Stateful Usage Control for Android Mobile Devices*

Aliaksandr Lazouski, Fabio Martinelli, Paolo Mori, and Andrea Saracino

Istituto di Informatica e Telematica, Consiglio Nazionale delle Ricerche
Pisa, Italy
name.surname@iit.cnr.it

Abstract. This paper proposes a framework for regulating data sharing on Android mobile devices. In our approach, the user downloads a copy of the data on his Android device, then the framework controls the data usage by enforcing the usage control policies which have been embedded in the data itself by the data producer. The usage control policy is based on the Usage Control model, whose main feature is to allow the usage of the downloaded data as long as conditions specified in the policy are satisfied. The proposed framework secures the data access procedure relying on both the Android security mechanisms and the introduction of Trusted Platform Module functions. The paper details the proposed framework, presents some preliminary results from the prototype that has been developed, and discusses the security of the prototype.

Keywords: Usage Control, Mobile devices, XACML, Android

1 Introduction

In the last years, the spreading of mobile devices, such as mobile phones and tablets, has dramatically increased so that nowadays the majority of the population owns at least one of them. New generation mobile devices are managed by operating systems which can be customized by installing applications (apps) offering a plethora of functionalities. Smartphones and tablets are equipped with fast multi-core processors, they have a good storage capacity, and a strong connectivity. Mobile devices can connect to the Internet through several interfaces, e.g., operator network (3G/4G), WiFi, or they can communicate directly with other devices through Bluetooth or NFC interfaces.

Thanks to these features, mobile devices are currently an excellent platform to download and access data, whose usage sometimes should be restricted by security policies.

Currently, Android is the leading operating system for mobile devices and is run by almost 80% of smartphones and tablets [1]. Android is a multi level open source platform, which provides an environment for the execution of mobile applications. Mobile applications are distributed to the final users through repositories (markets), that are managed

* This work was supported by the EU FP7 project *Confidential and Compliant Clouds* (CoCoCloud), GA #610853

[1] http://www.gartner.com

by trusted parties (e.g., Google Play Store), where developers upload, distribute and sell applications. Android provides a robust security architecture that has been designed to ensure the protection of the platform, i.e., of data, resources and applications, while reducing the burden on application developers as well as allowing users to control the access rights assigned to applications. In fact, each Android application is executed in a sandbox that prevents it from interacting with other applications, and the access to resources is regulated by a permission system, as detailed in Section 3.1.

However, to regulate the access to downloaded data, the native security features of Android are not sufficient. This paper focuses on controlled data sharing on mobile devices running Android. As a matter of fact, as long as data are stored in the producer domain, they can be protected using traditional access control techniques, but once these data have been downloaded on a mobile device, typically no further controls are performed to regulate their usage. However, many scenarios require that the usage of data is regulated even when they are stored outside the producer's domain.

1.1 Contribution and Motivation

This paper proposes a framework which allows the sharing of data on Android mobile devices while protecting the data by regulating their usage. In particular, the data producer embeds a data usage control policy in the data he shares with another user. The data users exploit our framework to download a copy of the data on their Android devices, and the framework controls their subsequent accesses and the usage of these data by enforcing the usage control policy which is embedded in each data copy. The usage control policy is based on the Usage Control model, defined by Sandhu in [1], and its main feature is that it allows the usage of data as long as a set of conditions are satisfied.

Our approach protects the data from the users of the device and from the applications running on the device. Also, since the device is not always connected to the internet it should be possible to make access decisions locally. Our framework consists of a Data Protection System, that manages the storage and the accesses to the data, and of a policy decision point which is capable to evaluate policies written in UXACML language, an extension of XACML OASIS standard for Usage Control. We implemented the prototype of our stateful data usage control system as a couple of Android applications, called Data Protection System (DPS) app and UXACML Authorization app.

Many real scenarios would benefit of the proposed framework. For example, the data producer could be a team manager who wants to share his business documents with his employees. Since the documents are critical, the manager wants to allow his employees to visualize them on the company's tablets when they are located within the building of the company department, or when they are outside this building and the network interfaces of their devices are switched off. Hence, the manager could use our framework to share documents with his employees, embedding in these documents a policy that allows the visualization of the document according to the previous condition. Other real scenarios like this one can be envisioned in the Bring Your Own Device (BYOD) paradigm, in which employees are allowed to use the same mobile device both for business and private usage. The employers want the employees to separate the two domains, private and business, on their device. Thus, some business functions should

be disabled when the employee is using the device for private matters. With the introduction of smartphone and tablets and the increasing popularity of cloud environments, BYOD topic is of central importance and several security challenges have still to be addressed [2]. The approach described in this paper can be used to prevent the access to data and cloud functionalities, when a business-related security policy is not valid any more.

