Stateful Data Usage Control for Android Mobile Devices*

Aliaksandr Lazouski Fabio Martinelli Paolo Mori
Andrea Saracino
Istituto di Informatica e Telematica, Consiglio Nazionale delle Ricerche, Pisa, Italy
email:{name.surname@iit.cnr.it}

Abstract

Modern mobile devices allow their users to download data from the network, such as documents or photos, to store local copies and to use them. Many real scenarios would benefit from this capability of mobile devices to easily and quickly share data among a set of users but, in case of critical data, the usage of these copies must be regulated by proper security policies. To this aim, we propose a framework for regulating the usage of data when they have been downloaded on mobile devices, i.e., they have been copied outside the producer's domain. Our framework regulates the usage of the local copy by enforcing the Usage Control policy which has been embedded in the data by the producer. Such policy is written in UXACML, an extension of the XACML language for expressing Usage Control model based policies, whose main feature is to include predicates which must be satisfied for the whole execution of the access to the data. Hence, the proposed framework goes beyond the traditional access control capabilities, being able to interrupt an ongoing access to the data as soon as the policy is no longer satisfied. This paper details the proposed approach, defines the architecture and the workflow of the main functionalities of the proposed framework, describes the implementation of a working prototype for Android devices, presents the related performance figures, and discusses the security of the prototype.

*This work was supported by the EU FP7 project *Confidential and Compliant Clouds* (CoCoCloud), GA #610853 and by the H2020 EU-funded project *European Network for Cyber Security* (NeCS), GA #675320.

1 Introduction

The increasing popularity of mobile devices, such as smartphones, tablets and phablets, combined with the availability of the Internet connection everywhere, make modern mobile devices real competitors of desktop computer systems. As a matter of fact, new generation mobile devices are actually comparable to desktop computers both from the hardware and the software points of view. In particular, they have fast multi-core processors, a large storage capacity, and a strong connectivity because they can connect to the Internet through several interfaces, e.g., operator network (3G/4G) or WiFi, or they can communicate directly with other devices through Bluetooth or NFC interfaces. From the software point of view, modern mobile devices run operating systems which have the same functionalities as the ones running on Personal Computers, i.e, on which several applications (apps) can be installed and executed, thus offering a plethora of functionalities, from email and other office applications to audio and video applications or games.

Android is the leading operating system for mobile devices and it is run by almost 80% of mobile devices[1]. Other operating systems for mobile devices exist on the market, such as Apple iOS and Microsoft Windows Phone. 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 by trusted parties (e.g., Google Play Store), where developers upload and sell their applications. An-

[1]http://mobiforge.com/research-analysis/global-mobile-statistics-2014-part-a-mobile-subscribers-handset-market-share-mobile-operators

droid provides a robust security architecture designed to ensure the protection of the platform, i.e., the protection 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, as detailed in Section 4.1.

The previously described features make mobile devices an excellent platform for users to download data, store and use them. In some real scenarios, the usage of data outside of the producer's domain should be restricted according to some security policies (e.g., defined by the data producer), but the native Android security support does not allow to enforce such Usage Control policies. In particular, data are created by data producers who share them on the Internet, and once these data have been downloaded on mobile devices, typically no further controls are performed to regulate their usage.

Hence, this paper is focused on regulating the usage of data copies stored on Android mobile devices by enforcing the security policies based on the Usage Control model which are embedded in the data copies.

1.1 Contribution and Motivation

This paper describes the framework we defined to protect the data shared on mobile devices, focusing on Android ones. In our reference scenario, the data producer embeds a Usage Control policy in the data he creates before sharing them with other users. The other users download their copies of the data on their mobile devices, and the framework regulates the subsequent usage of these copies by enforcing the embedded policies. Since these policies are based on the Usage Control model (see Section 4.3), they are continuously enforced in order to allow the usage of the data as long as a set of conditions are satisfied. As soon as a policy is violated, the actions that are in progress on the data are properly interrupted.

Hence, the main contribution of this paper is the definition of an approach, the design of the related framework, and the implementation and validation of a prototype for regulating the usage of data copies which have been downloaded on Android mobile devices.

The novelty of the proposed approach and framework is that the Usage Control policy paired with the data is enforced in a continuous fashion while the data are used on the Android device where they have been downloaded. This allows for a quick reaction to any change of the attributes describing the access context in order to detect a possible policy violation and suspend the data usage. In fact, the Usage Control policy takes into account mutable attributes of users, data and environment, i.e., attributes that frequently change their values because of the normal operation of the system. The motivation leading to the definition of the proposed framework is that the wide spreading of mobile devices and the availability of the Internet connections would encourage users to perform the sharing of their data with others through mobile devices. However, the lack of a proper security support ensuring the protection of the shared data could instead prevent users to perform such sharing. The proposed framework fills this gap by providing the security support required to protect the shared data by regulating their usage.

This paper extends our previous work [1] by: *i)* presenting a complete and detailed description of the framework architecture and of the interactions among its components; *ii)* introducing a new use case example to show the capabilities and the applicability of the proposed framework; *iii)* presenting a new set of extensive experiments which measure the performance of the framework both in case of local and remote attributes, taking also into account attribute updates; *iv)* providing a more detailed description of the use of TrustZone as a module to enable TPM security.

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 and they are on duty. 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.

1.2 Paper Structure

The paper is structured as follows. Section 2 describes two reference examples that will be used throughout the paper. Section 3 describes some work related to the protection of data and resources on mobile devices, while Section 4 describes some background on which the pro-

posed approach is based, such as the Usage Control model, the security support of the Android operating system, and the Trusted Platform Module. Section 5 describes the core of the proposed approach, i.e., the adoption of the Usage Control model to protect the usage of local copies of data, along with simple security policies for the reference examples. Section 6 gives a complete and detailed description of the architecture of our framework and of the interactions among its components, while Section 7 describes the prototype implementation along with a set of performance figures and a security analysis. Finally, Section 8 draws the conclusions.

2 Reference Examples

This section describes a couple of examples of data sharing on mobile devices where such data are personal and/or have high economic value, so the proposed framework could be successfully applied to regulate their usage.

Example 1: Let us suppose that a team manager T of a company C wants to share with his employees some strategic documents he wrote, such as business reports or contracts, exploiting the company's tablets. The team manager is the data producer, because he created these documents. Since these business documents are critical for the company, the manager needs to impose some restrictions on their usage. First of all, no modifications to the documents issued by the manager can be done by the employees. Moreover, T wants that each employee is allowed to visualize only the documents related to the projects he is currently working on, and that each employee can access only documents related to the same project at the same time. We suppose that the set of projects assigned to each employee can change over time. T also wants to allow his employees to visualize the documents on the company's tablets only when they are located within the building of the company, they are on duty, i.e., they have clocked in, and only during working hours. Finally, T wants to allow the department heads to create further copies of the documents related to the projects they are involved in, up to a maximum of N copies for each document.

Our framework can be exploited by the team manager to perform a controlled sharing of his documents with his employees. In particular, our framework allows the manager to embed a policy in his documents which allows visualizing and copying of the document according to the previous constraints. In this way, if an employee tries to open the business document when he is at home, the framework will not visualize the document. Moreover, if an employee successfully opens the business document when he is at work and he is on duty, as soon as he leaves the company's building or he clocks out, the framework enforces the policy by closing the business document. Finally, if the employee opens a document related to project A and after he opens another document related to project B, the framework enforces the policy by closing the document related to project A.

Example 2: Let us suppose that a specialist doctor, a radiologist, wants to share a document representing the result of an examination he performed on a given patient P with some other healthcare professionals. However, the document representing the examination result, being health-related data, must be properly protected to guarantee the patient's privacy. Hence, the radiologist would like to grant the access to this document only to other specialists in orthopedics, who will define the treatment, and to the nurses of the department of orthopedics as long as P is hospitalized in that department to administer the treatment. Moreover, the ragiologist would like that only one nurse at a time can visualize that document.

Finally, in the e-health scenario the patient must give his explicit consent in order to grant the rights to access the document including his examination results to the previously mentioned subjects, and we assume that this consent is sufficient to be fully compliant with European laws. However, this consent could be withdrawn by the patient at any moment in time. In this case, nobody is allowed to access the document any more.

We recall that healthcare professionals, such as the specialist in orthopedics and the nurses in our example, cannot change the results of the examination issued by the radiologist (which are signed by the radiologist). Instead, they could append the new contents they produce to the document.

Again, our framework can be exploited in this scenario to control the sharing of the examination result produced by the radiologist on the mobile devices of the healthcare organization. In particular, our framework allows the radiologist to embed a policy in the examination result which allows its visualization according to the previously de-

scribed constraints. In this way, only specialists in orthopedics can always visualize this examination result on the devices of the healthcare organization. Instead, the nurses of the department of orthopedics can display the examination result on the hospital's tablets while P is hospitalized, and they lose this right as soon as P is discharged from the hospital. Moreover, supposing that a nurse downloaded the examination result on a device and she left it opened, the visualization is interrupted as soon as P is discharged from the hospital. Finally, our framework is also capable to manage the patient consent, even allowing to interrupt an ongoing visualization of the document as soon as the patient withdraws his consent.

It is worth noticing that the previous examples present situations where the owner of the device is not directly the user of the device and this justifies the role and presence of the TPM. Moreover, we also notice that in some scenarios the data producer is not the only subject entitled to access the data he shares, but other people could have some rights on these data according, e.g., to laws or regulations. In these cases, the data producer could abuse our framework by preventing people having rights on the data he shares from accessing them by setting up unduly constraining policies on these data. However, the solution to avoid this problem is developed in the ambit of the Coco Cloud project[2], of which the proposed framework is part, where the policy authoring process was designed taking into account also legal aspect, such as data protection regulations, and user preferences [2]. Hence, the policy authoring process ensures that all the stakeholders have their right respected. This also implies that proper usage of the TPM should be fostered.

3 Related Work

In the last years, several frameworks have been proposed to enforce security policies on Android devices. These frameworks either exploit native security mechanisms, or include new features for a direct fine-grain control of the Android's API. In the following, we will survey some relevant work with a focus on those frameworks enforcing access and usage control on both Android and other mobile systems.

The work described in [3] proposes a framework for 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 our paper because it is focused on controlling the information flow among applications. Our work instead, as previously discussed, does not consider information flow issues, since controlled data are not exchanged between apps and every information exchange is handled through the secure Android native communication mechanisms. A system for data Usage Control is proposed in [4] and [5]. This system is focused on data flow analysis for cross-boundaries distributed systems, presenting a formal model for data transmission in order to define a distributed enforcement infrastructure. To this end the analysis is largely focused on network connections, in particular an application to TCP is presented. Our framework, instead is more focused on aspects of data Usage Control enforced on end-devices, which in the specific are Android smartphones or tablets, interacting with a cloud Usage Control framework. Since data are downloaded from the cloud, which is a logically centralized entity and thus stored on the device, without the possibility of being exchanged between devices, or with third parties, data flow issues are not considered in the present work, which is mainly focused on challenges and solutions for enforcing data Usage Control on Android systems. As an extension of the native permission system, several frameworks have been proposed to enforce a finer grained access control model, which however is mainly directed toward securing operations and device resources, rather than pieces of data or documents, which is instead the focus of the present work. 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 con-

[2]http://www.coco-cloud.eu/

text, 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).

