Adaptive Context-Aware Security for Android and IoT Smart Applications

by

Saad INSHI

THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
IN PARTIAL FULFILLMENT FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
Ph.D.

MONTREAL, AUGUST 09, 2023

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

FOREWORD

This dissertation is subjective to address different aspects in Android and IoT adaptive context-aware privacy and security policies, which have emerged as a new hot topic. This research work successfully ended with 2 published conference papers, 1 published journal paper, and 2 journal papers under review. For the sake of accuracy of the published and submitted versions, these articles are presented without modifications. Although each of the articles address different aspects, yet they are all closely interrelated. While a separate section is devoted for the literature, in each article, a specialized state of the art review is presented and analyzed based on the research problems and contributions subject to that particular work.

ACKNOWLEDGEMENTS

Countless people supported and guided me throughout this journey and without them this work would not have been possible.

I would like to express my sincere and utmost gratitude to my supervisor Prof. Chamseddine Talhi, whose expertise was the most valuable in formulating the research problems and methodologies and in conducting the thesis experiments. His level of kindness, support, patience, knowledge, and ingenuity is something I will always keep aspiring to. I would also like to thank Prof. Hakima Ould-Slimane for her collaboration in this research. Besides my advisor, I would like to thank the thesis committee members for their efforts and insightful comments.

I am grateful to my friend and best research partner, Rasel Chowdhury. Without his support and wise thoughts, this project would have not been the same.

I also express my endless gratitude to my beloved mother, to my father who have passed away during this journey, to my family and friends for their kind wishes, prayers, and continuous encouragement.

A special heartfelt thanks go to my beloved wife Najah, and to my lovely kids Noha, Mohad and Fadeel. Nobody has been more patient in the pursuit of this work than them. Thank you for the love and strength that you have provided me during this work.

This journey has been quite remarkable, and I believe it is a unique steppingstone to many of my achievements ahead ISA.

Sécurité contextuelle et adaptative pour les applications intelligentes Android et IoT

Saad INSHI

RÉSUMÉ

La notion de la sensibilité au contexte lors dû l'utilisation des applications mobiles attire de plus en plus l'attention, de nombreuses applications doivent s'adapter aux environnements physiques des utilisateurs et des appareils, tels que la localisation, l'heure, la connectivité, les ressources, etc. En outre, prédire les activités sensibles au contexte à l'aide des techniques d'apprentissage automatique évoluent chaque jour pour devenir plus facilement disponibles en tant que le moteur majeur de la croissance des applications IoT qui réponds aux besoins des futur environnements autonome intelligents. Cependant, avec les risques de sécurité croissante d'aujourd'hui dans les techniques cloud émergents, partager des capacités de données massives, imposer les exigences de la réglementation sur la confidentialité et émergence de nouvelles technologies multi-utilisateurs, multi-profils, multi-smartphones, il y a un besoin croissant des nouvelles approches pour relever les nouveaux défis de l'autonomie de la prise de conscience du contexte et leurs modèles de sécurité précis.

Récemment, de nombreux intérêts de recherche qui ont été écrits ont ciblé des différents aspects de la sensibilité au contexte à la sécurité Android et aux environnements intelligents IoT. Pourtant, aucune des œuvres existantes répond aux exigences de nos scénarios réels pour la création automatique des politiques de sécurité qui se basent sur le contexte, et le chiffrement automatisé selon ces politiques générées automatiquement. Se basant sur ces faits, cette thèse comble le manque des résultats des recherches existantes. En particulier, les apports de cette thèse sont triples: (1) Fournir un Framework des politiques de sécurité pour le contrôle des communications internes entre les applications Android afin d'atténuer les fuites de confidentialité. (2) Définir un langage de spécification des politiques de sécurité formelles sensibles au contexte qui décrit effectivement les consentements définis par les utilisateurs. (3) Fournir une nouvelle création dynamique sécurisée des politiques sensibles au contexte, réalisé en utilisant les techniques d'apprentissage intelligents qui tirent parti du chiffrement par attributs (ABE) pour le chiffrement dynamique. Divers prototypes qui ont été développés et évalués par des expériences approfondies par lesquels les résultats ont prouvé l'efficacité des solutions proposées. Finalement, la solution respecte les nouvelles réglementations imposées en matière de confidentialité et exploite toute la puissance des environnements intelligent de l'IoT.

Mots-clés: Conscient du contexte, Android, ABE, confidentialité, sécurité, apprentissage automatique, environnement intelligent

Adaptive Context-Aware Security for Android and IoT Smart Applications

Saad INSHI

ABSTRACT

The notion of Context-Awareness of mobile applications is drawing more attention, where many applications need to adapt to physical environments of users and devices, such as location, time, connectivity, resources, etc. Also, predicting context-aware activities using machine learning techniques is evolving to become more readily available as a major driver of the growth of IoT applications to match the needs of the future smart autonomous environments. However, with today's increasing security risks in emerging cloud technologies, sharing massive data capabilities, imposing regulations requirements on privacy, and the emergence of new technologies of multi-users, multi-profiles, multi-devices, there is a growing need for new approaches to address the new challenges of autonomous context-awareness and its fine-grained security enforcement models.

Recently, many research interests have been drawn targeting different aspects of Context-Awareness in Android security and in IoT smart environments. Yet, none of the existing works fulfills our real scenarios requirements for automated creation of context-aware policies, and the automated encryption according to these auto-generated policies. From these premises, this dissertation fills the lack of the existing research outcomes. Particularly, the contributions of this thesis are threefold: (1) Providing a policy enforcement framework for Inter-App communication between Android applications to mitigate privacy leakage. (2) Defining a formal context-aware policy specification language that effectively describe users defined consents. (3) Providing a novel secure dynamic creation of context aware policies, which has been achieved through smart learning techniques that leverages Attribute-Based Encryption (ABE) for dynamic encryption. Various prototypes have been developed and evaluated via extensive experiments through which the results have proved the efficiency of the proposed solutions. Finally, the solution fulfils the new imposed privacy regulations and leverages full power of IoT smart environments.

Keywords: Context-Aware, Android, ABE, Privacy, Security, Machine Learning, Smart Environment

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

IoT	Internet of Things
ML	Machine Learning
API	Application Programming Interface
APK	Android Package Kit
XACML	Extensible Access Control Markup Language
XML	Extensible Markup Language
JSON	JavaScript Object Notation
IPC	Inter-Process Communication
GUI	Graphical User Interface
ABE	Attribute-Based Encryption
KMS	Key Management System
PKI	Public Key Infrastructure
GPS	Global Positioning System
MAC	Mandatory Access Control
RBAC	Role Based Access Control
DT	Decision Tree
KNN	K-Nearest Neighbour
RF	Random Forest
NB	Naive Bayes classier

SVM	Support Vector Machines
TN	Tactical Network
C4ISR	Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance

INTRODUCTION

0.1 Motivation and Problem Statement

In today's highly competitive development of smart mobile and IoT applications, enterprises need flexibility in meeting customers complex demands by developing tens of thousands of applications and make them available on the application markets, such as Google Play and App Store. For Instance, Android has become the leading mobile platform and has dominated the market with share increase from 85.1% in 2018 to 86.6% in 2019 IDC (2020). While these apps can benefit customer needs, it may also have an undesirable effect that might occur without user's consent, namely, privacy leakage. Existing application authorization system in Android allows users to control only the permissions that are classified as dangerous. For example, a user can choose to allow a camera app to access the camera, but not to the contact information without his consent or awareness. Therefore, a flexible policy-based access control system is needed to monitor APIs functions in Android applications, especially those requiring access to the user's sensitive and crucial information.