1.2 Paper Structure

The paper is structured as follows. Section 2 describes some related work. Section 3 describes the standard security support provided by the Android operating system, and the Trusted Platform Module. Section 4 describes the adoption of the usage control model in our scenario along with simple examples of security policies. Section 5 shows the architecture of the proposed framework, describes the prototype implementation along with some performances and security analysis. Finally, Section 6 draws the conclusions.

2 Related Work

The authors of [3] propose a framework to enhance mobile devices security focused on the protection of the resources of the mobile devices from the applications that are executed on these devices. In particular, the framework prevents the misuse of mobile device's resources using a runtime monitor that controls the application behaviour during its execution, and policies are expressed in Conspec [4]. The paper presents a prototype for Symbian OS, running on Nokia E61 and N78 devices, and a prototype for OpenMoko linux running on HTC Universal devices, and evaluates the overhead and the battery consumption introduced by the security support.

The work described in [5] proposes the run-time enforcement of policies to regulate the information flow among applications on Android devices. Policies are expressed through labels, applied to applications or to their components, which specify the secrecy level, integrity level, and declassification and endorsement capabilities of the object it is paired with. The paper describes an implementation on a Nexus S phone running Android 4.0.4, built on the top of Android's activity manager, that intercepts the calls between the components. This framework is different from the one proposed in this paper because it is focused on controlling the information flow among applications.

Another system designed to enhance the security support of Android, CRêPE, is proposed in [6] and [7]. CRêPE is a fine grained context related policy enforcement system, where each policy consists of an access control policy, composed by standard access rules, and an obligation policy, which specifies some actions that must be performed. Each policy is paired with a context, a boolean expression over physical and logical sensor of the mobile device, and when a context expression is evaluated to true the corresponding policy is enforced. The paper describes and evaluates the effectiveness and the performance of a prototype derived from the Android Open Source Project (AOSP).

The work in [8] describes a framework called *Kirin* for security attestation of apps before they are installed on a mobile device. The framework looks for malicious pattern

in the application package and avoids the deployment of dangerous apps. Kirin does not perform analysis of the runtime behavior of the application and does not apply any security policy.

Another work focusing on the analysis of access to security critical resources is *TaintDroid* presented in [9]. This framework is based on hooks put in all the methods that handle access to user or device data. The information flow is then tracked, to understand which applications are effectively able to see the tracked data. This framework requires the operative system to be modified.

The work proposed in [10] describes a security framework which enhances the Android permission system allowing the user to choose the permissions to grant to an application and handling correctly the exceptions for revoked permissions, preventing the apps from crashing. With this framework the user can choose effectively which permissions grant to the app. However, the security policies which can be defined through this approach are static and limited to the coarse-grained permission system.

Access control to privacy sensitive information has been addressed in [11], which propose a framework called *TrustDroid* to define access level to specific device resources and operation. TrustDroid also allows to define some context based policies, but this policies only apply to applications installed on the device, thus private data in the wrong context can still be accessed by the user. Moreover a mechanism of remote attestation is missing and it requires modification of the operative system.

A preliminary work concerning sticky policies for mobile devices is presented in [12] where authors claim that context-aware access and usage control can be a significant support for users in mobility. The paper presents a prototype for Android systems, called ProtectMe, which allows to specify sticky policies including access and usage control directives to be enforced on the data they are attached to.

There some significant differences between the work in [12] and the one presented in this paper. Mainly, it is due to the security policy languages that the two frameworks enforce, respectively, PPL [13] and U-XACML [14]. The PPL language is designed to express obligations, i.e., actions that must be enforced by the PPL engine. Instead, the the U-XACML engine handles continuous authorizations and conditions, i.e., constrains on mutable attributes which should be re-evaluated when attributes change and a resource is still in use. A violation of these authorizations and/or conditions implies the access revocation. Then, the U-XACML assumes a distributed authorization infrastructure which requires collaboration of many parties on sharing and updating security attributes. Thus, it is capable to enforce policies which govern usage of all data copies rather than local only. Finally, the work in [12] does not provide performance analysis.

3 Background

We report in this section background notions on the security mechanism on which the proposed framework relies.

3.1 Android Security Overview

The Android framework includes several elements to enforce security on the physical device, applications and user data. The Android native security mechanisms are the

Permission System and Application Sandboxing, which enforce, respectively, access control and isolation. Through the permission system, every security critical resource (e.g., camera, GPS,Bluetooth, network, etc.), data or operation is protected by mean of a permission. If an application needs to perform a security critical operation or access a security critical resource, the developer must declare this intention in the app `AndroidManifest.xml` (manifest for short) file asking the permission for each needed resource or operation. Permissions declared by the application are shown to users when installing the app, to decide if he wants to consider the application secure or not. If the application tries to perform a critical operation without asking the permission for it, the operation is denied by Android. The manifest file is bound to the application by means of digital signature. The integrity check is performed at deploy time, thus the Android system ensures that if an application has not declared a specific permission, the protected resource or operation cannot be accessed. In the last Android versions, users can dynamically revoke and re-grant specific permissions to applications, however this practice requires a level of knowledge and expertise greater than that of average users. In fact, revoking permissions will often result in app misbehaviors including crashes, due to the unhandled exception caused by the missing permission.