Access control has been enforced on mobile devices also on systems previous to Android. The authors of [8] propose a framework to enhance mobile devices security focused on the protection of mobile devices resources 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 [9]. 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. As for the previous, this framework is different from the one proposed in our paper, since it is focused on the protection of device resources and operation, rather than data. Access control to privacy sensitive information has been addressed in [10], which proposes a framework called *TrustDroid* to define access level to specific device resources and operation. TrustDroid also allows to define some context based policies, but these 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 usage control framework designed for Android devices is UCDroid, presented in [18]. This framework exploits the usage control engine to dynamically revoke and grant authorizations to apps, through the permission system. Differently from the present work, UCDroid limits its action to the features controllable through Android permissions and does not enforce control on data access and usage. Another work focusing on the analysis of access to security critical resources is *TaintDroid* presented in [11]. This framework is based on hooking 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. A work similar to ASF is the ASM framework presented in [12]. The paper in [13] presents a framework for context-based access control. This framework exploits probabilistic techniques to understand automatically the user context, demanding the enforcement of such policies to the access control framework presented by the same author in [14]. This system, named *flaskdroid* extends the native access control mechanisms, i.e. permissions system, with a more flexible ones which also allows the definition of security policies. A system with similar functionalities is presented in [15]. The framework presented in [16] is aimed at avoiding disclosure of private information. However, this work mainly focuses on controlling the actions made by users, providing a secure user interface aimed at avoiding accidental disclosures. On the other hand our framework is more focused on allowing the controlled sharing of data on Android devices, denying and or revoking the access only when the Usage Control policies do not allow the access to them. Thus, being able to enforce more complex policies, the presented framework is in the end more general.

Another set of frameworks merely focused on Access Control in Android is the following. *Android Security Framework* (ASF) presented in [17] provides security APIs to developers interested in writing security extensions. Target of this framework are both manufacturers and governments interested in enforcing specific security policies on users' mobile devices. This shapes ones of the main difference with our UCON framework, which is instead more general, being usable both by companies employees and by normal users of cloud environments.

A methodology which is similar to access control, generally applied on multimedia data is the Digital Right Management (DRM). The work in [19] presents an overview on DRM mechanisms available for Android devices, which mainly aims at protecting the code and data of an application. The presented mechanisms are based on the idea of downloading the file on the device from the server, only if the DRM policy is verified. The overhead introduced by such a methodology is consistent, especially for large files. On the other hand, our framework keeps the data already on the device, secured through encryption. The overhead caused by the used symmetric encryption method is sensitively lesser than the one introduced by continuous downloads. Moreover, the policies which could be defined by our framework are in principle more general than the ones proposed in [19], which

are bounded to the DRM standard. Ongtang et al. propose in [20] a framework to handle DRM on Android devices named Porscha. The proposed system introduces the possibility to control access to documents stored on the device, using DRM policies to assess the applications and the device type on which data can be accessed. Some simple policies on period and number of uses are also supported. However, differently from our system, Porscha does not support dynamic policies. Furthermore, our framework allows to define more complex and general policies, based on several attributes which can also be mutable.

The work presented in [21] describes *Idea*, a system to enforce security policies on Android systems by inlining of reference monitors to control security relevant actions. The system is effective in enforcing complex security policies, but the imposed performance overhead of 35% is hardly acceptable. From the same authors, a lightweight framework for enforcing of security policies, but more oriented at increasing the flexibility of the permission system is *AppGuard* [22] [23]. However, the presented framework is not focused on data protection and also requires the modification of app code to allow code instrumentation. On the contrary, our framework does not perform any modification to apps and system code, also the overhead is limited.

A framework for Android that does not require the modification of the operative system is presented in [24]. This framework, named Aurasium, enforce security policies to applications by repackaging them, modifying the functions that could harm the user security. Still, Aurasium approach is not focused on specific access and Usage Control policies, moreover, differently from our framework Aurasium requires modification of the controlled apps code. The work proposed in [25] 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. Our framework, instead, allows the definition of finer grained policies, which are taking into account attributes potentially coming from any domain.

A well known framework to enforce security in mobile systems is the Security-by-Contract (SxC) framework [26]. The rationale behind SxC is the idea that application are shipped together with a contract specifying security relevant actions performed by the applications. The contract is matched with the device's security policies. Thus, if the contract matches the policy, the application runs without any additional control, otherwise a monitor is attached to enforce the security policy on the app. Starting from the idea that the android manifest itself is a contract, an application of SxC to Android has been proposed in [27]. This framework extends the Android manifest including the complete contract, which describes the probabilistic behavior of the application. This extension of the SxC allows also the definition of policies with probabilistic clauses, more flexible than those of classic SxC. The drawbacks of the SxC platform is the assumption that every developer is able and willing to generate a contract for the application. Also the SxC requires the device to be rooted. The proposed Usage Control framework instead does not require any modification to device nor installed apps.

A preliminary work concerning sticky policies for mobile devices is presented in [28] 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 are some significant differences between the work in [28] and the one presented in this paper. Mainly, it is due to the security policy languages that the two frameworks enforce, respectively, PPL [29] and U-XACML [30]. 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 reevaluated 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 [28] does not provide perfor-

mance analysis. For further references to security frameworks in mobile systems, the interested reader can refer to the survey presented in [31].

4 Background

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

4.1 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 latest 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 isolated runtime environment implemented through Virtual Machines (VM) and the underlying Linux kernel. In Android every application runs in a VM named Dalvik Virtual Machine (DVM) up to release 4.4 and Android Runtime Environment (ART) in the following releases. DVM and ART are an optimized version of the Java Virtual Machine, in particular ART also includes the support for Ahead of Time compilation for improved performance. In DVM and ART 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 VM instance 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.

4.2 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 a 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 smartphones and tablets. The system model presented in [32] 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 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 en-

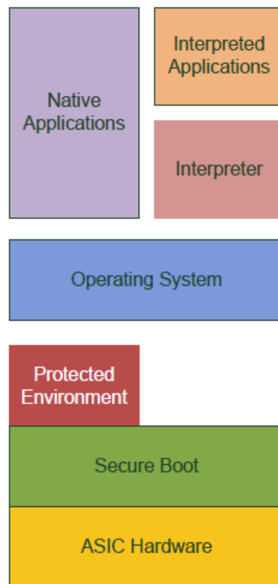

Figure 1: Secure Mobile Device Architecture Proposed by the TCG

vironment 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's 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, the integrity of the kernel, of the operative system and of the application installed on the device is verified. When the integrity of a level cannot be verified, the device is considered non-secure from that level to higher ones. Since the mobile TPM has still to be standardized, currently there are no off-the-shelf Android devices supporting TCB. However, some steps have been done recently toward the inclusion of TCB in Android through the use of `TrustZone`.

`TrustZone` [33] is a component which is by default available on the greatest majority of ARM processors. Thus, the main component to enable TCP functionalities is already present on almost any Android-powered smartphone and tablet. Still, device manufacturers do not include, up to now, the possibility to interact on the user side with the `TrustZone` component.

However, the potential of having trusted computing base-enabled mobile devices, has already been envisioned in recent works, both from industry and academia, proposing implementation based on `TrustZone` on non-off-the-shelf devices. In particular Samsung proposes the *Samsung Knox* [34] service to its customers. This service exploits the ARM's `TrustZone` technology [33] to create a trusted computing base on mobile devices, which interacts with an external framework to implement different security policies. Devices with Samsung Knox compatibility are not distributed to private users, but only to companies interested in implementing BYOD policies. Another work exploiting TrustZone is [35], which implements a secure storage platform on Android, exploiting the trusted memory space provided by the TrustZone environment. The system has not been implemented on smartphones or tablet, but on a programmable board running the Android operative system.

However, the TCP building blocks that have to be included on the Android platform have been described in [36], 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 noting that the TrustZone is a component embedded onto mobile devices processors. We will assume it to be delivered with an Android version with APIs to interact with the TrustZone . 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.

4.3 Usage Control Model

The Usage Control (UCON) model, introduced in [37], has been designed to regulate the usage of modern and distributed systems, and it encompasses and extends traditional access control models introducing mutable attributes and new decision factors besides authorizations, i.e., obligations and conditions. This section summarizes the main concepts of the UCON model, but for a detailed (and formal) description please refer to [38] [39].

Mutable attributes encode features of the subjects, of the resources, and of the environment, which change their values over time, and this could affect the execution rights of other accesses that are in progress [40]. These attributes could change their values because of attribute update statements included in the Usage Control policies, which can be executed before (*pre-Updates*), during (*ongoing-Updates*), or after (*post-Updates*) the execution of the access, because of actions performed by the user, or for other reasons. For instance, the value of the attribute which represents the number of copies of the same data in the systems is increased by a *pre-Update* policy statement every time the creation of a new copy is authorized. Instead, the position of a user changes every time the user moves from one location to another. Traditional attributes (i.e., *immutable attributes*), instead, do not change frequently, and they are modified only through administrative actions. For instance, the *role* attribute is updated when the subject gets a career advancement. Since mutable attributes can be updated during the usage of an object, in the following we show that each decision factor can be evaluated before (as in traditional models) and/or during the usage of the object (*continuous control*). Reevaluating the access right when the access is in progress and interrupting this access when the related right is no more valid reduces the risk of misuse of resources.

Authorization predicates are evaluated to determine whether a subject requesting access to an object holds the corresponding right. This decision making phase takes into account subject/object attributes, and the action that the subject requested to perform on the object. The UCON model defines two categories of authorizations: pre-Authorizations (*preA*), where the decision phase is performed when the subject requests to access the object, and ongoing-Authorizations (*onA*), where the decision phase is performed while the access is in progress, in a continuous way.

Obligations are predicates that checks whether certain requirements have been fulfilled in order to access objects. Pre-oBligation (*preB*) predicates verify whether some requirements have been fulfilled before the access, while ongoing-oBligations (*onB*) continuously check that the requirements are fulfilled while the access is in progress.

Conditions are requirements that do not depend on subjects or objects. They evaluate environmental or system status (e.g., current time) to decide whether to allow access or not. Pre-Conditions (*preC*) are evaluated at access request time, while ongoing-Conditions (*onC*) are continuously evaluated while the access is in progress.

The UCON model has been successfully adopted in many different scenarios, such as Web, Grid, Cloud or Next Generation Networks (NGG) to protect the usage of several kind of resources.

In [41], Sandhu et al propose the adoption of their model in collaborative computing systems, such as the GRID environment. In their architecture, they propose a centralized Attribute repository (AR) for attribute management, that works in push mode (i.e. the attributes value is submitted to the authorization service by the user himself) for immutable attributes, and in pull mode (i.e. the attributes value are collected by the authorization service just before their use) for mutable attributes. They use the eXtensible Access Control Markup Language (XACML) [42] to specify several aspects of the Usage Control model. Basically, they have more than one XACML policy for the different policies necessary in the Usage Control model.

Alexander Pretschner et al, in [43], propose an Usage Control enforcement mechanism for applications, showing an implementation for a common web browser and using this prototype to control data in a social network. The proposed mechanism allows the data owner to prevent data from being printed, saved, copied&pasted, etc., after this data has been downloaded by other users. Instead, in [44] Pretschner shows an application of the Usage Control model to preserve people privacy in video surveillance systems.

The authors of [45] propose a Process Algebra based authorization system based on Usage Control for pro-

tecting GRID computational services. The policy is expressed exploiting a process description language, which is shown to be suitable to model the core usage policy models of the original UCON model. Moreover, they describe a prototype implementation for GRID computational services, and they show how the proposed language can be used to define a security policy that regulates the network usage to protect the local computational service from the applications executed on behalf of remote GRID users. The Usage Control authorization system prototype has been validated through a proper testing process described in [46].

The work in [47], instead, addresses the problem of continuous Usage Control when multiple copies of a data object are stored in distributed systems, such as the Cloud. This work defines an architecture, a set of workflows, a set of policies and an implementation for the distributed enforcement. The policies, besides including access and usage rules, also specify the parties that will be involved in the decision process. In fact, the policy might be evaluated on one site, enforced on another, and the attributes needed for the policy evaluation might be stored in (many) other locations.