Besides the privacy challenges, Context-awareness services is a key driver for the modern mobile operating systems which are commonly prompting users by showing authorization dialog boxes asking for allowing or denying access to some functionalities. These services opened a big interest in defining, managing, and enforcing context-aware policies especially for those scenarios that put users under the risk or leaking or misusing their credential information. Accordingly, several studies have covered many context-aware aspects Sarker *et al.* (2020) Nawrocki, Sniezynski, Kolodziej & Szynkiewicz (2020) Mshali, Lemlouma & Magoni (2018) Alkhresheh, Elgazzar & Hassanein (2018) Sarker & Salah (2019) Phung, Mohanty, Rachapalli & Sridhar (2017) Inshi, Chowdhury, Elarbi, Ould-Slimane & Talhi (2020),of which two main aspects are still challenging: The first aspect is the lack of a context-awareness broker that can manage the complex identities of interconnected sensors from different devices. In such situations, context-aware policies will play the main role in deciding what and when data needs to be processed or shared, and much more. In addition, the current smart IOT environments need such context-awareness broker as central points of control, as different middle-ware solutions developed by different parties will be employed to connect to sensors for collecting, modeling, and reasoning context information.

The second aspect is updating contents based on the context information, while the data aggregated by the sensors needed to be shared with the authorized providers or observers, consents are not fixed and can be updated not to share the aggregated date at certain contexts. For example, consent can be updated to share data when switching between devices, switching between users, or not to share data based on location and time (meeting, enterprise, home). Also, consent can be updated to share data again based on events, such as an insurance company needing to restore data to know what caused an accident. Therefore, to effectively perform this aspect, we need an intelligent dynamic adaptation of contexts, which can be achieved through smart learning techniques. In such situations, machine learning techniques will take the context information as inputs to learn models able to adapt to the consents accordingly.

Furthermore, the evolution of the 6G networks is continuously evolving to match the needs of the future smart, self-adaptive applications. Nowadays, billion of IoT devices are connected through machine-to-machine (M2M) and these interrelated devices are provided with unique identifiers and the ability to transfer data over a network with improved efficiency, accuracy, and economic benefit of the IoT environments. Therefore, future applications and business models are required to fulfill the new performance criteria, such as big data, multi-devices, limited access control, security, privacy, and regulations. Smartwatches are good examples of multi-devices that complement Smartphones with the capabilities to check time, messages, emails, notifications and many more functionalities with easier accessibility.

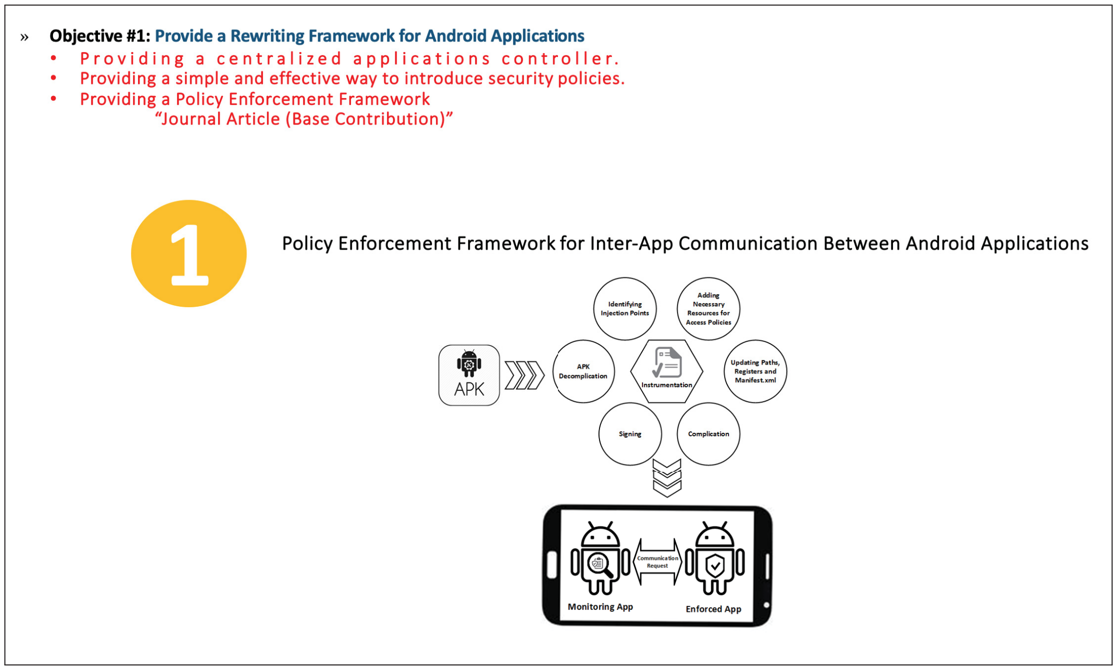
Besides all above challenges, security is a key requirement to the evolution of the new automated smart IoT environments that provide security services and identity management. This growth raises the need to protect data exchange between different entities by many ways (e.g., authorization, automated access control, and data encryption). In this scope, many encryption-based access control schemes have been proposed and adopted in IoT environments, starting with symmetric key encryption and moving to the Public Key Infrastructure (PKI) which provides data signature to ensure integrity, and session keys to ensure confidentiality. Still, PKI requires huge infrastructure (Certifying Authorities, Registration Authorities, Repository, Archives and End Entity) to manage and maintain the certificates Jancic & Warren (2004). Goyal et al Goyal, Pandey, Sahai & Waters (2006) introduced an asymmetric encryption technique called AttributeBased Encryption (ABE) which is one of the most recent access control-based encryption scheme. This technique allows the definition of fine grained access control policies for data privacy and security. Yet, none of the existing works fulfills our requirements of context-aware, smart, adaptive environment use cases.

Ultimately, smart Context Awareness Encryption is the key solution to the new secure smart systems evolution. Therefore this thesis presented a novel contributions that discloses a Context Aware, Adaptable, Intelligent and Lightweight Security Solutions in different Android and IoT smart environments.

0.2 Research Objectives

This research has a number of aims and objectives, but the main focus is to propose a novel Context-Aware, Adaptable, Intelligent and Lightweight Security Solutions for Android and IoT smart applications. Therefore, throughout this work, various research questions have arisen and hence several objectives had to be set, which are highlighted throughout the articles presented in the following chapters:

- Due to the limitation of existing Android security model where user behavior and app permissions can sometimes lead to security vulnerabilities. Also, the possibility of malicious apps slipping through Google Play Store's screening process. We have started in the first article (Chapter 2) by developing an efficient app enforcement framework as shown in Figure 0.1. In fact, control capabilities should rely on efficient interaction points (inside applications) and a centralized controller. Accordingly, this rewriting framework will force any Android applications to communicate with our centralized applications monitor before calling any sensitive API calls.