On the other hand, isolation is enforced through the synergy of two elements: the Dalvik Virtual Machine and the underlying Linux kernel. In Android every application runs in a virtual machine called Dalvik Virtual Machine (DVM). The DVM is an optimized version of the Java Virtual Machine, where each application has its own memory space, can act like it is the only application running on the system and is isolated from other apps. Moreover each instance of the DVM is registered as a separate user of the Linux kernel. This means that each installed app is considered a user at the kernel level, able to run its own processes and with its own home folder. The home folder of each application stores application files on the device internal memory, thus it is protected from unauthorized access by the Linux kernel itself. In fact, files stored in the home folder can be accessed only by the application itself. However, since the device internal memory is limited, the amount of data that can be stored in the home folder is limited and generally using the internal memory is a deprecated practice.

3.2 Trusted Platform Module

Integrity of the mobile device architecture can be assured through usage of *Trusted Computing*. Verifying device integrity is mandatory to ensure that other applications cannot interfere with the DPS and UXACML Authorization Apps. The *Trusted Computing Platform* (TCP) is an hardware framework that enforces hardware level security on a specific system. Devices protected through trusted computing embed a hardware module called *Trusted Platform Module* (TPM). The TPM includes keys and routines necessaries to verify the integrity of various levels of the device, from the firmware level to the application one.

The Trusted Computing Group is currently proposing a standard for the application of the TPM to mobile devices, such as smartphone and tablet. The system model presented in [15] and depicted in Figure 1 can be matched with the Android architecture. In fact, the Operating System block can be seen as the underlying Linux kernel

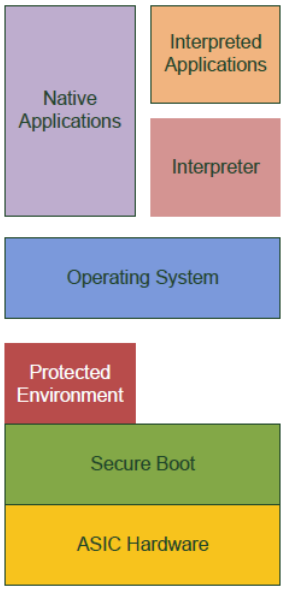

Fig. 1. Secure Mobile Device Architecture Proposed by the TCG

of Android devices, the Native Applications block is embodied by the native application set provided by Google or the device manufacturer on stock devices e.g. `Dialer`, `Hangout`, `GMail` etc.. Furthermore, the Interpreter block is the DVM which executes all the applications on the device (Interpreted Applications). The protected environment is the part protected by the TPM where secure information is stored.

The TPM securely stores inside its registers (PCR) the values to perform the integrity checks of the device components. Integrity check is performed by the TPM with a bottom-up approach, in a process which is defined as *Chain of Trust*, where the integrity of each level is considered the *Root of Trust* for the higher level. This process involves a set of measurements on its configuration, which includes a set of hash computations of the code of its kernel and of the running applications. The root-of-trust is rooted to the physical platform TPM. Hence, to measure the initial integrity of the device, starting with the TPM, the following steps are required. Firstly, the TPM applies a set of measurements on the boot-loader, so that from now on, all the steps can be measured from boot to kernel loading and its modules. Then is verified the integrity of the kernel, operative system and finally of the application installed on the device.

When the integrity of a level cannot be verified, the device is considered non-secure from that level to higher ones. If the device cannot be considered secure, the integrity of the DPS and UXACML Authorization apps cannot be ensured. This check is effective against malware or device rooting attempts. In fact, if the user tries to root the device, i.e. tries to obtain root privileges to avoid security policies, the TPM will detect this modification and, consequently, it will block access to data removing the application and all its data (see Section 5). Moreover the external provider of the app will be no-

tified. The same goes for dangerous applications. Any installed application is matched against a database of authorized application signatures. If a new application is installed then the value stored into the TPM is updated only if the app is in the signature database, otherwise the device will be considered non-secure.

Since the mobile TPM has still to be standardized, currently there are no Android devices supporting it. However, the TCP building blocks that have to be included on the Android platform have been described in [16], and are the following: a Root-of-Trust for Measurement (RTM), a Root-of-Trust for Storage and Reporting (RTS/RTR) and a Static Chain of Trust (SCoT). These elements have been developed as extension of the Linux Kernel, realizing a virtual TPM which implements all the interfaces to perform integrity checks and update stored values. However, being a software component this virtual TPM is not able to ensure any security property.

It is worth noticing that the TPM module is a hardware component embedded onto mobile devices, which will be delivered with an Android version able to manage it. The proposed framework will be available only for those devices equipped with TPM. However, the installation of the proposed framework will not require any modification of the existing version of Android, since it will exploit the existing standard interface to exploit the TPM.