5 Usage Control for Data Protection

The framework proposed in this paper enables Data Producers (DPs) to protect the data they share by regulating their usage when these data are stored outside the DPs' domains, i.e., on users' devices. It has been designed for allowing the controlled data sharing through mobile devices running Android operating system, but the approach could be applied to other kinds of devices. The framework is based on an application, namely the Data Protection System application (DPS app), running on the mobile device which allows Data Users (DUs) to download from the sharing server a copy of the data (Data Copy, DC) on their mobile devices, and which is the only way to access these copies on the devices. In particular, each DC embeds its Usage Control policy, and the DPS app allows to access a DC through a set of predefined actions only, always enforcing the embedded policy to regulate the DC usage. In each scenario, the DPS app implements these actions for the specific kind of data that are shared. For instance, in the reference scenario described in *example 1*, the shared data are text documents, and the application is a text document reader.

The data protection policies are based on the Usage Control model, described in Section 4.3, which is defined on top of the following core components: subjects, objects, actions, attributes, authorizations, conditions, and obligations. This section briefly describes how the core components are instantiated in our scenario focussing on the reference examples described in Section 2.

The *subjects* are the Data Users who exploit the DPS app to perform some actions on the local DCs (which are the *objects*) that have been downloaded on their mobile devices. In *example 1*, the employees are the subjects and the business document is the object that is shared. In *example 2*, instead, the orthopedists and the nurses are the subjects and the examination result is the object. Each DU has a unique id, which is represented by a subject attribute. Each DC has a unique id as well, and it is also paired with the id of the original document. These two ids are represented by attributes of the object, respectively, named copyId and id. When a new DC is produced, the related copyId is derived from the one of the source by adding a proper suffix. The id of the original document paired with the new DC, instead, is the same as the one paired with the source. For instance, with reference to *example 1*, let us suppose that a department head, whose unique id is "paolo.mori", holds on his device a copy of a business document whose id and copyId are, respectively, "12gr67h" and "12gr67h.345g6y5". When the department head produces a new document copy from the one he has, the new copy will have the same id as the source, i.e., "12gr67h", while its copyId will be obtained by adding a proper suffix to the copyId of the source, e.g., "12gr67h.345g6y5.ret78gr". Obviously, distinct copies obtained from the same document will have different copyIds.

The DPS app implements the set of *security relevant actions* which represent the only way for accessing and operating on DCs. This paper exploits the following small set of security relevant actions, which cover the reference examples described in Section 2.

- $read(s, o_{dc})$: the DU (s) reads the DC object o_{dc};
- $append(s, o_{dc}, nd)$: the DU (s) appends new data nd to the DC object o_{dc};

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

The read action is exploited by the DPS app to visualize DCs on the mobile device, while the append action is meant to append further data at the end of DCs. The replicate&send action creates a copy of the DC to be sent to another DU, while the delete action allows to cancel the DC from the device. Please note that our framework does not allow pieces of data of a DC to be copied into other documents or to be transferred to other applications of the mobile device, in order to avoid policy enforcement issues. In fact, the usage of each piece of data must be always regulated by its original policy, even when it is copied on another document or transferred to another application. An interesting solution to address these issues is presented by Kelbert and Pretschner in [4] and [5].

The attributes paired with DCs can refer to: *i)* a specific DC, e.g., the creation date; *ii)* a subset of DCs, e.g., the number of data copies created after a given date; *iii)* all DCs derived from the same data object, e.g., the global number of data copies. The immutable (traditional) attributes of the DC, e.g., id, copyId, creation date, or producer, are typically embedded in the data object itself, because their values will not change during the objects lifetime. Mutable attributes of data, instead, are not embedded in the data object because a large overhead for updating them and for keeping consistency among all copies of the same attribute would be required. Consequently, mutable attributes of data are stored on proper Attribute Managers (AMs) that are invoked to retrieve the current values of these attributes when required to perform the decision process and to perform the attribute updates.

The attributes of the subjects and of the environment, both mutable and immutable ones, are managed by proper AMs as well, and these servers are exploited by our framework to retrieve the current values of these attributes every time it performs the decision process.

Our framework is general, and it supports attributes provided by AMs which run on the mobile device, as well as attributes provided by remote AMs, i.e., AMs which run on remote servers. For instance, with reference to *example 1*, the role of the user is a static attribute that could be represented by a credential issued by his company, and it could be stored and managed by a local AM, while his status (on or off duty), that is a mutable attribute, could be stored in a remote AM connected with the timecards system of the company. Another example is the user location, which is a mutable attribute that could be provided by an AM running on the mobile device.

Our approach assumes that, for each specific scenario in which it is adopted, a proper set of attributes is defined. These attributes represent the specific features of subjects, objects, and environment which are relevant for that scenario in order to be able to define and enforce the desired Usage Control policies. Consequently, a proper set of AMs should be in place to manage such attributes, and the way in which each of these attribute is updated depends on the specific feature encoded by the attribute itself. In fact, some attributes are updated because of attribute update statements in the usage policy (*pre-Updates*, *ongoing-Updates*, and *post-Updates*), while other attributes are updated as a consequence of other events or actions performed by the subjects (e.g., the updates of the subject position attribute are due to the movements of the subject). For instance, with reference to *example 1*, we assume that the AM providing the global number of DCs derived from the same data object is invoked: *i)* to retrieve the current attribute value every time is required to evaluate the policy; *ii)* to increase the attribute value every time a new copy is created; *iii)* to decrease the attribute value every time an existing copy is destroyed. Since in this example we assume that the delete action is the only way to delete a DC and, consequently, to decrease the global number of data copies attribute, this attribute takes also into account the copies stored on unreachable mobile devices (e.g., broken or disconnected devices). As a matter of fact, these devices could be repaired and reconnected to the internet after a while.

In the data protection scenario, *authorizations* are predicates involving attributes of DUs and DCs. The reference example *example 1* requires the definition of Usage Control policy including several authorization predicates. For instance, one of these predicates states that a subject s is allowed to visualize a given document o only if he is an employee of the company, and another predicate requires that o is related to a project on which s is currently working on. As previously stated, the role of a person in the company, such as “employee”, is represented by an at-

tribute paired to the subject, while the project a document refers to, is an attribute of the object. Since the previous attributes are immutable, the previous predicates are pre-Authorizations. The policy could require that some authorization predicates are satisfied continuously during the access time (ongoing-Authorizations), because some attributes of the subject or of the object could change while the access is in progress in a way such that the policy is violated. As an example, the attribute which represents the state of the employee requires continuous control because, for instance, the employee could change his state from “on duty” to “off duty” clocking out while his access to the document is still in progress, thus violating the policy.

Conditions are environmental factors which do not directly depend upon subjects or objects. For instance, in *example 1*, the condition predicates involve the current date and time. In particular, the current date and time is evaluated both when the access is requested (*pre-Condition*) in order to grant the access only during the working hours, and continuously while the access is in progress (*ongoing-Condition*), in order to revoke the access as soon as the current date and time is off working hours.

Obligations verify the fulfilment of some mandatory requirements before performing an action (*pre-oBligation*), or while performing it (*ongoing-oBligation*). For instance, the policy could state that an email must be sent to the DP every time some new data are appended to a DC.

5.1 UXACML Policy Language

Our framework exploits the UXACML language to express enforceable policies, because UXACML is an extension of the XACML language we explicitly designed to support the peculiarity of the Usage Control model, i.e., the continuity of policy enforcement. In fact, the XACML language [42], is a well-known and consolidated standard developed by the OASIS consortium for expressing access control policies in a distributed environment, and we extended it with the constructs required for Usage Control as follows.

UXACML represents the continuity of policy enforcement by adding a new clause, called `DecisionTime`, in the `<Condition>` and in the `<ObligationExpression>` elements of the XACML language. The `DecisionTime` clause defines when the evaluation of conditions or obligations must be executed. The admitted values are `pre` and `on` denoting, respectively, pre and ongoing decisions. In this way, the conditions whose decision time is set to `pre` are usual XACML conditions, while the conditions whose decision time is set to `on` must be continuously evaluated while the access is in progress. We recall that XACML conditions are exploited in UXACML to represent both UCON authorizations and conditions. In the same way, the obligations whose `Decision Time` clause is set to `pre` are usual XACML obligations, while ongoing-obligations will be expressed by setting the `DecisionTime` clause to `on`.

To change the values of *mutable attributes*, UXACML introduces a new element, `<AttrUpdates>`, which allows to define the attribute update statements in the policy. This element includes a number of `<AttrUpdate>` elements to specify each update action. Each `<AttrUpdate>` element also specifies when the update must be performed through the clause `UpdateTime` which can have one of the following values: `pre` (pre-Update), `on` (ongoing-Update) and `post` (post-Update). In case of ongoing-Updates, the update statements also specify the events which trigger the update. A detailed description of the UXACML policy language can be found in [30].

5.2 Policy Example

This section gives a couple of examples of Usage Control policies taking into account the reference scenarios described in Section 2. The proposed examples are meant to show the capabilities of our approach, and they are very simple with respect to the Usage Control policies enforced in real scenarios. Moreover, the authoring of data Usage Control policies in real scenarios could be a complex task involving several actors. For instance, in the scenario of *example 2*, besides the radiologist, also the patient should be allowed to express his policy to regulate the usage of the document representing the results of his examination. Since the framework proposed in this paper is part of the Coco Cloud project, we assume to exploit the policy authoring tool developed within this project to create proper Usage Control policies for real scenarios, since

it allows multiple actors to participate to the definition of such policies [2].

Although in the proposed framework enforceable policies are expressed using the UXACML language, for the sake of simplicity this section exploits a human-readable language because the UXACML one is too verbose. Table 1 shows a representation of the Usage Control policy of *example 1*, where a team manager wants to regulate the sharing of some critical business documents with his employees. In this policy, s represents the subject requesting the access, and o is the document that s wants to access. Moreover, s.X represents the attribute X of the subject s (e.g., s.role represents the attribute role of the user s), while o.Y represents the attribute Y of the document o.

The policy of Table 1 is actually a policy set consisting of two policies (namely policy-1 and policy-2), which is applicable only to the document it is attached to (whose original document id, represented by the attribute o.id, is ”12gr67h”). The target of policy-1 is the security relevant action ”read(s,o)”. The first pre-Authorization predicate of policy-1, (”employee” ∈ s.role), requires that the company grants the role ”employee” to the user requesting the access. We assume that department heads also hold the role ”employee”. This is a pre-Authorization because the attribute involved in the predicate (s.role) is immutable in our example, i.e., the role is assigned to the user through an administrative action. The attribute s.projects represents the list of projects assigned to the user s by the team manager T, and Policy-1 checks that the value of the attribute o.project is included among the values of s.projects both in the pre-Authorization section and in the ongoing-Authorization section, exploiting the predicate (o.project ∈ s.projects). This ensures that the employee can open the document o only if o belongs to one of the projects assigned to him, and this document will be closed as soon as this project is removed from the list s.projects by the team manager T. Moreover, *example 1* requires that the employee is on duty and is located within the building of the company while he accesses the document. Hence, policy-1 includes pre-Authorization predicates which check the mutable attributes of the user s.onDuty and s.location at request time in order to grant the access to the document, and ongoing-Authorization predicates which continuously check the value of the same attributes while the access is in progress in order to revoke such access as soon as a change in these attributes causes a policy violation. Policy-1 also includes an authorization predicate which ensures that the user is working on documents of the same project only at the same time. In fact, the id of the project the document o belong to is assigned to the attribute s.lastOpenedProject in the pre-Update section, i.e., when the access to o is granted, and the pred-

```
policy-set:
  target:
    (o.id = ”12gr67h”)

  policy-1:
    target:
      (action = ”read(s, o)”)
    pre-Authorization:
      (”employee” ∈ s.role) AND
      (o.project ∈ s.projects) AND
      (s.onDuty = TRUE) AND
      (s.location ∈ COMPANY_LOCATIONS)
    pre-Condition:
      (e.dateTime ∈ WORKING_HOURS)
    pre-Update:
      (s.lastOpenedProject := o.project)
    ongoing-Authorization:
      (o.project ∈ s.projects) AND
      (s.lastOpenedProject = o.project) AND
      (s.onDuty = TRUE) AND
      (s.location ∈ COMPANY_LOCATIONS)
    ongoing-Condition:
      (e.dateTime ∈ WORKING_HOURS)

  policy-2:
    target:
      (action = ”replicate&send(s, s’, o)”)
    pre-Authorization:
      (”departmentHead” ∈ s.role) AND
      (o.project = s.project) AND
      (o.nOfCopies < N)
    pre-Update:
      (o.nOfCopies++)
```

Table 1: Security policy for *Example 1*

icate (s.lastOpenedProject = o.project) in the ongoing-Authorization section checks that no documents related to other projects are opened while o is being read. Supposing that in our framework the attribute s.lastOpenedProject is managed by a remote AM, the policy takes into account the documents opened on all the devices of the user. Moreover, the pre-Condition and the ongoing-Condition sections include a predicate which controls that the access is performed during the working hours by checking the value of the environmental attribute e.dateTime. Ongoing-Authorizations and ongoing-Conditions must be verified for the whole time the employee visualizes the document. The position is a mutable attribute that automatically changes due to the movements of the user. The attribute s.onDuty changes its value when the user clocks in (the new value is TRUE) and out (the new value is FALSE). Finally, the mutable user attribute lastOpenedProject is changed by the pre-Update clause of policy-1 when the user visualizes a new document. The framework automatically detects when one (or more) of these attributes changes its value, it performs a new evaluation of the ongoing authorizations and conditions, and it interrupts the reading of the business document if the policy no longer grants the access right to the user.