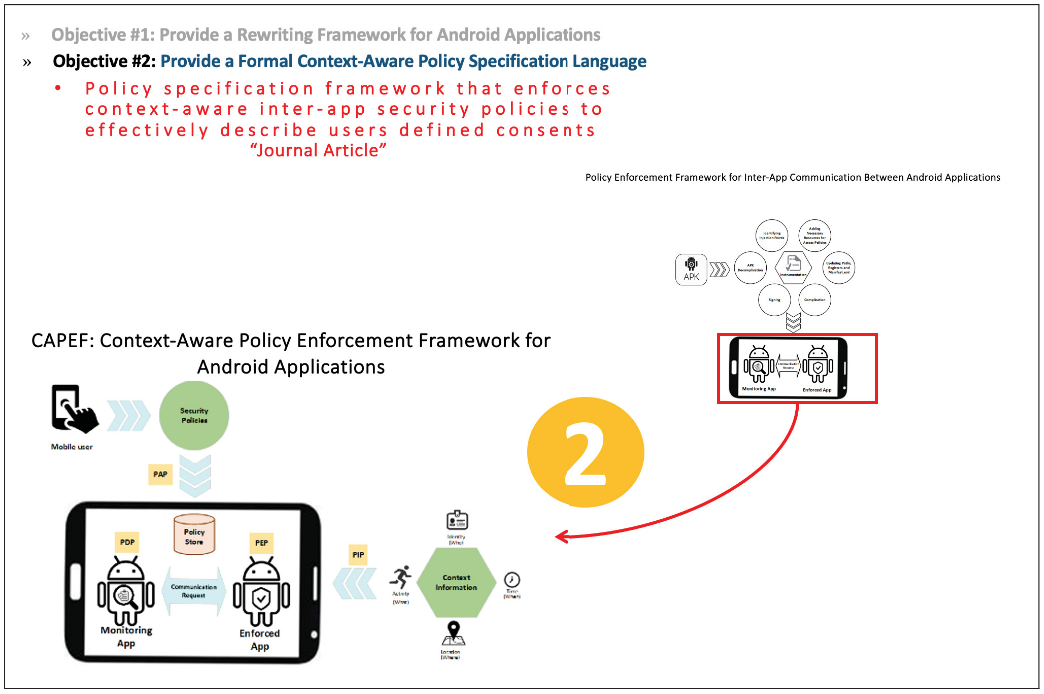


» **Objective #1: Provide a Rewriting Framework for Android Applications**
- Providing a centralized applications controller.
- Providing a simple and effective way to introduce security policies.
- Providing a Policy Enforcement Framework
  "Journal Article (Base Contribution)"

1
Policy Enforcement Framework for Inter-App Communication Between Android Applications
APK
APK Decomplication
Identifying Injection Points
Adding Necessary Resources for Access Policies
Instrumentation
Updating Paths, Registers and Manifest.xml
Signing
Complication
Communication Request
Monitoring App
Enforced App

Figure 0.1 Rewriting Framework: objectives, contributions and architecture.

- To effectively describe users defined consents, in the second article (Chapter 3) and as shown in Figure 0.2 we have provided a formal context-aware policy specification language that could run on resource-limited devices. By using this language, users should be able to define fine-grained context-aware policies depending upon various context attributes, such as time, location, activity, date, application etc.



» Objective #1: Provide a Rewriting Framework for Android Applications

» Objective #2: Provide a Formal Context-Aware Policy Specification Language

- Policy specification framework that enforces context-aware inter-app security policies to effectively describe users defined consents
  "Journal Article"

Policy Enforcement Framework for Inter-App Communication Between Android Applications
APK
Monitoring App
Enforced App
CAPEF: Context-Aware Policy Enforcement Framework for Android Applications
Mobile user
Security Policies
PAP
2
PDP
Policy Store
PEP
Communication Request
Monitoring App
Enforced App
PIP
Context Information

Figure 0.2 Policy language: objectives, contributions and architecture.

- Encrypting critical data on devices is needed to overcome many security and privacy threats. Encryption ensures that sensitive information, such as personal messages, passwords, and financial details, is protected from unauthorized access. Also, encryption can prevent unauthorized access to app data, making it harder for malicious apps to gather sensitive information from the device. Moreover, encryption ensures that data transmitted between a phone and other devices or servers is secure, guarding against interception and tampering. Therefore, in the third article (Chapter 4) and as shown in Figure 0.3 we have integrated lightweight ABE capabilities to provide a context-aware security model. The proposed security model is capable of granting a secure inter-app data communication, by encrypting all requested sensor's sensitive data and making it available only for the authorized applications according to pre-defined context-aware policies.
- To further cope with the current technological advancements of intelligent and dynamic prediction of context-aware activities using machine-learning techniques in Android and IoT