4 Usage Control on Android

The framework proposed in this paper is aimed at enabling Data Producers (DPs) to perform a controlled sharing of their data, i.e., it allows other users, data users (DUs), to download a copy of these data (Data Copy, DC) and it imposes some constraints on the usage of these data. The framework has been designed for allowing the sharing through mobile devices running Android operating system. In fact, the data users run our application on their Android devices to download a local copy of the shared data from our sharing server and to access it. The downloaded data copies are paired with their usage control policies, which have been defined by the DP, and our application enforces these policies when data are accessed and stored on Android devices.

4.1 Usage Control Model

The policies enforced on shared data are based on the Usage Control model (UCON) defined in [1]. The core components of the UCON model are: subjects, objects, actions, attributes, authorizations, conditions, obligations. In the following, we briefly describe how they are instantiated in our scenario. However, for a detailed description of the UCON model please refer to [17,18,19].

The *subjects* are the Data Users (DUs) who perform some actions on the local copies of the data (which are the *objects*) that have been downloaded on their mobile devices. In the previous example, the employees are the subjects and the business document is the object. We assume that DUs are registered to Google, as we use the Google account as subject identity. In the reference scenario of this paper the policy covers a small set of security relevant *actions*:

- $visualize(s, o_{dc})$: the DU (s) visualizes the data copy object o_{dc};

- $append(s, o_{dc}, nd)$: the DU (s) append new data nd in the data copy object o_{dc};
- $replicate\&send(s_{from}, s_{to}, o_{dc})$: the DU ($s_{from}$) sends a copy of its data copy object (o_{dc}) to the (device of the) other user s_{to};
- $delete(s, o_{dc})$: the DU (s) deletes the data copy object o_{dc} from the mobile device.

We recall that the set of actions performed by the application could be larger than the previous set because not all the actions performed by the application (and not all their parameters) are security relevant.

Attributes are used to represent subjects' and objects' features. Attributes are mutable when their values are updated as a consequence of actions performed by subjects. Mutable attributes can be updated before the access (*pre-update*), or during (*on-update*), or after (*post-update*) the usage. For example, the subjects' reputation is an attribute that could be updated with a post-update. An object attribute can refer to: i) a specific Data Copy (DC), e.g., "id" or "creation date", ii) a subset of DCs, e.g., "number of data copies belonging to the user U", iii) all DCs derived from the same data object, e.g., "global number of copies".

Immutable attributes of data, e.g., id, creation date, or producer, are embedded in the data object itself, because their values will not change during the object lifetime. Mutable attributes of data, instead, are not embedded in the data object because a large overhead for keeping consistency among all copies of the same attribute would be required. Consequently, mutable attributes of data and users are stored on proper Attribute Managers that are invoked to retrieve the current value of these attributes when required. Since mutable attribute might be read and updated concurrently, Attribute Managers must be able to manage concurrent accesses avoiding inconsistencies. Mutable attribute management is out of the scope of this paper; an example is given in [20].

Authorizations are predicates that involve attributes of the subjects and of the objects. The policy could require that an authorization predicate is satisfied to begin the access (*pre-authorization*), or continuously while the access is in progress (*on-authorization*). In the latter case, as soon as the authorization predicate is no longer satisfied, the access is terminated. *Conditions* are environmental factors which do not depend upon subjects or objects. The evaluation of conditions can be executed before (*pre-condition*) or during (*on-condition*) the action. *Obligations* verify the fulfilment of some mandatory requirements before performing an action (*pre-obligation*), or while performing it (*on-obligation*).

4.2 Policy Example

Let us consider the the previous example, where a team manager wants to share some critical business documents with his employees. The following policy example is expressed using a more readable language then UXACML language:

```
policy-set:
 Target: (o.id = 'document') AND (s.role = 'employee')

 policy-1:
   Target: action = 'visualize(s, o)'
   on-authorization:
          (e.location = 'company-department')
           OR
```

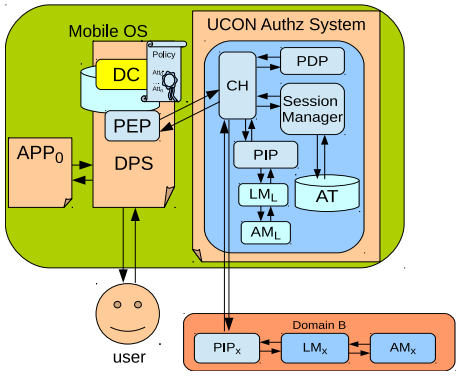

Fig. 2. Usage Control Architecture

```
        (e.inet = 'OFF') AND (e.bluetooth = 'OFF') AND (e.nfc = 'OFF')

policy-2:
  Target: action='replicate&send(s, s', o)'
  pre-authorization:
        (o.nOfCopies < N) AND (e.inet = 'ON') AND (s'.role = 'employee')
  pre-update:
        (o.nOfCopies++)
```

The manager wants to allow his employees to visualize the document on their tablets only when they are located within the building of the company department, or when the network interfaces of their devices are switched off (policy-1). This policy includes an on-authorization predicate which requires that the employee is located within the building of the company department, or that all the network interfaces of his devices are switched off. This predicate must be verified for the whole time the employee visualize the document. In fact, if an employee successfully opens the business document when he is at work, as soon as he leaves the company's building, the framework automatically detects it and closes the business document if at least one of the network interface on his device is switched on.