The policy set includes another Usage Control policy, policy-2, which authorizes department heads working on a project to produce further copies of the project documents from their copies, but it only allows a maximum of N copies of the same document. The attribute representing the number of existing data copies is mutable and global. In fact, the value of this attribute is updated by the pre-Update predicate of policy-2, and this requires to adopt a remote Attribute Manager.

The DU can download the same document on two (or more) devices of his. In this case, distinct copies of the original document are created, and each device operates on its local copy. Race conditions could arise because the simultaneous access to these copies on distinct devices results in the concurrent enforcement of the same Usage Control policy which, in turn, causes concurrent reads and updates of the same remote attributes. However, these issues are managed by the Lock Manager which is paired to each AM, as described in Section 6.

```
policy-set:
  target:
    (o.id = ”sd4n68k”)

  policy-1:
    target:
      (action = ”read(s, p, o)”)
    pre-Authorization:
      (”nurse” ∈ s.role) AND
      (s.department = ”orthopedics department”) AND
      (p.hospitalized = ”orthopedics department”) AND
      (o.patientConsent = TRUE)
    pre-Update:
      (o.openedBy := s.id)
    ongoing-Authorization:
      (p.hospitalized = ”orthopedics department”) AND
      (o.patientConsent = TRUE) AND
      (o.openedBy = s.id)

  policy-2:
    target:
      (action = ”read(s, p, o)”)
    pre-Authorization:
      (”orthopedist” ∈ s.role) AND
      (o.patientConsent = TRUE)
    ongoing-Authorization:
      (o.patientConsent = TRUE)
```

Table 2: Security policy for *Example 2*

Table 2, instead, encodes the Usage Control policy related to *example 2* of Section 2. The target of this policy set is the document where the policy itself is embedded, (whose original document id is ”sd4n68k”), and the policy set consists of two policies, policy-1 and policy-2. The target of policy-1 is the read action. The pre-Authorization section includes two predicates which require that the subject who performs the action holds the role ”nurse” and works in the department of orthopedics by checking the value of the subject attributes s.role and s.department which, obviously, represents the role of the subject and the department of the hospital the subject belongs to. Since these attributes are static, their values are controlled in the pre-Authorization section only. In the scenario depicted in *example 2* the document producer

wants to allow the nurses of the department of orthopedics to read the document only when the patient is hospitalized in that department. This is implemented by exploiting an attribute of the patient, called hospitalized, whose value represents the name of department of the hospital where the patient is hospitalized (or null otherwise). This attribute is mutable, because its value changes when the patient is hospitalized and when he is discharged from hospital. Hence, the value of the hospitalized attribute of the patient is compared with the id of the department of orthopedic (which is ”orthopedics department” in our example) both in the pre-Authorization section, in order to open the document only if the related patient is currently hospitalized in that department, and in the ongoing-Authorization section, in order to revoke the read permission as soon as the patient leaves that department. A further predicate concerns the consent of the patient on the document. In particular, any access to the document is allowed as long as the patient gives his consent. The patient's consent related to a document is represented by a mutable attribute of the document itself, called ”patientConsent”. The value of this attribute changes every time the patient accesses to the web interface provided by the e-health organization to manage his e-health documents and changes the related consent preferences. In this paper, we suppose that the patient consent is a boolean attribute, i.e., the patient either gives or not the consent to access the document to all the subject of the scenario. Alternatively, the patient consent could be a list of people to which the patient wants (or does not want) to give the read permission on that document. Hence, policy-1 includes the predicate checking whether the patient consent is given or not both in the pre-Authorization and in the ongoing-Authorization sections. In this way, if the patient consent is revoked when a nurse is reading the document, the read operation is interrupted. The last predicate of the ongoing-Authorization section checks the id of the last subject who opened the document, because only one nurse at the same time can read the document. Hence if the last nurse who opened the document is not the subject of this request, the access must be interrupted. The attribute representing the id of the last subject who opened the document is mutable, because the policy include a pre-Update that changes its value when a new subject performs a read action on the same document.

Finally, the target of Policy-2 is the read action performed by subjects whose role is ”orthopedist”. Like in the previous case, the policy controls that the patient's consent is given and remains valid while the access is executed.

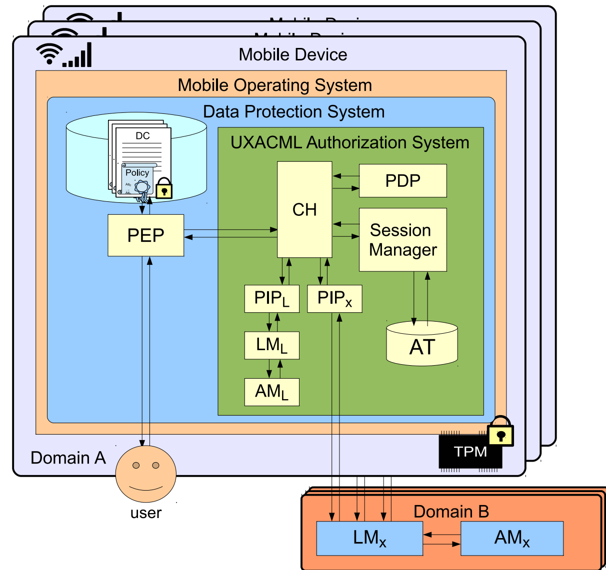

Figure 2: Usage Control Architecture

6 System Architecture

Figure 2 shows the architecture of the proposed framework, which consists of two main components, the Data Protection System and the UXACML Authorization System, which are deployed on the mobile device,

The Data Protection System (DPS) is the component which allows the user to access the DCs stored on his mobile device. In order to ensure that the Usage Control policy is always enforced when the local DCs are accessed, the DPS embeds the Policy Enforcement Point (PEP) in a way such that all the accesses are controlled by the UXACML Authorization System, and the DPS is the only component that is able to access the DCs. In fact, the DCs stored on the mobile devices are encrypted, and only the DPS keeps decryption keys to access the data and perform decryption should the access be granted to the requester. The approach adopted for distributing the keys to mobile

devices is out of the scope of this paper. In fact, the proposed framework relies on the key distribution approach defined within the Coco Cloud project, of which it is a part of.

The policy enforcement point (PEP) is the component which intercepts invocations of security-relevant access requests, suspends them before starting, queries the UXACML Authorization System for access decisions, enforces obtained decisions by resuming the suspended requests in case of positive answer or by skipping the execution of the requests in case of negative answer. The PEP also interrupts ongoing accesses as soon as it is notified by the UXACML Authorization System due to a policy violation.

The UXACML Authorization System extends the XACML reference architecture [42], and it is composed by the following components:

- Context Handler (CH) is the front-end of the UXACML Authorization System, and it manages the communication protocol with the PEP. It coordinates the other components of the UXACML Authorization System and manages the communications with them for the execution of the decision processes;

- Policy Decision Point (PDP) is a component which evaluates security policies and produces the access decision. In our framework the PDP evaluates standard XACML policies because the Usage Control specific features are managed by the Session Manager and by the Access Table;

- Session Manager (SM) is the component which is responsible for keeping track of the ongoing usage sessions to allow the continuous enforcement of the policy;

- Access Table (AT) is the component which actually stores the meta-data regarding ongoing sessions. It manages a table where each entry represents an ongoing session, and records the session ID, the access request, the policy, the session status (i.e., pending, active), the list of attributes required for the policy evaluation, and other information;

- Attribute Managers (AMs) are components which manage attributes, allowing to retrieve and to update their current values. They could be local, i.e., they run on the mobile device, or external (remote), i.e., they could run on external servers that could be even located in other administrative domains;

- Policy Information Points (PIPs) are the components which provide the interfaces to query the AMs for retrieving and updating attributes. In general, the UXACML Authorization System needs to interact with several AMs for the evaluation of the policy. In fact, each scenario requires a distinct set of AMs to manage the set of attributes required for the evaluation of the policy. Hence, our system defines a configurable chain of PIPs, where each PIP is implemented to deal with a specific AM. In particular, each PIP implements the specific protocol required to interact with the related AM and exploits the provided mechanisms for securing the communications and/or verifying attribute assertions. For instance, if the mutual authentication between the AM and the UXACML Authorization System is supported, the PIP holds the required credentials and performs the authentication process. For instance, Figure 2 shows two PIPs: PIP_L, which manages local attributes stored on the mobile device, and PIP_X, which collects and updates a set of remote attributes shared among several authorization systems. Moreover, each PIP is also in charge of triggering the policy reevaluation when the value of an attribute has changed. To this aim, the PIP could rely on the subscription mechanism provided by the AM or, if the AM does not support it, the PIP must emulate the subscription mechanism, for instance, by periodically retrieving the current attribute value in order to detect whether it is different from the one previously collected. Finally, the PIPs are also in charge to support other features concerning the attribute retrieval phase. For instance, the PIPs should implement proper strategies to allow the policy evaluation even when AMs are not reachable.;

- Lock manager (LM) is a component which guarantees consistency in concurrent retrieval and updating of mutable attributes. It supports a locking mechanism which determines whether the attribute query/update should be served or should be delayed by placing it in a queue and executing it later.

With respect to the XACML reference architecture, our

framework introduces two additional components: the SM and the AT. We prefer to keep them as separate components, and not to embed them within the PDP, because their task is to determine which ongoing sessions must be reevaluated and which policies must be used, and they are not involved in the evaluation of such policies.

The architecture has been designed to deal with concurrent retrieval and update of mutable attributes, because distinct decision processes can read and update the same mutable attributes concurrently. For instance, a common set of remote attributes are exploited by the UXACML Authorization Systems installed on two distinct mobile devices to evaluate the policies of two copies of the same document. Hence, the CH exploits a strategy for retrieving mutable attributes which prevents concurrency issues such as race conditions. This strategy implies that the attributes are collected in the predefined order (e.g., attributes are ordered by name alphabetically), and the acquisition of a lock on the attribute is required before reading it. This lock is released after the attribute update. In theory of concurrency control, this strategy is usually referred as a dead-lock free two-phase locking algorithm. However, we don't focus on this aspect in this paper. For further details please refer to [47].

6.1 Workflow

This section describes the interactions among the components of the architecture previously described when a DC is accessed on the mobile device.