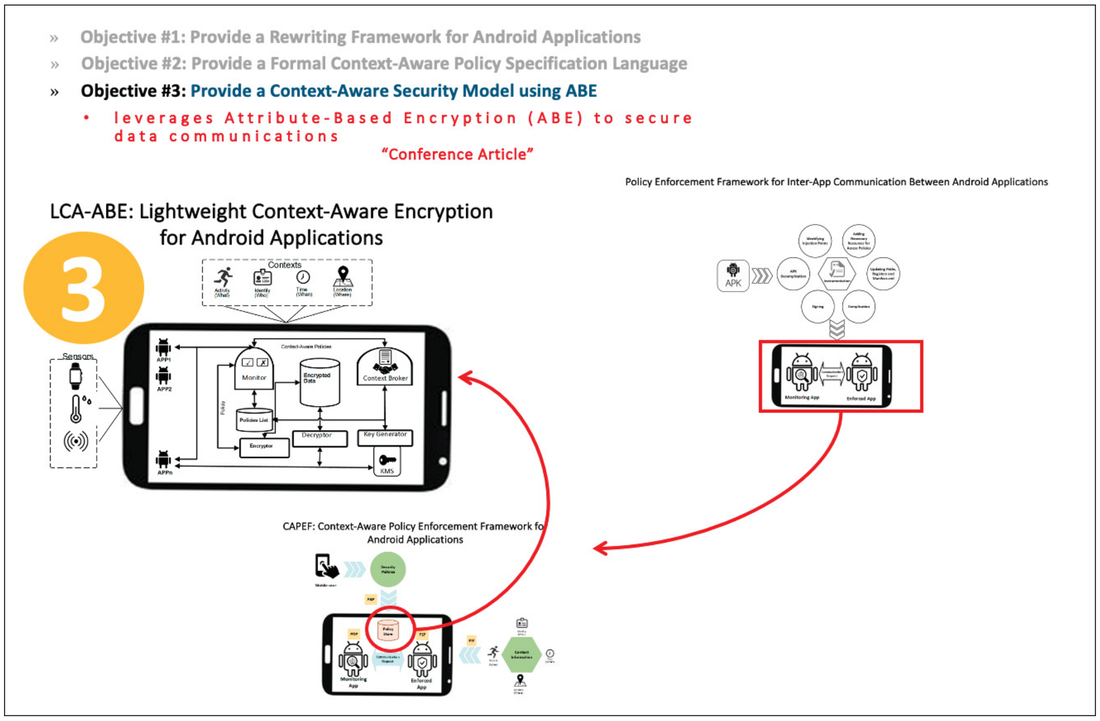
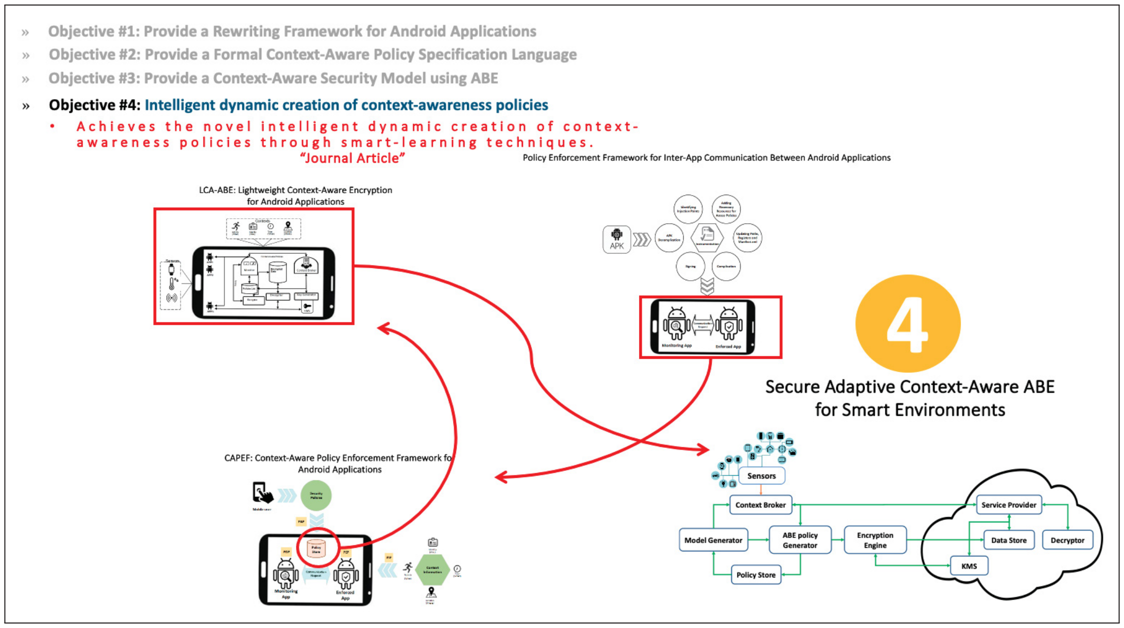

» Objective #1: Provide a Rewriting Framework for Android Applications
» Objective #2: Provide a Formal Context-Aware Policy Specification Language
» Objective #3: Provide a Context-Aware Security Model using ABE
• leverages Attribute-Based Encryption (ABE) to secure data communications
"Conference Article"
Policy Enforcement Framework for Inter-App Communication Between Android Applications
LCA-ABE: Lightweight Context-Aware Encryption for Android Applications
3
Contexts
Sensors
APP1
APP2
APPn
Context-Aware Policies
Monitor
Encrypted Data
Context Broker
Policies List
Encryptor
Decryptor
Key Generator
KMS
APK
Monitoring App
Enforced App
CAPEF: Context-Aware Policy Enforcement Framework for Android Applications
Security Policies
Context Information

Figure 0.3 LCA-ABE: objectives, contributions and architecture.

smart applications. In the fourth article (Chapter 5) and as shown in Figure 0.4, we have proposed a secure intelligent dynamic creation of context-awareness policies. This framework will allow defining and updating consent according to the user's context information in autonomous ways. We also provided a context-aware dynamic encryption model by leveraging ABE capabilities. Our developed model could be applied in many activities such as, predicting user's behavior based on his context information, and his data will be encrypted accordingly to safeguard privacy, by making it difficult for unauthorized parties, including service providers and cybercriminals, to intercept and decipher communications.

- Finally, by applying and examining our adaptive security model for Tactical Networks within a realistic military battlefield scenario, we can open up new and promising research directions.



» Objective #1: Provide a Rewriting Framework for Android Applications

» Objective #2: Provide a Formal Context-Aware Policy Specification Language

» Objective #3: Provide a Context-Aware Security Model using ABE

» **Objective #4:** Intelligent dynamic creation of context-awareness policies

• Achieves the novel intelligent dynamic creation of context-awareness policies through smart-learning techniques.

"Journal Article"

Policy Enforcement Framework for Inter-App Communication Between Android Applications
LCA-ABE: Lightweight Context-Aware Encryption for Android Applications
APK
Monitoring App
Enforced App
4
Secure Adaptive Context-Aware ABE for Smart Environments
CAPEF: Context-Aware Policy Enforcement Framework for Android Applications
Security Policies
Policy Store
Monitoring App
Enforced App
Context Information
Sensors
Context Broker
Service Provider
Model Generator
ABE policy Generator
Encryption Engine
Data Store
Decryptor
KMS
Policy Store

Figure 0.4 Adaptive Contextaware Model: objectives, contributions and architecture.

0.3 Methodology

To achieve our objectives, we have considered and applied the following methodologies:

- **Dynamic rewriting of Android applications:** The API methods in the Android system allow developers to use the most advanced features. Applications can thus access various sensitive properties provided by the Android system. Therefore, these features must be monitored to ensure more secure execution of applications. We have applied the Byte-code Rewriting Approach which is very challenging to modify the behavior of Android applications, by injecting monitoring code before each selected API method's call to intercept it at run time. Android applications are typically written in Java, then the Java compiler compiles the Java source code file to multiple Java Bytecode files, and finally a tool called DX converts the Java Bytecode files to a single Dalvik Bytecode file. Therefore, in order to rewrite any application, we need to convert the Dalvik Bytecode file back to Java Bytecode using existing tools, such as dex2jar, or rewriting the Dalvik Bytecode file directly. In our proposed system, and in order to rewrite the Java Bytecode, we have experimented the AspectJ compiler

Falcone & Currea (2012) for handling the compilation and weaving of the aspects as well as injecting a library dependency required by the aspects. Alternatively, we will rewrite the Dalvik Bytecode file directly in order to avoid any errors during the conversion process. This step requires understanding and mastering the intermediate programming language Smali JesusFreke (2013), which is based on the bytecode. However, before initiating these bytecode changes, we have to focus on de-compilation and recompilation of APK files to obtain the bytecode. To do this, the present study examines the APKTOOL tool Tumbleson (2010), which tolerates "BackSmaling". This work is based on intercepting top API calls used by malwares and automatically rewriting its Android applications. Then, our system will compile the new modified APK that will be able to communicate with the monitoring applications.