Another possible usage control policy in this scenario is the one that authorizes one (or more) employees to produce further data copies form their copy, but it allows a maximum of N copies of the same data (policy-2). This policy requires that the device is connected to the internet because the attribute which encodes the number of data copies is global and should be queried/updated from the remote attribute repository.

5 Prototype

This section describes the prototype architecture and implementation, and reports performance and security analysis.

5.1 Architecture Components

Figure 2 shows the components of the architecture. Though main security components reside on the mobile device, the architecture is distributed because the security attributes

required for the evaluation of the security policy might be spread over several domains. This architecture has been designed to deal with concurrent retrieval and update of mutable attributes, but we don't cover this aspect in this paper.

The main components are the following:

- Policy Decision Point (PDP) is a component which evaluates security policies and produces the access decision;
- Policy Enforcement Point (PEP) is a component which intercepts invocations of security-relevant access requests, suspends them before starting, queries the PDP for access decisions, enforces obtained decisions by resuming suspended requests, and interrupts ongoing accesses when the policy violation occurs;
- Context Handler (CH) is the front-end of the authorization system, that manages the protocol for communicating with PEPs and PIPs. It converts and forwards messages sent between components in the proper format. During the pre decision phase, the CH also contacts different PIPs and retrieves security attributes;
- Attribute Manager (AM) is a component which manages attributes and knows their current values;
- Policy Information Point (PIP) is a component which provides interface for other security components to query attributes. The PIP communicates then with the specific AM to retrieve attributes;
- Lock manager (LM) is a component which guarantees consistency in concurrent retrieval and updating of mutable attributes It determines whether the attribute query/update should be served or should be delayed by placing it in a queue and executing it later. The LM might be embedded into the AM;
- Session Manager (SM) is a components which is responsible for continuous control and manages ongoing usage sessions;
- Access Table (AT) is a component which keeps meta-data regarding ongoing sessions. It contains a table of the current sessions with their statuses and a table of IDs of the attributes needed to service each session;
- Data Protection System (DPS) is the component which allows the user to access the data. The DPS embeds the PEP in a way such that all the accesses to the data are controlled by the usage control system. The DPS keeps encryption keys to access the data and perform encryption/decryption is the access should be granted to the requester.

5.2 Components Implementation

UCON Authorization System Implementation. The core of our prototype is the UCON authorization system which implements functionality of CH, PDP, PIP, LM, AM, AT, and Session Manager. It was realized as Android application, UXACML Authorization App (see Figure 3), which consists of a set of Android services and each Android service implements specific component or several components of the architecture.

The front-end of the UCON authorization system, the CH, is a bound Android service which implements client-server synchronous interactions. Initially, the PEP binds to the CH and then sends access requests using inter process communication (IPC) of Android OS. Communications between the PEP and the CH are blocking, i.e., the code where the PEP is inserted can not continue unless an authorization decision is provided.

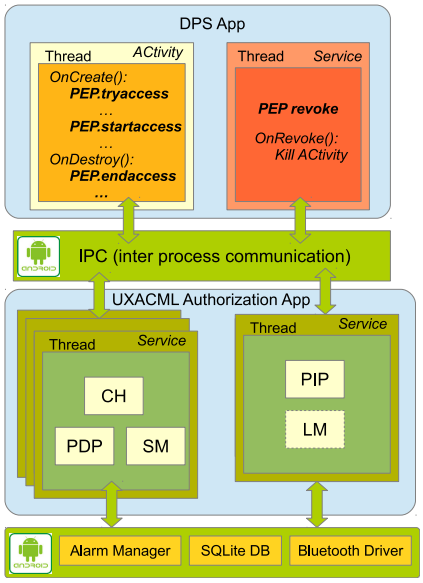

Fig. 3. Usage Control Implementation in Android

The CH may accept several access requests from different applications (PEPs) and each new request is evaluated in a separate thread. The Android OS handles a group of working threads to facilitate efficient evaluation of new requests. Therefore, the UXACML Authorization App is powerful to handle multiple calls (access requests) happening at the same time.