Figure 3 shows the workflow of the pre-Decision phase. When the DU tries to perform a security-relevant action on a DC through the DPS, the PEP intercepts the access request R and: (1) Extracts from the DC the Usage Control policy and the static attributes embedded into the document, $attrs_e$, and (2) sends the *tryaccess* request, which includes R, $attr_e$ and the Usage Control policy, to the CH; (3) The CH determines the attributes needed for the evaluation of the Usage Control policy and, according to the previously described strategy, initiates the attribute retrieval phase. (4) At first, the CH retrieves local attributes. Since the PIP_L, LM_L, and AM_L are installed on the local device, in Figure 3 they are aggregated into a single component, and the concurrency control for local attributes is simplified, as we assume that they can be considered as a single attribute. Hence, the CH sends the attribute query to this component. When the exclusive lock is obtained, this component returns the values of the attributes to the CH. The lock will be released at the end of the decision process; (5) Then, the CH retrieves the remote attributes. For each remote attribute $attr_x$, the CH sends the attribute query to the corresponding PIP_x. The attribute query is converted in the format supported by the AM_x, and sent to the LM_x. If attr_x is not locked, the LM_x set a lock on it and forwards the attribute query to the AM_x. The attribute value returned by the AM_x is forwarded to the CH. The lock is not released. Instead, if attr_x is already locked by another process, the LM_x delays the execution of the attribute query waiting that this lock is released; (6) When the CH has collected all the required attributes, it sends R, the collected attributes, and the Usage Control policy to the PDP for the access evaluation (*pre-request*). The PDP returns the access decision and the attribute updates; (7) The CH sends the update and unlock messages for all the attributes involved in the decision process to the related PIPs, in order to update their values on the AMs and to unlock them. The order of performing updates and unlocking attributes is not important in the two-phase locking protocol. Since remote attributes are still locked because of step 5, each LM_x asks the related AM_x to perform the update and then releases the lock. The CH performs the same procedure for local attributes. (8) When all attributes are updated, if the access decision is "permit", the CH sends the *create entry* message to the SM for creating a new entry in the AT that represents the new usage session; (9) Finally, the CH replies with the access decision to the PEP.

Figure 4 shows the workflow of the ongoing-Decision phase: (1) When the access to the data object has began (e.g., the DPS app is displaying the DC on the screen of the mobile device), the PEP sends to the CH the *startaccess* message with the id of the session created during the pre-Decision phase (sID); (2) The CH contacts the SM to change the status field of the database entry related to the session to "active", and the SM replies with the policy and the list of attributes needed for the access reevaluation; (3) The CH performs the first evaluation of the ongoing policy for the session. The workflow is very similar to one of the pre-Decision phase. Indeed, the diagram "OngoingDecisionFirst" is similar to the steps 4-7 of the diagram "PreDecision" (Figure 3). The only difference is that the CH also performs the attribute subscription in steps 4-5.

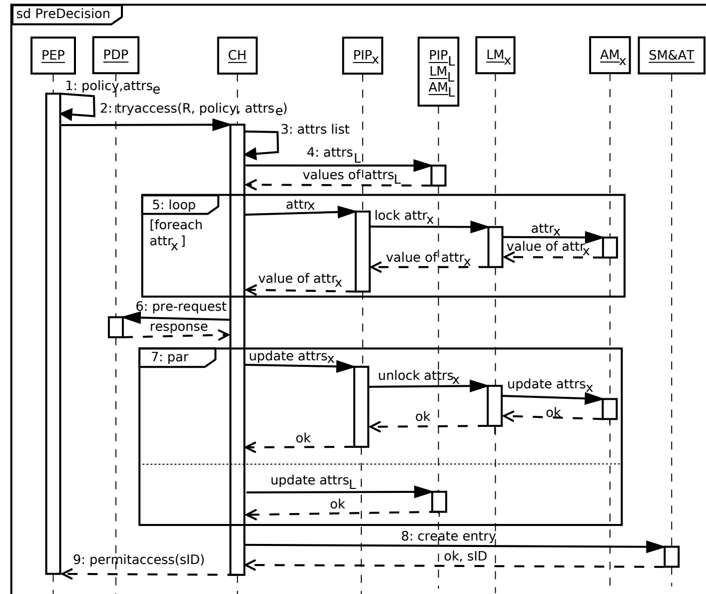

Figure 3: Sequence Diagram of the Pre-Decision phase

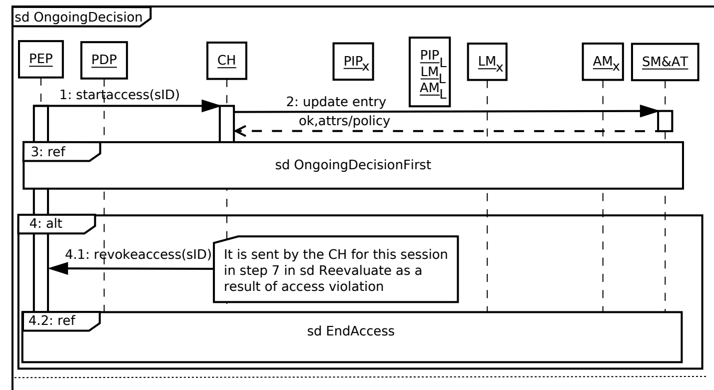

Figure 4: Sequence Diagram of the Ongoing-Decision phase

If the result of the first evaluation of the policy is “deny”, the CH cancels the attribute subscriptions and sends the *revokeaccess* message to the PEP. In Figure 4, we assume that the result of the first evaluation of the policy is “permit”; (4) From this moment on, the continuous control phase starts for this session, i.e., the policy is reevaluated every time an attribute changes its value (Figure 5) until either *i)* the policy is violated and the session is revoked (the PEP receives the *revokeaccess* message for this session), or *ii)* the user explicitly terminates the session, e.g., closing the DPS app (Figure 6).

Figure 5 shows the workflow of the reevaluation of the policies for all active sessions. (1) When the value of one subscribed attribute changes, the CH receives a notifica-

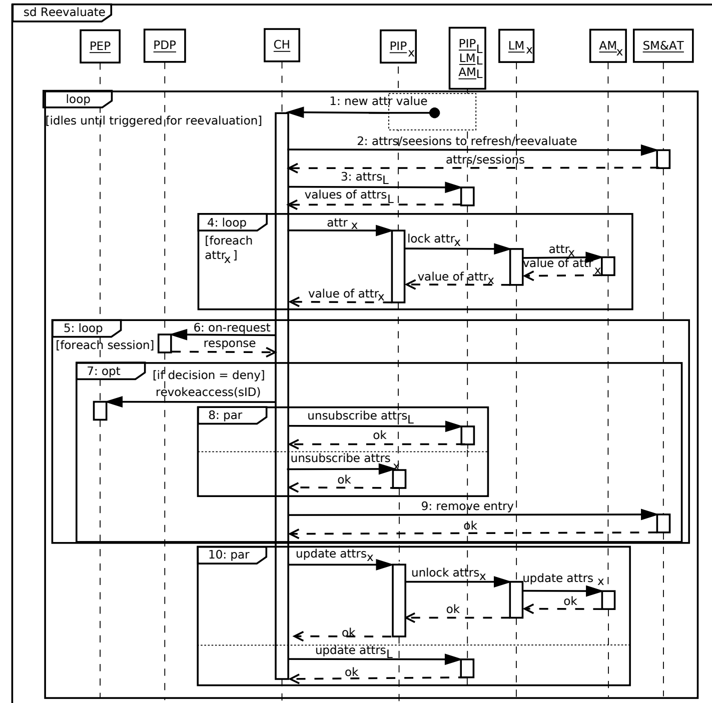

Figure 5: Sequence Diagram of the Policies Reevaluation

tion from that PIP and this triggers the reevaluation of the policies for all the active sessions involving such attribute; (2) The CH requests to the SM the list of the active sessions which must be reevaluated and the corresponding lists of attributes required for the evaluation. (3-4) The CH retrieves the current values of all the local and remote attributes in the lists and locks them in the same way as in the pre-Decision phase. From now on, the CH will use its local cache for managing the values of the collected attributes; (5) For each active session AS, the CH performs the policy reevaluation. The order chosen to reevaluate the sessions could affect decision processes, but we don't focus on this issue in this paper; (6) The CH sends the original access request R, the collected attributes and the Usage Control policy of AS to the PDP; (7-9) The CH manages the response received from the PDP, and if the access decision is "deny" it: *i)* sends the *revokeaccess* message to the PEP; *ii)* cancels the attribute subscription related to the session AS; and *iii)* asks the SM to remove the entry related to AS in the AT; (10) Finally, when all the sessions have been reevaluated, the CH sends the update messages for all the attributes to the related PIPs, in order to transfer the updated values from its local cache to the related AMs and to release the attribute locks.

Figure 6 shows the workflow in the case of a normal end of access: (1) When the PEP detects the normal end of an ongoing access (e.g., the user terminates the DPS app), it sends the *endaccess* message to the CH; (2) The CH sends the *remove entry* message to the SM, which removes the entry related to this session from the AT and replies to the CH with the metadata of the session, i.e., its policy and the attribute list; (3) For each attribute of this list, the CH send the *unsubscribe* message to the related PIP to stop getting new attribute values for this session. The remaining steps of the workflow (4-7) are similar to steps 4-7 of the pre-Decision phase, and are meat to perform the post-Updates of attributes.

The previous sequence diagrams clearly show that the

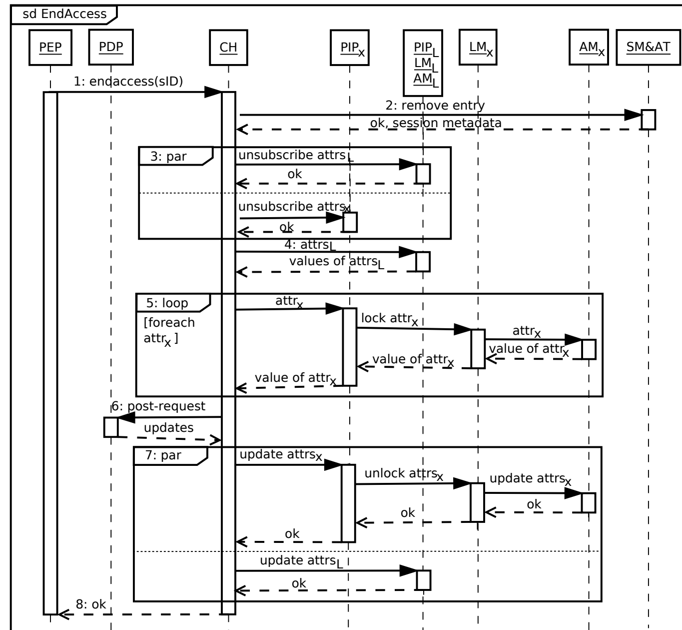

Figure 6: Sequence Diagram of the End of Access

PIPs are in charge of managing all the interactions with the remote AMs. Hence, the PIPs must implement proper strategies to allow the policy evaluation even when an AM is not reachable (e.g., it is down, the network connection is not available, or the DU cuts off the network connection of the mobile device on purpose), as shown in Section 7.3.

7 Prototype

7.1 Implementation

The implementation of the proposed framework consists of two Android applications: the UXACML Authorization app and the DPS app, as shown in Figure 7.

The **UXACML Authorization app** is the core of our prototype, and it implements the UXACML Authorization System. This application consists of a set of Android services each implementing a specific component of the architecture.

The front-end of the UXACML 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. 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 given. The Android IPC mechanism allows the CH to verify the identity of the PEP, i.e., the DPS app connecting to the UXACML Authorization app. Moreover, the UXACML Authorization app is configured to accept only IPC invocations from the applications signed with the same key and this forbids other applications but the DPS app to invoke the CH.

The CH may accept several access requests 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 able to handle multiple calls (access requests) at the same time.