- **Defining a formal policy specification language:** To enforce context-aware inter-app security policies and to effectively describe users defined consents. Thus, we have customized the XACML policy language OASIS (2011) to suit Android applications. XACML is an OASIS standard which expresses the access control and security policies based on XML format. We are interested to customize this language because it introduces a number of features that match our needs, namely: the Policy Administration Point (PAP), the Policy Decision Point (PDP), the Policy Enforcement Point (PEP) and the Policy Information Point (PIP). Furthermore, it provides standards for both access control policies and access control request-response format. The compilation can be done statically (offline) or dynamically to load policies for execution. Now focusing on the policies defined by the user, we have adapted this language to develop a centralized monitoring application that allows them to define context-aware policies by specifying various parameters including time, location, applications, device status, etc.
- **Providing secure dynamic creation of context aware policies:** Which has been achieved through smart learning techniques that leverages Attribute-Based Encryption (ABE) for

dynamic encryption. As a result, The data which is being generated from different sources needs to be encrypted before being stored in the device itself or sent to the cloud. Also, based on the context of the user, the data privacy will be changing. In order to encrypt data, ABE needs a policy which contains attributes and operators combining these attributes, e.g. "Bob AND 7AM AND Exercise And Well-being" is a policy for encrypting the GPS data, which makes sure that Fitness application can access the data for that context. But the catch for this process is to generate the ABE policy dynamically, which is challenging. In order to hurdle the challenge, we are introducing a new procedure for the automatic generation of the ABE policies.

- Various prototypes have been developed and evaluated via extensive experiments through which the results have proved the efficiency of the proposed solutions.

0.4 Technical Contributions

Through this thesis, we were able to offer the following contributions:

- We have started by rewriting Android applications to communicate with our centralized applications monitor in order to have access to any sensitive API calls. Moreover, users will be able to enforce their own context-aware polices using our anticipated policy language. However, providing context-aware policies and solutions to these issues are very challenging.
- Next, we have defined a formal context-aware policy specification language that effectively allows users to define fine-grained context-aware policies depending upon various context attributes, such as time, location, date, application etc. Moreover, policies can be remotely defined and enforced using Bluetooth, Wi-Fi or Cloud.
- Provided a secure context-aware encryption model that leverages the power of a lightweight Attribute-Based Encryption (ABE). Where, the encryption and decryption methods could run on resource limited devices such as smartphones and IoT edge devices.

- To further address the new challenges of data security and autonomous context-awareness, we have provided a novel secure dynamic creation of context-aware policies, which has been achieved through smart learning techniques that leverages ABE capabilities for dynamic encryption.
- Promoting the use of our adaptive security model in Tactical Networks domain using a realistic military battlefield scenario.

0.5 Author's Publications

The research outcomes and the contributions of the author, which are presented in this manuscript-based thesis, are either published or submitted in the following journals and conferences:

- Inshi, S., Boudar, O., Ould-Slimane, H., & Mourad, A. (2020, March).Policy Enforcement Framework for Inter-App Communication Between Android Applications. In 2020 Journal of Information Security and Applications. Journal article is under review and updates for re-submission. (Base contribution).
- Inshi, S., Chowdhury, R., Elarbi, M., Ould-Slimane, H., & Talhi, C. (2020, October). LCA-ABE: Lightweight context-aware encryption for Android applications. In 2020 International Symposium on Networks, Computers and Communications (ISNCC) (pp. 1-6). IEEE. Conference article is accepted and published.
- Inshi, S., Elarbi, M., Chowdhury, R., Ould-Slimane, H., & Talhi, C. (2022, December). CAPEF: Context-Aware Policy Enforcement Framework for Android Applications. In Journal of Engineering Research and Sciences (JENRS). Journal article is accepted and published.
- Inshi, S., Chowdhury, R., Ould-Slimane, H., & Talhi, C. (2023, April). Secure Adaptive Context-Aware ABE for Smart Environments. In MDPI IoT Journal 2023. Journal article is accepted and published.
- Inshi, S., Chowdhury, R., Ould-Slimane, H., & Talhi, C. (2022, November). Dynamic Context-Aware Security in a Tactical Network Using Attribute-Based Encryption. in

MILCOM 2022 - Cyber-Physical Security in Mission-Critical Tactical Networks (CyberNet) - Cyber Physical Security (#1570862859) IEEE. Conference article is accepted and published.

0.6 Thesis Organization

We start with a general literature review then we detail each of our contributions in different chapter. Finally, in the last part, we conclude the thesis and draw some future directions out of remaining open research questions.

CHAPTER 1

LITERATURE REVIEW

In this chapter, we review different approaches relevant to various aspects subject to this thesis to formulate a general context for this research work . Yet, we leave the analysis of these approaches to each chapter separately, where we compare our propositions with existing works based on different aspects relevant to each contribution.

1.1 Malware Detection

Enck et al. Enck *et al.* (2014) proposed a static and dynamic analysis tool called TaintDroid to detect and analyze real time private information leakage in Android applications. TaintDroid allows applications to track information-flow along with register values by replacing the Dalvik virtual machine with a tracking implementation. This implementation requires a dedicated modification of the underlying operating system. Mann et al. Mann & Starostin (2012) proposed a framework for static detection of privacy leakage in Android APIs and identified some critical privacy policies. Efforts have also been made by other researchers and propose some approaches such as AndroidLeaks Gibler, Crussell, Erickson & Chen (2012), SCANDAL Kim, Yoon, Yi, Shin & Center (2012) and CHEX Lu, Li, Wu, Lee & Jiang (2012) to detect and analyze privacy leakage either statically or dynamically. These approaches can scan applications for potential attacks without the need for executing the application in some situations; also, can scan applications while accessing sensitive information at runtime. DroidForce Rasthofer, Arzt, Lovat & Bodden (2014) provided a technique to detect and report data-flows by enforcing complex data-centric polices on Android applications. This approach allows users to specify which data can be accessed or used on their devices. The supported policies by this approach can be dynamically changed at runtime ant it requires no modification in the underlying operating system. For example, by using this system we can enforce a policy that limits the number or the frequency of SMS messages that can be sent to a specific set of phone numbers. This policy will help detecting and preventing one of the major treats which is stealing the user's money through premium-rate SMS messages. Yang et al. Yang *et al.* (2014) proposed APKLancet tool that can automatically diagnosis Android application and discover unwelcome code fragment such as advertising libraries or malicious code. The proposed approach does not require modification of the diagnosed system. Also, Android/BadAccents Rasthofer, Asrar, Huber & Bodden (2015) presented a technique that used a phishing attack to steal bank account credentials. In such bank Trojan attacks, the victims will be asked to enter their confidential data through a Graphical User Interface (GUI) that looks identical to the one of a benign mobile banking application. However, the malware's GUI is designed by the attacker, and is developed to steal the private information. More recent, AndroDialysis Feizollah, Anuar, Salleh, Suarez-Tangil & Furnell (2017) is proposed to explore Android intents (implicit and explicit) as a feature for malware detection.