After receiving the access request, the CH pulls attributes from PIPs and then sends the access request and the UXACML policy to the PDP for the access evaluation. We modified Balana XACML Engine of OW2 in order to make it running on Android. If the access decision is "permit", the CH calls the Session Manager and the AT to store the metadata regarding new usage session, i.e., the UXACML policy, attributes needed for the continuous access revaluation, address of the PEP where the revocation message should be sent in case of the policy violation, etc. All session-related metadata are stored in the AT, Android SQLite DB, which is private data of the UXACML App and can not be accessed by other Android apps. Access to the AT is done to be thread-safe. Then, the CH subscribes to remote PIPs which holds mutable attributes needed for the continuous access re-evaluation. Finally, the CH responds to the PEP with access decision and this ends the pre-authorization phase.

The CH collects attributes from local and remote PIPs. The local PIP provides interfaces to environmental attributes (e.g, status of a bluetooth connection) and attributes related to local data copies (e.g., number of accesses to the DC stored on the device). Local attributes are stored in PIP's SQLite DB and the LM guarantees the consistent access to these attributes if multiple attribute queries are happening at the same time.

The CH queries remote PIPs for global attributes, e.g., the overall number of the DC in the system. We used implementation of the remote PIP described in [citation re-

moved for blind review]. The remote PIP is implemented as a web-service with locking deadlock-free concurrency mechanism to attributes stored in PostgreSQL DB. Besides, the PostgreSQL triggers allow the CH to subscribe for updates of attributes needed for continuous access re-evaluation. The CH and the remote PIP communicate via wi-fi network.

Indeed, in mobile scenarios the wi-fi network might be down and some remote attribute changes will not be delivered to the CH. To avoid such situations which lead to possible access violation, the CH is periodically triggered by Android Alarm Manager. The UXACML Authorization App administrator is in charge to specify re-evaluation interval. If during the access re-evaluation the CH is not capable to reach remote PIPs and local attributes do not make the policy applicable, the CH fires the access revocation for such sessions and sends corresponding messages to PEPs.

PEP Implementation. Figure 3 shows the example of where the PEP code should be inserted if the Android ACtivity is considered as a resource and time when the ACtivity is visible as the long-lasting security-relevant action.

The PEP binds to the UXACML Authorization App on “tryaccess” and unbinds on the “endaccess” or “revokeaccess” (i.e., when the ACtivity is destroyed). If the resource and actions on it are represented by another abstraction and the lifetime of the resource is longer than that of the activity/service which provides access to it, the binding between the PEP and the UXACML Authorization App is executed upon each security message exchange. The UXACML Authorization App also binds to the PEP if the revocation of usage session is detected. Thus, the PEP receives the revocation message (Android intent) and starts a new thread that handles the real access revocation.

5.3 Performance Analysis

We tested the performance of our prototype in the presence of many concurrently running usage sessions. We installed the UXACML Authorization App and the DPS App with embedded PEP on Motorola Moto G which runs Android 4.4 KitKat and is equipped with 1 GB RAM and Quad-Core 1.2 GHz Cortex-A7 CPU.

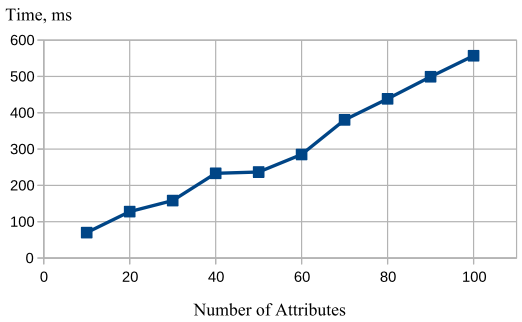

Fig. 4. Overhead of Pre-authorization Phase

First, we measured the overhead which occurs during the pre-authorization phase as a result of the access request construction, attributes retrieval, and evaluation of the policy against the access request (see Figure 4). Without loss of generality, we considered a scenario in which there is a PEP which should authorize the single request. We varied the number of attributes from 10 to 100 which are required by the UCON system to perform the decision process. As a matter of fact, there is a slight growth of t_{pre} with the number of attributes. In our tests, we used local attributes and non-updating UXACML policies (i.e., no attribute updates are needed as a result of access evaluation), while in real systems remote PIPs could be exploited, possibly increasing the time needed for attributes retrieval.

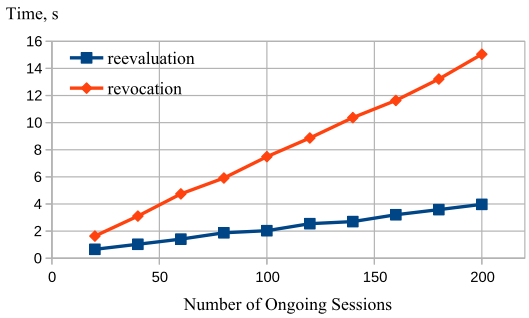

Fig. 5. Overhead of Ongoing Access Phase

Then, we measured the re-evaluation time of many sessions running concurrently varying the number of sessions from 20 to 200. All session used the same 10 attributes which were queried once but all session were re-evaluated independently and sequentially. Figure 5 (blue chart) shows the results obtained. We see that the time needed for re-evaluation of sessions grows linearly in the number of running sessions and for 200 ongoing sessions it is about 4 seconds. In fact, the results shown that the mobile device is powerful to handle a big amount of ongoing sessions.