The PDP is implemented through a modified version of OW2 Balana XACML Engine in order to make it running

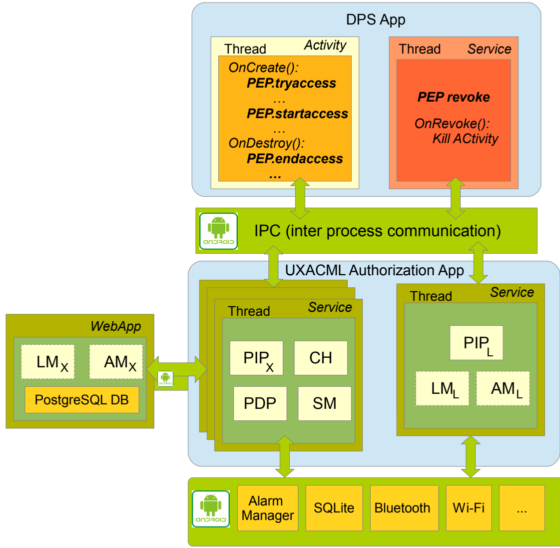

Figure 7: Usage Control Implementation in Android

on Android.

The Session Manager is an Android service which exploits the AT to store the metadata regarding the new usage session, i.e., the session ID, the access request, the policy, the session status, the list of attributes required for the policy evaluation, and other information. The AT is implemented by an Android SQLite DB installed on the device. The data about sessions are private data of the UXACML Authorization App and can not be accessed by other Android apps. Access to the AT is implemented to be thread-safe.

In our prototype the local PIP, PIP$_L$, provides interfaces to some environmental attributes (e.g, the status of the wifi and bluetooth connections) and to the location of the device (and, consequently, of the user). The local PIP also manages the attributes related to local data copies (e.g., a number of accesses to the DC stored on the mobile device), which are stored by the AM$_L$ in the SQLite DB installed on the device. The LM$_L$ guarantees the consistent access to these attributes when multiple attribute queries and updates are performed concurrently.

Remote attributes, e.g., the overall number of DC created from the same original document in the system, are managed by another PIP, PIP$_X$, which communicates via protected channel (TLS) with the remote LM$_X$/AM$_X$. We used the implementation of the remote LM$_X$/AM$_X$ described in [47]. Remote attributes are stored in PostgreSQL DB and are exposed via a web-service interface. The PostgreSQL DB is very powerful on supporting a variety of locking mechanisms and we exploit a dead-lock free two-phase locking algorithms for collecting remote attributes.

PIPs are also in charge of notifying the CH when the value of a subscribed attribute changes. For local attributes, the PIP$_L$ emulates the subscription mechanism by periodically retrieving and checking the current values of such attributes in order to detect whether some changes occurred. To this aim, the PIP sets the Android Alarm Manager in order to be invoked every T seconds, where T is a configuration parameter. When at least the value of one attribute has changed, the PIP$_L$ awakes the CH and triggers the access reevaluation. In our implementation, the remote AM does not support subscription. Hence, for remote attributes, the PIP$_X$ emulates the subscription mechanism as well by periodically polling the AM$_X$ checking if there are some updates. If so, the PIP$_X$ invokes the CH in order to trigger the access reevaluation.

In some cases, a PIP could not be able to retrieve the current values of the attributes, because the related remote AM is unreachable (e.g., the AM is down or the network is not available). In real scenarios, each PIP should implement proper strategies to manage these cases, as described in Section 7.3.

The DPS app allows the access to the local copies of the shared data, by implementing a set of operations on such data. However, how these operations are implemented is not relevant for our aims, while we concentrate on the implementation of the PEP which is embedded in the DPS app.

When a user launches the DPS app and requests access to a DC, the PEP forms the XACML access request which includes the id of the DC, of the original document, of the user, and of the requested action, along with some other attributes embedded in the DC. We exploited the Google Account id to represent the user in our implementation, though, it could be changed to the identity within a scope of the DPS app provided that the DPS account is set on the mobile device.

The structure of the Activity of the DPS app which includes the PEP functionalities is depicted on top of Fig-

ure 7. 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. The PEP receives the revocation message (Android intent) and starts a new thread that handles the real access revocation. The DPS app is configured to accept only IPC invocations from the UXACML Authorization app, therefore communications between the PEP and the CH are secured and rely on the IPC mechanisms of Android.

The DPS app stores encrypted bundles of the DC in the external memory of the mobile device (SDCard) while the keys to decrypt these bundles are stored in the internal memory of the DPS app. When a user wants to display a DC, the DPS app decrypts the bundle, gets the usage control policy and sends it via the PEP to the UXACML Authorization app. If the access decision is granted, the DPS app displays the DC. We used AES algorithm for encrypting/decrypting the bundles.

7.2 Performance Analysis

To validate the proposed approach, we installed the UXACML Authorization App and the DPS app on a 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, and we tested the performance of our prototype in case of a large number of concurrently running usage sessions. Our tests were performed on our department network, i.e., the mobile device was connected to our department network through the WiFi interface, and the remote AM run on a server connected to the same network. Please note that the results obtained from our tests are obviously dependent on the features of our testbed. Moreover, since no TPM implementation is available, we cannot measure whether it affects the performance of our framework. However, the TPM would only affect performance at system start-up, when integrity check is performed.

First, we measured the overhead introduced by the pre-Decision phase, which affects the time required to access the data. Without loss of generality, we considered a scenario in which the system receives and authorizes a single request at a time. We performed our tests varying the number of attributes required by the UXACML Authorization app to perform the decision process from 1 to 100. Please notice that 100 is a quite large number of attributes.

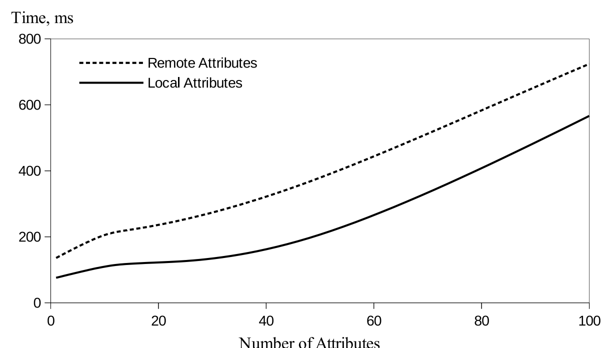

Figure 8: Overhead of Pre-Decision Phase without Updates

Figure 8 shows the UXACML Authorization app overhead in two distinct configurations; in the first configuration (represented by the continuous line) all the attributes are local, i.e., the related AMs run on the mobile device, while in the second configuration (represented by the dashed line) all the attributes are remote, i.e., the related AM run on a remote server, and network communications are required to retrieve attributes values. In this set of tests, we exploited a policy which does not perform attribute updates, i.e., the UXACML Authorization app only retrieves attribute values.

The time required to retrieve remote attributes is larger than the time required to collect local ones, although the difference is quite small. For instance, the time to evaluate a policy with 100 attributes is around 724 ms in case of remote attributes, while it is about 566 ms in case of local attributes. This difference is mainly due to the time required for the communications with the remote AM. We recall that these communications are secured through TLS, and this introduces further delay. However, we noticed that our remote AM implementation is more efficient than the implementation of the local ones, because

local AMs run on the mobile phone with limited computational resources, while the remote AM run on more powerful PC (4 GB RAM and Intel Core Duo CPU E8500 3.16GHz). Hence, in our experiments the communication delay is somehow compensated by faster responses of the remote AM.

Moreover, we noticed that there is a growth of the time required to perform the access decision with the number of attributes, as we would expect. In fact, the time required to evaluate the policy with 10 remote attributes is 205 ms, while in case of 50 attributes the time is 379 ms, and in case of 100 attributes it is 724 ms.

The next set of experiments evaluates the overhead introduced by the UXACML Authorization app in the pre-Decision phase enforcing policies that also include attribute updates. We performed two sets of experiments, in the first we exploited only local attributes, while in the second set we used only remote attributes.

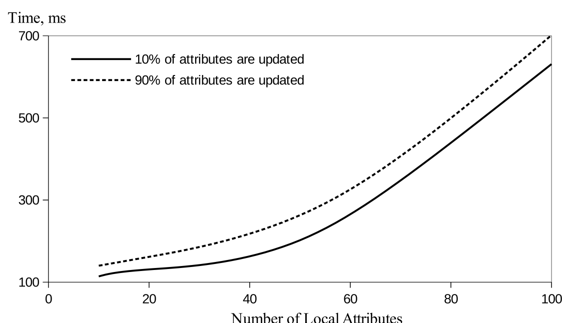

Figure 9: Overhead of Pre-Decision with Local Attributes and Updates

Figure 9 shows the results in case of local attributes. Further, we identified two cases: in one case (represented by the continuous line in the graph) the policy updates 10% of the attributes involved in the access evaluation (hence the policy that includes 10 attributes updates only one of them, while the policy including 100 attributes updates 10 attributes), while in the other case (represented by the dashed line) the policy updates 90% of its attributes. We notice that the time to perform the pre-Decision phase in case of policy that updates 10% of its attributes is, on average, about 87% of the time required when the policy updates 90% of its attributes. In fact, in case of 10 attributes, the time required to evaluate a policy that updates 1 attribute is 114 ms, while the time for a policy that updates 9 attributes is 140 ms, while in case of 100 attributes the time is 631 ms when 10 attributes are updated and 702 ms when 90 attributes are updated. Hence, we conclude that the update of local attributes does not heavily affect the pre-Decision time. As observed in the previous graph, the authorization overhead grows with the number of attributes.

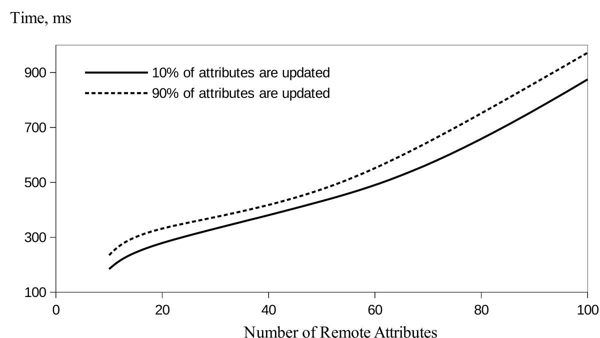

Figure 10: Overhead of Pre-Decision with Remote Attributes and Updates

Figure 10 reports the results of the experiments evaluating the UXACML Authorization app overhead enforcing policies exploiting remote attributes with updates. Like in the previous experiments we performed our tests taking into account two cases: in one case (represented by the continuous line in the graph) the policy updated 10% of the attributes, while in the other case (represented by the dashed line) the policy updated 90% of its attributes. The evaluation time of the policy that updates 10% of its attributes is, on average, about 89% of the time required when the policy updates 90% of its attributes. In fact, in case of 100 attributes, the evaluation time of the policy that updates 10 attributes is 875 ms, while the time for evaluating the policy that updates 90 attributes is 972 ms. These results show that the behaviour of the system in case of policies including remote attribute with updates is similar to the one in the scenario where all attributes are local. If we compare the time required to evaluate the policies with local attributes with the time needed for policies with remote attributes we find that on average the former is about 70% of the latter.