Discussion: Based on the reviewed approaches, the current malware poses many challenges to malware analysis techniques in order to trigger malicious behavior, showing the need for further research in this area. Therefore, we will discuss some policies that can be enforced by our proposed solution. For instance, we look at malware that tends to subscribe to premium-rate services with background SMS messages. Based on this characteristic, most existing malware intercept incoming SMS messages (e.g., to block warning or billing information). This problem might be rooted in the lack of fine-grained control of related APIs (e.g., sendTextMessage). In this case Android permission model can be possibly expanded to include additional Context-aware Policy Enforcement to better facilitate users' sound and informed decisions. Moreover, we can propose a policy that can monitor and limit the number or the frequency of SMS messages that can be sent to a specific set of telephone numbers. In addition, by proposing data-centric policies, we can provide unique advantages in blocking mobile malware from entering the contacts and emails. As an example, we can enforce policies that control access to the location of the private data center where every attempt of sending contacts data over the Internet must be manually approved by the user (via popup notification). Also, we can allow applications to access the location data and the Internet, but prevent it from sending out the location data to a remote adversary. In another example, let's consider malware that tends to hide malicious activities in native code. As indicated above, in the example hiding the credentials of the E-mail account in native code, this makes the detection hard for existing static analysis approaches. Therefore, we can propose some new or updated static analysis techniques that can enforce policies to monitor and prevent such activities. In addition, we can propose some policies that can prevent dynamic loading ability of both native code and Dalvik code which are being actively abused by existing malware. The challenging point will be to determine how to develop effective solutions that prevent such activities from being abused while still allowing legitimate uses to proceed.

1.2 Repackaging Detection

Nowadays, reverse engineering the Dalvik bytecode that has been used in the Dalvik virtual machine is not a hard issue for many Android application developers. Therefore, Repackaging Android applications have become a serious problem that violating the users' privacy and the developers' intellectual property. In order to detect repackaged application and to prevent their propagation, many researchers proposed repackaging detection algorithms for Android applications. For instance, Crussell et al. Crussell, Gibler & Chen (2012) proposed a tool called DNADroid which compare program dependency graph for two Android applications to detect any similarity or cloning. Zhou et al. Zhou, Zhou, Jiang & Ning (2012) Applied specialized hashing technique, called fuzzy hashing and developed a tool called DroidMOSS. This tool can systematically detect repackaged Android application by splitting the program instructions into small units and a hash value computed for each local unit instead of computing a hash over the entire program instruction set. Also, Zhou et al. Zhou, Zhang & Jiang (2013) proposed and developed a prototype called Appink which applied watermarking mechanism to prevent Android applications repackaging attacks. AppInk introduced a concept called manifest app to make the watermarking mechanism deployable into current applications development practice. Manifest app encapsulates a sequence of input events for the target application automatically without any user intervention. More recent, CloneSpo Martín & Hernández (2019) presented a methodology to detect repackaged android applications using meta-information available in most application markets.

Discussion: In this section, we look at repackaging or impersonating other legitimate applications. This means an application gets installed to impersonate an original app and activate some malicious components. In this case, we could effectively mitigate the threat by enforcing the existing Android apps for repackaging detection. However, the challenges lie in the large volume of new apps created on a daily basis as well as the accuracy needed for repackaging detection. Moreover, Android malware could encloses platform-level exploits to escalate their privilege. For example, the attacker tries to obtain Android Device Administration privileges without the user's knowledge. In this characteristic, we can monitor and enforce policy on the Android Device Administration API. Finally, other malware characteristics can be extracted from the reviewed techniques, such as attacks that can generate a UI element which can be layered over applications and routes touch gestures to the underlying application. In this case we have to enforce policy on the Android user interface (UI) design API, more precisely, by monitoring and enforcing inter-apps communication policies, which is one of our main research objectives.

1.3 Modifying the Android Platform

Many researchers introduced security frameworks that rely on modifying the open-source Android platform to develop custom builds of Android. Hornyack et al. Hornyack, Han, Jung, Schechter & Wetherall (2011) provided AppFince tool that applied an information-flow based solution. This solution requires modifications on the stock software stack to enforce privacy policies, either at the source or the sink to mitigate leakage of private information.
More recent, Feth et al. Feth & Pretschner (2012) proposed a more fine-grained security system to enhance the standard permissions of Android operating system. This work leverages TaintDroid Enck *et al.* (2014) prototype to detect and report data-flows for Android applications by enforcing complex polices built based on special conditions. This approach allows users to prohibit some actions such as playing movie more than twice even if several same copies have been created. Also, SecureDroid Arena, Catania, La Torre, Monteleone & Ricciato (2013) addressed the issue of controlling security policies while applications are running in the Android environment. During the installation of an application, Android allows the user to

grant permission for an application to use certain features of the system. This work introduced an extension of Android's security framework in order to improve the standard permission control provided by the operating system. In Martinelli, Mori & Saracino (2016) the authors proposed modification to Android architecture based on usage control model for application isolation, though it is possible to achieve application isolation by proper utilization of multi-user environment. Recently, CoDRA Thanigaivelan, Nigussie, Hakkala, Virtanen & Isoaho (2018) presented as access control system that provides a fine granular policy and an enforcing ability that enables diverse policy configurations within different levels of system operations. CoDRA extended the Android security mechanisms, such as sandboxing and permission model, in order to improve the overall security of the device.

Discussion: Modified Android platform has a number of major drawbacks such as:

- Requires building custom firmware and platform code. Although there are some benefits to having custom platform such as accessibility for extra features, it is still considered as a disadvantage where users might void their warranty or some of the device features not always work.
- Requires rooting the device to gain administrative privileges. This service allows users to run special applications which needs root access on the device; on the other hand, this might void the user's warranty.
- Different devices and different versions of the platform. This problem will prevent some features of the applications from working effectively in most of the times.
- Applications are limited to the security policies that are supported by the modified Android framework.
- The modification of the Android platform will make it difficult to be deployed on millions of mobile devices.