Further, we measured the revocation time of all sessions whose policies use the same security attribute which changes its value from good to bad and violates the policies. The revocation time of all sessions defines the period of time passed from the point when the CH is triggered for access re-evaluation until PEPs receive revocation messages for all sessions that have to be revoked. Figure 5 (red chart) shows the results obtained. We see that the revocation time of all sessions grows linearly in the number of running session and for revoking 200 sessions we need about 15 seconds. The revocation time of all sessions is several times higher than the time needed just for re-evaluation of the access for these sessions. This is due to heavy resource consumption for the IPC between PEPs and the UXACML Authorization App.

Finally, we measured how many computation resources are consumed by the UXACML Authorization App which runs along with other applications on device and con-

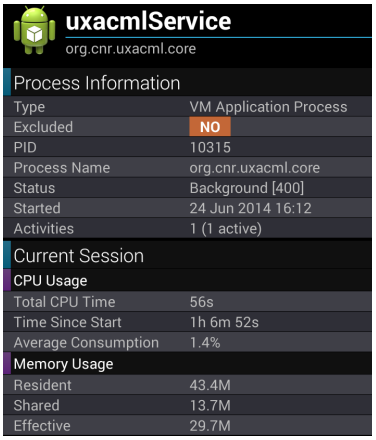

Fig. 6. Resources Consumption of UXACML Authorization App

sumes CPU, battery, and memory. Figure 6 shows the resources consumption in the case of 100 ongoing usage sessions and when the access re-evaluation is triggered every 30 seconds. With such load the the UXACML Authorization App takes just about 1.4 per cent of the CPU time and needs approximately 30 MB of RAM.

5.4 Security Analysis

This subsection presents a security analysis of the proposed system. The security of our implementation relies on the Android security support, enhanced with the use of a TPM, as discussed in Section 3.2. The proposed system prevents users (including other applications) from directly accessing the DCs downloaded on this device, which are encrypted. As a matter of fact, the DCs stored on the device must be accessed only through our Data Protection System, which enforces the related usage control policies. We consider the following model of attackers:

i) Users trying to access data stored in the SDCard. Users can browse the SDCard contents through a file manager application.
ii) Users trying to access encryption keys used to encrypt data stored on the SDCard.
iii) Users trying to acquire root privileges to access restricted memory space.

The implementation described in this paper assumes to have a mobile device with an embedded TPM, although no Android device currently provides it. The files including the DCs to be protected are stored in the mobile device external memory, i.e. in the SD-Card, and they are encrypted through the AES algorithm. This protects from attackers of type i) who will find encrypted files (and not usable) when accessing the SDCard of the device. The related secret key is stored in the device internal memory, in the home space of the Data Protection System application. Differently from the external memory, this space can only be accessed by the Data Protection System application, so that encrypted files can be decrypted only by the Data Protection System. This security mechanism protects the key from attackers of type ii). Thanks to the inclusion of

trusted computing, it is possible to ensure that only the Data Protection System application can access the secret key. In fact, the secret key will be no more accessible as soon as the device has been rooted or unknown dangerous applications have been installed on it. In particular, as soon as the integrity verification fails, the Data Protection System application and all the related files, residing in both the internal and external memory, are removed from the device. Furthermore the Data Protection System application will not be released anymore to the device owner, unless she can prove the device status is secure again, e.g. through a factory reset. This ensures the protection also from an attacker of type iii).

The communications between the Data Protection System and the PDP and between the PDP and the PIP are secured, i.e., the mutual authentication is performed to verify the identity of the communicating applications, and also the integrity of the messages is preserved through the standard Java security library: Java SE Security.

The security policy is paired with each data copy, and it is encrypted and stored with the data itself. In this way, the security policy cannot be modified by the mobile device user or by other applications to obtain unauthorized accesses.

The status of the UXACML Authorization app, e.g., the set of the sessions that are currently opened, is stored in the private space of his app. In this way, neither the device user nor the other applications can modify it.

Finally, if the mobile device user terminates the Data Protection System, and/or the UXACML Authorization app, this only prevents him from accessing the data, since the Data Protection System app is the only way to perform the access to the data.

6 Conclusions

In this paper we have discussed a framework for regulating data sharing on Android mobile devices. In this framework, the data producer embeds a usage control policy in the data, which is downloaded by the user and enforced on the device to prevent unauthorized access to specific pieces of data. The access regulation is performed over time through the usage control paradigm. In particular, the usage control policy is able to interrupt accesses in progress when a previously valid policy is not satisfied any more. We have also presented an implementation of the proposed framework for Android smartphones and tablets, which exploits both the Android native security mechanism and a TPM to ensure system integrity. The TPM inclusion is an assumption, since currently no Android devices with a physical TPM are available, though a standard for TPM on mobile device is currently under the analysis of the TCG. Implementation on an Android-based device with TPM has been scheduled as future work.