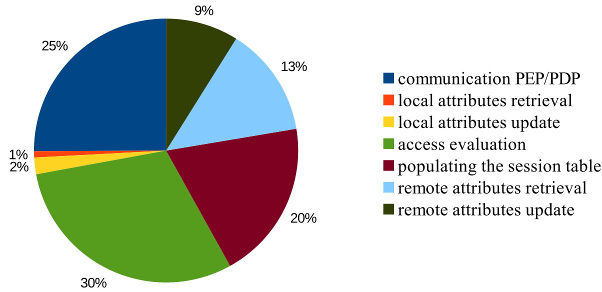

Figure 11: Factors Impacted on Pre-Decision Overhead

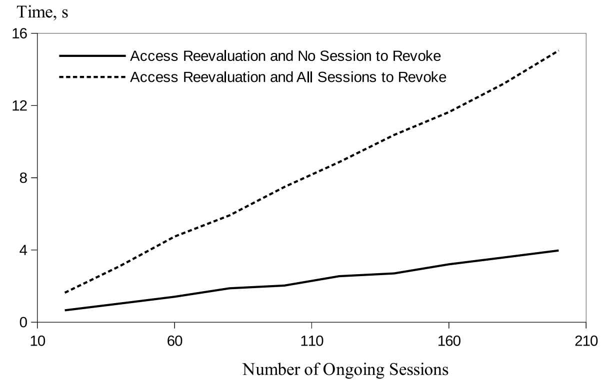

Figure 12: Overhead of Ongoing-Decision Phase

We also evaluated how each phase of the decision process impacted on the overall pre-Decision phase overhead. In general, the overhead compounds of the times that are required for sending access request/response between the PEP and the CH, retrieving local and remote attributes, evaluating the security policy, performing updates of local and remote attributes as a result of the access evaluation, populating the session table for ongoing sessions which should be controlled continuously. Figure 11 shows the results of a scenario where the policy evaluated 45 local attributes and 45 remote attributes, and updated 15 local attributes and 15 remote ones. The overall overhead is 865 ms, and 25% of it concerns the retrieving/updating local and remote attributes while 30% goes for the access evaluation. Another factor that significantly affects the pre-Decision phase overhead is the management of the data structures keeping information about ongoing sessions ("populating the session table"). In our experiment it took around 20% of the total overhead. Please, notice that this is just an example, in fact, choosing a different number of local and remote attributes would lead to different results.

The next set of experiments, whose results are shown in Figure 12, is aimed at measuring the computational load in the ongoing-Decision phase due to the continuous policy reevaluation. Varying the number of ongoing sessions (i.e., accesses which are in progress) from 20 to 200, we measured the time required to reevaluate the usage control policy and to revoke the ongoing session in case the response of the policy reevaluation is "deny". All sessions used 20 local attributes which were queried once but all sessions were reevaluated independently and sequentially.

There was no update of attributes as a result of the access reevaluation. Figure 12 (represented by the continuous line) shows the results obtained. We see that the time needed for reevaluation of sessions grows linearly in the number of running sessions. In particular, for 20 ongoing sessions (e.g., 20 documents opened at the same time on the mobile device) the time is about 0,6 seconds, while for 200 ongoing sessions it is about 4 seconds. In fact, the results show that the mobile device is powerful enough 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 reevaluation until PEPs receive revocation messages for all sessions that have to be revoked. Figure 12 (represented by the dashed line) shows the results obtained. We see that the revocation time of all sessions grows linearly in the number of running session and for revoking 20 sessions we need about 1,6 seconds, while 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 reevaluation 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 the mobile device and consumes CPU, battery, and memory. In case of 100

ongoing usage sessions and when the access reevaluation is triggered every 30 seconds, the UXACML Authorization App takes about 1.4% of the CPU time and needs approximately 30 MB of RAM.

7.3 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 Trusted Platform Module (TPM). We recall that, in this paper, we assume that the device owner is a different entity from the device user. The owner gives his device to the user with the DPS app installed to control the access and usage to specific data. As an example, the device owner could be the employer of a company which gives business smartphones to his employees, to allow them to remotely access specific documents.

The proposed system prevents the device user and other applications from directly accessing the DCs downloaded on the device. As a matter of fact, the DCs stored on the device can be accessed only through our Data Protection System, which provides a restricted set of actions and ensures the enforcement of the related Usage Control policies. In our threat model we consider two possible attackers which, having control on the device may try to access the data bypassing the policy enforcement mechanisms, thus violating data *confidentiality*. First, we consider the device *user*, attempting to read stored data on the device without receiving authorization from the DPS app, thus against the policy. Secondly, we consider the possibility of an *external attacker* which remotely controls the device (or at least a limited number of functionalities) through a malicious app which is installed on the device. Generally, the user is not aware of this intrusion since the malicious app often comes as a trojanized one, i.e., with the malicious code running in the background of an app showing a genuine behavior.

We consider the following three models to perform an attack (also shown in Figure 13):

1) Users/External attackers trying to access data stored in the SDCard (external storage). Users can browse the SDCard contents through a file manager application.

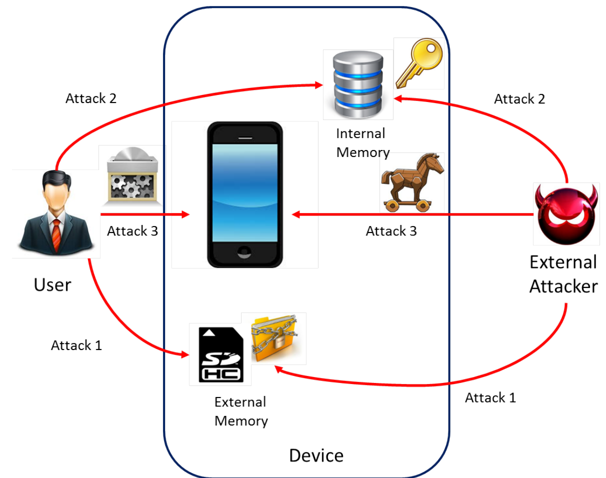

Figure 13: Envisioned attack models.

2) Users/External attackers trying to access encryption keys used to encrypt data stored on the SDCard.

3) Users/External attackers 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. The files including the DCs to be protected are stored in the mobile device external memory, i.e., in the SDCard, and they are encrypted through the AES algorithm. This protects from attackers of type 1) who will find encrypted files (and not usable) when accessing the SDCard of the device. We assume that the related secret key has been stored in the device internal memory, in the home space of the DPS app. Differently from the external memory, this space can only be accessed by the DPS app, so that encrypted files can be decrypted only by the Data Protection System. This security mechanism protects the key from attackers of type 2). Thanks to the inclusion of trusted computing, it is possible to ensure that only the DPS app can access the secret key. In fact, as soon as the system integrity verification performed by the TPM fails, the DPS app and all the related files, residing in both the internal and external memory, are removed from the device. Moreover, the secret key will no longer be accessible as soon as the device has been rooted or unknown dangerous applications have been installed on it. Hence, the service provider is

notified of the fact that the device is *not in a secure status* anymore. Furthermore, we assume that the secret key will not be released anymore unless the user can prove the device status is secure again, e.g. through a factory reset. This ensures the protection also from an attacker of type 3).

It is worth noting that we are not considering Man In The Middle (MITM) Attacks in our analysis, since we assume that all channels are secured. In particular interference of another app in the normal behavior of DPS app or UXACML Authorization App are not possible thanks to the isolation enforced by Android, exploiting intents, binders and IPC communication to create secure channels which ensures end-to-end encryption and integrity. Furthermore, in our reference implementation the network communications between the PIP and the AM are secured through the use of TLS, which ensures mutual authentication, confidentiality and data integrity, to be able to tackle, among others, the MITM attack. We recall, however, that the security of the connection between the PIP and a remote AM depends on the mechanisms that the AM is offering. In case a remote AM does not implement a TLS mechanism, and it is not possible to verify the author and the integrity of the attribute assertion through other mechanisms, the retrieved attribute is considered not trusted, hence it could be decided not to be used for the policy evaluation. Such a decision is dependent on the specific scenario.

A further threat to the confidentiality of the DCs is represented by malicious device users who have the right to access the DC and could try to copy pieces of data from a DC to another document or another application of the mobile device to avoid the enforcement of the (right) Usage Control policy on those data. Our framework prevents this attack because DCs can be accessed through the DPS app only, which forbids to copy or cut pieces of data from DCs.

For what concerns DC *integrity*, the file is always encrypted together with a digest of the original file. After decryption, DPS app verifies if the digest corresponds to the hash of the decrypted file (standard integrity check [48]) and, if not, the file is deleted and downloaded fresh, given that the user has still the right to access it. Thanks to the hash function properties, given that both the file and the digest are encrypted, it is not possible to alter the DC integrity without being detected through DPS app.

Moreover, once a DC has been downloaded on the mobile device, the *availability* of the related data to the DU should be guaranteed by the DPS according to the Usage Control policy embedded in DC itself. As already discussed, sometimes one or more AMs could be unreachable (e.g., they could be down or the network could be unavailable) and hence the related attributes can not be retrieved. On one hand, the missing attributes could cause a wrong policy evaluation. Consequently, the data could be made available to the DU although he actually does not hold the right to access them, or the data could not be made available although the DU holds the related right. On the other hand, the missing attribute updates could prevent the detection of a policy violation, thus allowing the prosecution of an existing access to the data even if the DU actually does not hold the related right any more. In some scenarios, this issue can be mitigated by implementing proper attribute retrieval strategies within the PIPs. For instance, in a scenario where multiple instances of the same AM are available, when one of them is unreachable, the PIP should try to interact with the others. Obviously, an important requirement in this case is that the consistency among the AM instances is properly mantained. A more general solution to the AM availability problem is one where each PIP provides an environmental attribute, called e.AM_XNotAvailable, representing how long the connection with the related AM, AM_X, is not available. This attribute enables the DP to explicitly define, in the Usage Control policy, proper rules for regulating the usage of the DC when an AM is not available, depending on the requirements of the specific use case. This solution also addresses the case where the DU cuts off the network connection of the mobile device on purpose, in order to prevent the DPS from communicating with the AMs. In fact, the DU could disconnect the mobile device from the network after the DPS granted him the right to access a DC, in order to prevent the DPS from receiving attribute updates which could invalidate this right. The DP can mitigate this attack by defining a Usage Control policy which allows the usage of the DC when the AMs are unreachable for a few minutes only. For instance, in *example 1*, the team manager could tolerate that an AM is unreachable for no more than a couple of minutes. In fact, *example 1* requires that the DU is located within the building of the company, where the network is supposed to be always available, and the remote AM is supposed

to be always available as well, because it is located within the company network and managed by the company itself.

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 internal status of the UXACML Authorization app, e.g., the set of 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 app, 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.

7.3.1 Decryption key protection

In Android, every application has a private space in the device internal memory, which is protected by the underlying Linux kernel. More specifically, as part of the isolation protection mechanism [49], Android creates a new Linux user for each app which is installed on the device. This user will receive an home folder, which could be used by the application itself to safely store data. In fact, the app of the space is the only one to have reading, writing and execution permission on his home directory. Thus, this private space constitutes a good candidate to store the decryption key for data Usage Control files.

The only way the user of the mobile device or another application can access such a private space is that such an application has *superuser* permissions. However, the Android's Linux kernel is a stripped version of the classical "vanilla" Linux, which, removing the switch user (`su`) command, does not allow any application to get superuser privileges. Also other commands used to modify authorizations such as `chgrp` and `chmod` have been removed in Android. Notwithstanding, some users can still try to get the super user privileges on their devices through specific applications, i.e. *rootkit*, which reintroduce the missing command through a buffer-overflow attack. This procedure, named rooting or jailbreaking modifies the Linux kernel violating its integrity.

As discussed, such an integrity check can be performed verifying the hash signature of the Linux kernel through TCB. In particular, as aforementioned in Section 4.2, the ARM's `TrustZone` offers a TCB functionalities which is natively embedded in almost any Android device. In particular, the presented framework exploits the `Secure Boot` functionality to avoid any firmware and OS modifications which may compromise the security of protected data. Through `Secure Boot`, the hash (digest) of each level of the current device status. More specifically, the data service provider uses its Private Key (PrK) to generate a signature of the code that they want to deploy, and pushes this to the device alongside the software binary. The device contains the Public Key (PuK) of the service provider, which can be used to verify that the system has not been modified and that it was provided by the service provider. Before storing the hash digest in the TPM the service provider should verify that the telephone is in a safe status, checking that has not been rooted beforehand. Such a check can be easily performed verifying either that the `sys` partition has not been modified (it should be read-only partition) or that it is not possible to invoke by command line the `su` command. Also, if the user wants that a new device status is saved in the TrustZone chip, must physically bring the device to the service provider.