1.4 Rewriting Android Applications

The latest set of works studied and proposed security mechanisms that do not require firmware modification. In our context, many efforts are made to rewrite Android applications in ordered to enforce some security policies. For instance, Aurasium Xu, Saïdi & Anderson (2012) is a concurrent approach that rewrites Android application to sandbox important native API methods and monitors the behavior of the application to detect any security violations. Jeno et al. Jeon *et al.* (2012) used Dalvik bytecode rewriting approach to interpose on all invocations of the APIs methods and inject the desired security policies. Davis et al. Davis, Sanders, Khodaverdian & Chen (2012) designed and implemented a rewriting framework called I-arm-droid. This prototype is capable to identify a set of security-sensitive API methods and specify their security policies, which can be modified to satisfy the security needs for each Android application. Davis et al. Davis & Chen (2013) provided an another rewriting prototype called RetroSkeleton based on their previous work I-arm-droid to insert, remove or modify Android application behavior without modifying the Android platform. This rewriting approach supports the transformation of variety of useful policies such as flexible fine-grained access control and improves some security policies of network communications. Moreover, this system allows policy-writers to specify high-level policies to be transformed automatically to arbitrary Android application. The work in von Styp-Rekowsky, Gerling, Backes & Hammer (2013) provides an inline reference monitoring approach to enforce security policies at runtime, by replacing the references to security-relevant methods in the Dalvik Virtual machine internal representation. Another research paper Zhang, Ahlawat & Du (2013) proposed a prototype called AFrame, which used the bytecode rewriting approach for isolating advertisements from Android applications to isolate untrusted third-party code from the applications. Moreover, some other researchers in Pearce, Felt, Nunez & Wagner (2012) Shekhar, Dietz & Wallach (2012) have proposed two other systems called AdDroid and Adsplit respectively which also have been used for separating smartphone advertising from applications. Zhang et al. Zhang & Yin (2014) deployed a Context-aware policy enforcement mechanism to mitigate privacy leakage in Android applications. This mechanism will enforce privacy policy based on user preferences. Moreover, they implemented a prototype called Capper. By using this system, when a user tries to install any Android application the bytecode rewriting engine called BRIFT will rewrite the program of this application by selectively inserts instrumentation code along taint propagation slices for monitoring and preventing any information leakage. In summary, to rewrite any given application by Capper, first they convert Dalvik DEX file into Java bytecode using the tool dex2jar pxb1988 (2014). Then they used Java bytecode optimization framework to translate the Java bytecode to intermediate representation (IR) to perform a static dataflow analysis and bytecode instrumentation. For the static instrumentation, they created a shadow for each entity in the taint propagation slices individually. Then they optimize the added instrumentation code to remove any redundant bytecode. Finally, they converted the rewriting bytecode to a new package with the old resources to create a new .apk file. More recent research in Xie, Fu, Du, Luo & Guizani (2017) presented AutoPatchDroid, which is a bytecode patching framework to defend against inter-app vulnerabilities in android application. Another work in Vronsky, Stevens & Chen (2017) provided SurgeScan tool that consists of static and dynamic loaded code to specify and enforce security policies on untrusted third-party code. Another interesting research called Weave Droid Falcone & Currea (2012) has provided a framework for weaving AspectJ aspects into an Android application. The framework takes two inputs at the beginning: APK and a set of aspects that will be weaved into the APK. The weaving process will be performed on the Android device. Also, very recently in Grace & Sughasiny (2022) They have developed a lightweight monitoring system to detect malware activities with the log file and they evaluated the proposed model according to Policy-based permissions.

Discussion: Rewriting Android applications for security enforcement has several advantages; on the other hand, it also has some limitations, such as the Android applications may tamper with Dalvik VM using native code for many reasons, such as removing any instrumentation that has been added to the original bytecode. In this case, this procedure will disable the policy check. In addition, some of the reviewed frameworks have provided enforcement mechanisms to mitigate Malware activities by enforcing context-related policies, however they didn't afford a policy specification language that runs on Android system as an application or a service without modifying the Android platform. Thus, this thesis is introducing a more featured policy specification language that allow regular users and any company to easily interpret and enforce their complex context-aware inter-apps policies on their Android mobile applications. The table 1.1 shows the summary and differences between our research and the other works.

Table 1.1 Summary of related work

Approach/ System	Methodology	Required modification	Policy Language	Context-aware inter-app Privacy Leakage Prevention
TaintDroid Enck *et al.* (2014)	Dynamic Analysis	Android Platform	✗	✗
Appink Zhou *et al.* (2013)	Watermarking	Application	✗	✗
Apex Nauman, Khan & Zhang (2010)	Policy Enforcement	Application	✗	✗
TISSA Zhou, Zhang, Jiang & Freeh (2011)	Resources Access Control	Android Platform	✗	✗
AppFince Hornyack *et al.* (2011)	Dynamic Analysis & Resources Access Control	Android Platform	✗	✗
Aurasium Xu *et al.* (2012)	Rewriting Java Bytecode	Application	✗	✗
I-arm-droid Davis *et al.* (2012)	Rewriting Dalvik bytecode	Application	✗	✗
AFrame Zhang *et al.* (2013)	Isolating Advertisements	Application	✗	✗
Capper Zhang & Yin (2014)	Rewriting Java Bytecode	Application	✗	✗
SecureDroid Arena *et al.* (2013)	Policy Enforcement	Android Platform	✓	✗
Weave Droid Falcone & Currea (2012)	Isolating Advertisements	Application	✗	✗
Grace & Sughasiny (2022)	Policy-based permissions	Application	✗	✗
CAPEF (Inshi, Elarbi, Chowdhury, Ould-Slimane & Talhi, 2023b)	Policy Language	Application	✓	✓

1.5 Context awareness

Context-awareness is a term that is first introduced by Schilit & Theimer (1994), to acquire, understand, recognize the context, and takes an action according to that precise context. Context-awareness evolved from desktop applications, web applications, mobile computing, cloud computing, to the Internet of Things (IoT) over the last years.

In this domain, many researchers and engineers have proposed and developed several solutions using context-aware computing techniques. Also, the context-aware policies vary depending on each domain requirements, such as mobile computing, web applications and recently IoT smart environments. One of the leading mechanisms of defining context types in IoT environments is presented in (Abowd *et al.*, 1999), where they have two main categories: Primary context, which represents any information retrieved directly from sensors, such as location data from a GPS sensor, user identity based on SIM card, time read from a device clock, or activity from a smart watch sensor. Secondary context, which represents any information that can be computed using primary context such as predicting the user activity based on the user calendar or predicting the user location based on an image retrieved from map service provider. Another context-awareness taxonomy for IoT environments is presented in (Perera, Zaslavsky, Christen & Georgakopoulos, 2013). According to the taxonomy, we can split the context-awareness in four main steps: aggregation, modeling and reasoning, named the context life-cycle (Perera *et al.*, 2013). Each phase of the life-cycle has its own challenges. The challenges of the acquisition step are related to the context-aware system capability to support the registration of new data sources (i.e., data providers, sensors, and devices), and its technical issues, as well as the network communication used to acquire the data source information. The modelling and reasoning phases hold the same challenges because these phases strictly depend on each other.

In today's era of big data and smart environments, many researchers raise the essential need for context awareness and the dynamic access policy management to protect resources from unauthorized access. In (Alkhresheh *et al.*, 2018) authors proposed an automatic access policy specification framework for IOT environments based on a rule-based method. This research conducted an interesting dynamic accessibility method, but does not provide an adequate machine learning method that supports such adaptive access control.

Discussion: Even though previous studies have covered many context-aware aspects, two main aspects are still challenging: The first aspect is the lack of a context-awareness broker that can manage the complex of interconnected sensors with different smart devices and service providers. Where in these situations context-aware policies will play the main role in deciding what and when data needs to be processed or shared and much more. In addition, the current smart IOT environments need such context-awareness broker as central point of control, as different middleware solutions developed by different parties will be employed to connect to sensors, collect, model, and reason context. The second aspect is updating contents based on the context information, while the data aggregated by the sensors needed to be shared with the authorized providers or observers, consents are not fixed and can be updated not to share date at certain contexts. For example, consent can be updated no to share data when switching between personas, switching between user profiles, or not to share data based on location and time (meeting, enterprise, home). Also, consent can be updated to share data again based on an event, such as insurance company needs to get the data back to know what caused as accident. Therefore, to effectively perform this aspect, we need an intelligent dynamic adaption of contexts, which can be achieved through smart learning techniques. Where machine learning techniques will take the context information as inputs to learn models able to adapt the consents accordingly.

