.	Mutual Authentication*	1197
1.	Mutual Authentication	446
2.	Session Authentication	257
3.	Session Encryption	795
Total Framework Overhead		***1498***

In Table 1 we can see the time overhead introduced by SAVED. The first row of the table refers to the first execution of the Mutual Authentication phase, while the second row refers to the subsequent executions. In the first case the more time required is justified by the need to update the keystore with the new digital certificates; this delay is paid once. The total framework overhead amounts to 1.5 second preserving the usability for real use cases.

We have chosen to test the framework on a HTC Magic that is one of the earliest models of Android, so the performance issues are more evident; clearly, new devices are more responsive and performance problem will always be less significant.

The testing phase has been really useful to verify the effectiveness of the framework and to solve some side issues due to the complexity of the interaction between processes in a real mobile device: for example, we noted that the device puts the application in stand-by mode when it notices a display rotation, this is because the OS calls the application that rotates the display, and later gives back the control of the system to the running application. This issue has been solved using onSaveInstanceState and onRestoreInstanceState when the application is put in stand-by and woken up from the OS. Implementing such methods prevents data loss due to changes in the state of the application.

7 Conclusion

In this paper we have presented a new framework called SAVED in which applications and services taking part can communicate and share safely functionalities and facilities. We have implemented a prototype of SAVED and we have tested it on a real device. The operational capacities of the framework have been verified.
Using the proposed framework enables complex use cases, with a range of value added services actually certified. Future extensions of the framework include: the enhancement of the discovery mechanism through which each Application receive information about SAVED VAS available on the device; a mechanism to audit the history of each VAS or Application, so as to keep track of all kinds of actions and information exchanged between the entities of the framework.

References

1. Android developers, “Security and Permission” (June 2010), `http://developer.android.com/guide/topics/security/security.html`
2. Gartner Inc., “Gartner Says Worldwide Mobile Phone Sales Grew 17 Per Cent in First Quarter 2010” (June 2010), http://www.gartner.com/it/page.jsp?id=1372013
3. Bachmann, F., et al.: Documenting Software Architecture: Documenting Interfaces. Sofware Enginerring Institute, Carniege Mellon (2002)
4. Lamb, D.A.: Sharing intermediate representations: the interface description language, Ph.D. Dissertation, Carnegie-Mellon University, Department of Computer Science (1983)
5. Gong, L., Mueller, M., Prafullchandra, H., Schemers, R.: Going Beyond the Sandbox: An Overview of the New Security Architecture in the Java Development Kit 1.2. In: Proceedings of the USENIX Symposium on Internet Technologies and Systems, Monterey, California (December 1997)
6. Burns, J.: Mobile application security on Android, Context on Android security, Black Hat (2009)

7. Burns, J.: Developing secure mobile applications for Android, iSEC Partners (October 2008)
8. PayPal, Adaptive Payments Guide (June 2009), PayPalIntegrationCenter https://www.x.com/community/ppx/documentation
9. PayPal Mobile Checkout, PayPal Mobile Checkout Developer Guide (October 2009), PayPalIntegrationCenter https://www.x.com/community/ppx/documentation
10. Google checkout, About Google Checkout, http://checkout.google.com