The secure boot verification avoids the possibility of data encryption key exposure due to jailbreaking (rooting) of the Android device. In fact, if the system is found to be not in a secure state, both protected data and encryption key are removed from the device, whilst the service provider is notified that the device is not in a secure state. This policy is aimed at preserving protected data from unauthorized accesses of potentially malicious users, which encompasses the two aforementioned different attackers: *i)* the device user attempting to maliciously access protected data stored on her device, when she is not authorized to do it. *ii)* An external attacker infects through a malicious application (malware) a user device in order to access the protected data, jailbreaking the device through a rootkit and then stealing the encryption key. Whilst for *i)* the user is intentionally violating the terms of service of the data provider, in *ii)*, communicating to the service provider that the device is not in a safe state anymore, could be the first step on noticing and solving a security threat which may potentially affect other device resources. In both cases, communicating the current not-safe status of the device should be considered as a necessary step to protect data, which belongs to the ser-

vice provider and not to the user. However, regardless from the attacker, attempting to access encrypted data has to be considered as a general violation of the terms of services, intentional in the first case and unintentional in the second one, still with user's responsibility who failed in protecting the device.

Since currently no off the shelf Android devices including TPM are available, we exploit a system similar to the one presented in [50], where the two main functions of TPM are implemented as a Linux module called micro-TPM. This module can be seen like an interface to a real TPM and could be used at its full extense as soon as commercial TPM-enabled Android devices are released. We recall that some works have already been done in this direction, joining Android with TPM on non-commercial devices [34] [35]. Furthermore, it is worth noting that current commercial Android device already include the TrustZone technology, which is embedded in ARM CPUs. However, TrustZone is currently not enabled on commercial devices, since device manufacturer do not provide the interfaces to interact with this component.

8 Conclusions

This paper presented a framework for regulating the usage of data which are shared on mobile devices. In our reference scenario, the data producer embeds a Usage Control policy in the data he shares, and the data users download a copy of these data on their mobile devices. Our framework enforces the related Usage Control policy every time each data user performs an access to the local copy of the shared data on his mobile device, in order to prevent unauthorized usage of the data itself. The main feature is that the Usage Control paradigm, besides regulating the right of initiate an access, also check that the right of using the data continuously holds while the access is in progress. To this aim, our framework is able to interrupt an access in progress as soon as the related policy is no longer satisfied.

The paper presented the detailed architecture of the proposed framework, its implementation on Android mobile devices, and a set of experiments aimed at evaluating the time required for the authorization process. The experiments show that the delay introduced by the authorization framework in the time required to access the data is quite small, and that the time required to perform the ongoing evaluation of the policy and the revocation of the accesses is again small for today's mobile devices in real scenarios.

Our prototype exploits both the Android native security mechanism and a TPM to ensure system integrity. However, the use of a TPM 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.

Finally, we envisage some extensions of our framework for future work, such as the support for managing the copy of pieces of data among DCs, which requires the capability of enforcing the right Usage Control policy on each piece of data when these data pieces are copied from one DC to another, the support for managing the accountability of the policy makers, as well as an implementation on an Android device equipped with a TCB.

References

[1] Lazouski, A., Martinelli, F., Mori, P., Saracino, A.: Stateful usage control for android mobile devices. In: Proceedings of the 10th International Workshop on Security and Trust Management (STM 2014). Volume 8743 of Lecture Notes in Computer Science., Springer Berlin Heidelberg (2014) 97–112

[2] Caimi, C., Gambardella, C., Manea, M., Petrocchi, M., Stella, D.: Technical and legal perspectives in data sharing agreements definition. In: Proceedings of Annual Privacy Forum, APF 2015. Volume 9484 of Lecture Notes in Computer Science., Springer Berlin Heidelberg (2015) 178–192

[3] 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. Springer Berlin Heidelberg (2013) 775–792

[4] Kelbert, F., Pretschner, A.: Data usage control enforcement in distributed systems. In: Third ACM Conference on Data and Application Security and

Privacy, CODASPY'13, San Antonio, TX, USA, February 18-20, 2013, ACM (2013) 71–82

[5] Kelbert, F., Pretschner, A.: A fully decentralized data usage control enforcement infrastructure. In: 13th International Conference on Applied Cryptography and Network Security (ACNS 2015). (2015) 409–430

[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] Costa, G., Martinelli, F., Mori, P., Schaefer, C., Walter, T.: Runtime monitoring for next generation java me platform. In Computers & Security **29**(1) (2010) 74–87

[9] 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

[10] 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

[11] 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. Communications of the ACM **57**(3) (2014) 99–106

[12] Heuser, S., Nadkarni, A., Enck, W., Sadeghi, A.R.: Asm: A programmable interface for extending android security. In: 23rd USENIX Security Symposium (USENIX Security 14), San Diego, CA, USENIX Association (August 2014) 1005–1019

[13] Miettinen, M., Heuser, S., Kronz, W., Sadeghi, A.R., Asokan, N.: Conxsense - context profiling and classification for context-aware access control. In: Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security (ASIACCS 2014), ACM (2014)

[14] Bugiel, S., Heuser, S., Sadeghi, A.R.: Flexible and fine-grained mandatory access control on android for diverse security and privacy policies. In: Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13), Washington, D.C., USENIX (2013) 131–146

[15] Nauman, M., Khan, S., Zhang, X.: Apex: Extending Android Permission Model and Enforcement with User-defined Runtime Constraints. In ACM, ed.: 5th ACM Symposium on Information Computer and Communication Security (ASIACCS'10). (2010) 328–332

[16] Nadkarni, A., Enck, W.: Preventing accidental data disclosure in modern operating systems. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security. CCS '13, New York, NY, USA, ACM (2013) 1029–1042

[17] Backes, M., Bugiel, S., Gerling, S., von Styp-Rekowsky, P.: Android security framework: Extensible multi-layered access control on android. In: Proceedings of the 30th Annual Computer Security Applications Conference. ACSAC '14, New York, NY, USA, ACM (2014) 46–55

[18] Martinelli, F., Mori, P., Saracino, A.: Enhancing android permission through usage control: A byod use-case. In: 31st ACM Symposium on Applied Computing (SAC 2016). (2016) 2049–2056

[19] Chuang, C.Y., Wang, Y.C., Lin, Y.B.: Digital right management and software protection on android phones. In: Vehicular Technology Conference (VTC 2010-Spring), 2010 IEEE 71st. (May 2010) 1–5

[20] Ongtang, M., Butler, K., McDaniel, P.: Porscha: Policy oriented secure content handling in android.

In: Proceedings of the 26th Annual Computer Security Applications Conference. ACSAC '10, New York, NY, USA, ACM (2010) 221–230

[21] von Styp-Rekowsky, P., Gerling, S., Backes, M., Hammer, C.: Idea: Callee-site rewriting of sealed system libraries. In: Engineering Secure Software and Systems - 5th International Symposium, ESSoS 2013, Paris, France, February 27 - March 1, 2013. Proceedings. (2013) 33–41

[22] Backes, M., Gerling, S., Hammer, C., Maffei, M., von Styp-Rekowsky, P.: Appguard - fine-grained policy enforcement for untrusted android applications. In: Data Privacy Management and Autonomous Spontaneous Security - 8th International Workshop, DPM 2013, and 6th International Workshop, SETOP 2013, Egham, UK, September 12-13, 2013, Revised Selected Papers. (2013) 213–231

[23] Backes, M., Gerling, S., Hammer, C., Maffei, M., von Styp-Rekowsky, P.: Appguard - enforcing user requirements on android apps. In: Tools and Algorithms for the Construction and Analysis of Systems - 19th International Conference, TACAS 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings. (2013) 543–548

[24] Xu, R., Saïdi, H., Anderson, R.: Aurasium: Practical policy enforcement for android applications. In: Presented as part of the 21st USENIX Security Symposium (USENIX Security 12), Bellevue, WA, USENIX (2012) 539–552

[25] 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

[26] Dragoni, N., Massacci, F., Naliuka, K., Siahaan, I.: Security-by-contract: Toward a semantics for digital signatures on mobile code. In: Public Key Infrastructure. Springer (2007) 297–312

[27] Dini, G., Martinelli, F., Matteucci, I., Saracino, A., Sgandurra, D.: Introducing probabilities in contract-based approaches for mobile application security. In: Data Privacy Management and Autonomous Spontaneous Security - 8th International Workshop, DPM 2013, and 6th International Workshop, SETOP 2013, Egham, UK, September 12-13, 2013, Revised Selected Papers. (2013) 284–299

[28] Di Cerbo, F., 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

[29] 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

[30] Colombo, M., Lazouski, A., Martinelli, F., Mori, P.: A proposal on enhancing XACML with continuous usage control features. In: proceedings of Core-GRID ERCIM Working Group Workshop on Grids, P2P and Services Computing, Springer US (2010) 133–146

[31] La Polla, M., Martinelli, F., Sgandurra, D.: A survey on security for mobile devices. Communications Surveys Tutorials, IEEE **15**(1) (First 2013) 446–471

[32] Trusted Computing Group: TPM 2.0 mobile reference architecture (draft) (April 2014)

[33] ARM-Ltd.: Building a secure system using trustzone technology (april 2009) Avaiable at: http://goo.gl/9p7SWG.

[34] Samsung-Electronics-Co-Ltd.: An overview of samsung knox (Jun 2013)

[35] Li, X., Hu, H., Bai, G., Jia, Y., Liang, Z., Saxena, P.: Droidvault: A trusted data vault for android devices. In: Engineering of Complex Computer Systems (ICECCS), 2014 19th International Conference on. (Aug 2014) 29–38

[36] 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

[37] Park, J., Sandhu, R.: The $UCON_{ABC}$ usage control model. ACM Transactions on Information and System Security **7**(1) (2004) 128–174

[38] 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

[39] Pretschner, A., Hilty, M., Basin, D.A.: Distributed usage control. Communications of the ACM **49**(9) (2006) 39–44

[40] Park, J., Zhang, X., Sandhu, R.S.: Attribute mutability in usage control. In: Research Directions in Data and Applications Security XVIII, IFIP TC11/WG 11.3 Eighteenth Annual Conference on Data and Applications Security. (2004) 15–29

[41] 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

[42] OASIS: eXtensible Access Control Markup Language (XACML) version 3.0 (January 2013)

[43] Kumari, P., Pretschner, A., Peschla, J., Kuhn, J.: Distributed data usage control for web applications: a social network implementation. In: Proceedings of the First ACM Conference on Data and Application Security and Privacy, CODASPY 2011. (2011) 85–96

[44] Birnstill, P., Pretschner, A.: Enforcing privacy through usage-controlled video surveillance. In: 10th IEEE International Conference on Advanced Video and Signal Based Surveillance, AVSS 2013, Krakow, Poland, August 27-30, 2013, IEEE (2013) 318–323

[45] Martinelli, F., Mori, P.: On usage control for grid systems. Future Generation Computer Systems **26**(7) (2010) 1032–1042

[46] Bertolino, A., Daoudagh, S., Lonetti, F., Marchetti, E., Martinelli, F., Mori, P.: Testing of polpa authorization systems. Software Quality Journal **22**(2) (2014) 241–271

[47] 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

[48] Armstrong, D.: An introduction to file integrity checking on unix systems (2003)

[49] Enck, W., Ongtang, M., McDaniel, P.: Understanding android security. Security Privacy, IEEE **7**(1) (Jan 2009) 50–57

[50] Nauman, M., Khan, S., Zhang, X., Seifert, J.P.: Beyond kernel-level integrity measurement: Enabling remote attestation for the android platform. In Acquisti, A., Smith, S., Sadeghi, A.R., eds.: Trust and Trustworthy Computing. Volume 6101 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2010) 1–15