References

1. Park, J., Sandhu, R.: The $UCON_{ABC}$ usage control model. ACM Transactions on Information and System Security **7** (2004) 128–174
2. Morrow, B.: BYOD security challenges: control and protect your most sensitive data. Network Security **2012**(12) (2012) 5–8

3. Costa, G., Martinelli, F., Mori, P., Schaefer, C., Walter, T.: Runtime monitoring for next generation java me platform. In Computers & Security **29** (2010) 74–87
4. Aktug, I., Naliuka, K.: ConSpec: A formal language for policy specification. In: Proceedings of the First International Workshop on Run Time Enforcement for Mobile and Distributed Systems (REM 07), ESORICS (2007) 107–109
5. Jia, L., Aljuraidan, J., Fragkaki, E., Bauer, L., Stroucken, M., Fukushima, K., Kiyomoto, S., Miyake, Y.: Run-time enforcement of information-flow properties on android. In Crampton, J., Jajodia, S., Mayes, K., eds.: ESORICS 2013. Volume 8134 of Lecture Notes in Computer Science. (2013) 775–792
6. Conti, M., Crispo, B., Fernandes, E., Zhauniarovich, Y.: Crêpe: A system for enforcing fine-grained context-related policies on android. IEEE Transactions on Information Forensics and Security **7**(5) (2012) 1426–1438
7. Conti, M., Nguyen, V., Crispo, B.: Crêpe: Context-related policy enforcement for android. In: 13 Information Security Conference (ISC10). (2010) 331–345
8. Enck, W., Ongtang, M., McDaniel, P.: On Lightweight Mobile Phone Application Certification. In ACM, ed.: 16th ACM conference on Computer and Communications Security (CCS'09). (2009) 235–254
9. Enck, W., Gilbert, P., Chun, B.G., Cox, L.P., Jung, J., McDaniel, P., Sheth, A.N.: Taintdroid: An information flow tracking system for real-time privacy monitoring on smartphones. Commun. ACM **57**(3) (2014) 99–106
10. Zhou, Y., Zhang, X., Jiang, X., Freeh, V.W.: Taming information-stealing smartphone applications (on android). In: 4th International Conference on Trust and Trustworthy Computing (TRUST 2011). (June 2011) 93–107
11. Bugiel, S., Davi, L., Dmitrienko, A., Heuser, S., Sadeghi, A.R., Shastry, B.: Practical and Lightweight Domain Isolation on Android. In ACM, ed.: 1st ACM workshop on Security and privacy in smartphones and mobile devices (SPSM11). (2011) 51–61
12. Cerbo, F.D., Trabelsi, S., Steingruber, T., Dodero, G., Bezzi, M.: Sticky policies for mobile devices. In: The 18th ACM Symposium on Acces Control Model and Technologies (SACMAT'13). (2013) 257–260
13. Trabelsi, S., Sendor, J., Reinicke, S.: Ppl: Primelife privacy policy engine. In: 2011 IEEE International Symposium on Policies for Distributed Systems and Networks, IEEE Computer Society (2011) 184–185
14. Colombo, M., Lazouski, A., Martinelli, F., Mori, P.: A proposal on enhancing XACML with continuous usage control features. In: proceedings of CoreGRID ERCIM Working Group Workshop on Grids, P2P and Services Computing, Springer US (2010) 133–146
15. Trusted Computing Group: Tpm 2.0 mobile reference architecture (draft) (April 2014)
16. Bente, I., Dreo, G., Hellmann, B., Heuser, S., Vieweg, J., von Helden, J., Westhuis, J.: Towards permission-based attestation for the android platform. In: Trust and Trustworthy Computing. Volume 6740 of Lecture Notes in Computer Science., Springer Berlin Heidelberg (2011) 108–115
17. Zhang, X., Parisi-Presicce, F., Sandhu, R., Park, J.: Formal model and policy specification of usage control. ACM Transactions on Information and System Security **8**(4) (2005) 351–387
18. Zhang, X., Nakae, M., Covington, M.J., Sandhu, R.: Toward a usage-based security framework for collaborative computing systems. ACM Transactions on Information and System Security **11**(1) (2008) 3:1–3:36
19. Lazouski, A., Martinelli, F., Mori, P.: Usage control in computer security: A survey. Computer Science Review **4**(2) (2010) 81–99
20. Lazouski, A., Mancini, G., Martinelli, F., Mori, P.: Architecture, worflows, and prototype for stateful data usage control in cloud. In: 2014 IEEE Security and Privacy Workshop, IEEE Computer Society (2014) 23–30