1.6 ML in context awareness

Context awareness aspects along with Machine learning techniques have been successfully used in different domains, including mobile cloud computing, E-health, smart homes and IoT smart autonomous environments where most of these systems rely on intelligent context-aware applications. Researchers in (Sarker *et al.*, 2020)(Sarker, Kayes & Watters, 2019) have investigated various popular machine learning classification techniques, such as decision tree (DT), k-nearest neighbour (KNN), random forest (RF), naive Bayes classifier (NB), support vector machines (SVM), and deep learning, for creating smart models for different Mobile use-cases. In mobile cloud computing domain, some studies, such as (Nawrocki *et al.*, 2020) have analyzed the context of the device's operations and their related security aspects. Also, they have developed an agent-based adaptive system that used machine learning algorithms for enabling the optimization of services on the mobile device and to secure allocation of tasks in conventional cloud resources. Moreover, the idea of using ML and context awareness is investigated in many studies, such as (Mshali *et al.*, 2018) (Sarker & Salah, 2019) to provide the appropriate services to users in E-health and smart homes.

Discussion: The above discussed ML and context awareness techniques allow the definition of fine grained context-aware access control policies for data privacy in different domains. Yet, none of the existing works fulfills our requirements of automated creation of context-aware policies, and the automated encryption according to these auto-generated policies. Therefore, we have experimentally tested more use-cases to automatically analyze the user behavior according to different context information and the utilization of his smart-devices datasets.

1.7 Attribute Based Encryption

(Bethencourt, Sahai & Waters, 2007)(Goyal *et al.*, 2006) are the first authors to propose encryption which allows access control based on the attributes. According to various research, ABE is one of the best techniques to have access control along with data security . ABE is a public-key encryption technique which utilizes attributes and policy for encrypting and decrypting the data as well as access control. Attributes can be basically anything like postal code, department, location, services, etc. There are two types of ABE, Cipher-text Policy ABE (CP-ABE)(Bethencourt *et al.*, 2007) and Key Policy ABE (KP-ABE)(Goyal *et al.*, 2006).

There has been lot of researches done in various domains with ABE. (Zhang *et al.*, 2020), (Oberko, Obeng & Xiong, 2021) and (Ullah *et al.*, 2021) performed different types of analysis regarding the feasibility, performance and surveys of ABE in cloud, smart devices and smart environment. (Ambrosin, Conti & Dargahi, 2015) provided a comprehensive study in terms of resource utilization and execution time for CP-ABE and KP-ABE in constrained devices. From their experimentation, it is clear that ABE can be incorporated in resource-constrained devices. (Maheswari & Gudla, 2017), (Akinyele *et al.*, 2011) proposed that ABE can be used to secure patients data before storing it on the cloud where only the specific personnel can be able to view the patient records and also (Taha & Chowdhury, 2020) showed that it is important to encrypt the patient data collected from different sources in the hospital before sending it to the cloud. (Chowdhury, Ould-Slimane, Talhi & Cheriet, 2017) has implemented a framework using OpenHAB to secure the data generated in the smart-home before being shared with the cloud service providers. In (Tan, Goi, Komiya & Tan, 2011), the authors studied the feasibility of the ABE in Body Sensor Network and proved that it has a very good prospective in the domain of smart devices. (Shao, Lu & Lin, 2015) proposed online/offline attribute-based proxy re-encryption for smart phones to encrypt the data before it is transferred to the cloud. They have also showed the performance and feasibility of using their ABE scheme in the smart devices like cellphones. (Taha, Ould-Slimane & Talhi, 2020) proposed a CP-ABE offloading technique where the authors performed partial encryption in the resource constrained devices and the actual encryption on the cloud, where there are enough resources.

Discussion: Security is a key requirement to the evolution of the new automated smart IoT environments that provide security services and identity managements. This growth raises the need to protect data exchange between different entities by many ways (e.g., authorization, automated access control, and data encryption). Therefore, context Awareness Encryption is the key solution to the new secure smart systems evolution. Table 1.2 shows the Comparison between our research and the related works.

Table 1.2 Comparison of existing related work

Reference	Domain	Rule base	Machine Learning Algorithm	Context aware	Dynamic	Multi users	Privacy	ABE
BehavDT(Sarker *et al.*, 2020)	Mobile	✗	Decision Tree	✓	✓	✗	✗	✗
ACAO(Nawrocki *et al.*, 2020)	Mobile Cloud Computing	✗	Naïve Bayes (NB), Decision Tree, Random Forest	✓	✓	✗	✓	✗
SMAF(Mshali *et al.*, 2018)	E-health	✗	Grey Model (GM)	✓	✓	✗	✗	✗
(Alkhresheh *et al.*, 2018)	IoT	✓	✗	✓	✓	✗	✓	✗
AppsPred(Sarker & Salah, 2019)	Smart-home	✓	Random Forest	✓	✓	✗	✗	✗
HybridGuard(Phung *et al.*, 2017)	Hybrid Mobile	✓	✗	✓	✗	✗	✗	✗
LCA-ABE(Inshi *et al.*, 2020)	Mobile	✓	✗	✓	✗	✗	✓	✓
Our work	Smart Environment	✓	Artificial Neural Network Markov Chain	✓	✓	✓	✓	✓

1.8 Conclusion

We reviewed in previous sections, existing approaches relevant to Android security, Context-awareness using machine learning and context-aware Encryption techniques for a smart autonomous environments. As discussed above, though the efficiency of the existing propositions, many technical limitations need to be addressed in order to meet with the new context-aware, adaptable, intelligent and lightweight security solution requirements. Each of the following chapters is devoted to tackle set of privacy and security issues and provide part of the full solution proposed in this thesis.

CHAPTER 2

POLICY ENFORCEMENT FRAMEWORK FOR INTER-APP COMMUNICATION BETWEEN ANDROID APPLICATIONS

2.1 Introduction

In today's highly competitive world of mobile applications development, enterprises need flexibility in meeting customers complex demands by developing tens of thousands of applications and make them available on the application markets, such as Google Play and App Store. Android has become the leading mobile platform and has dominated the market with share increase from 85.1% in 2018 to 86.6% in 2019 with many 5G launches IDC (2020). This leadership has been accompanied by the significant increase of Android applications that are available for users to easily install and download. While these apps can benefit customer needs, it may also have an undesirable effect that might occur without user's consent, namely, private information leakage Enck *et al.* (2014). In this context, previous studies Enck *et al.* (2014); Hornyack *et al.* (2011); Wu, Zhou, Patel, Liang & Jiang (2014) have revealed that many applications, either benign or malicious are sending private information to remote servers without user's awareness. Moreover, the permission system of the current Android platform still has some limitations, such as users having to grant all permissions requested by an application in order to install it, without being able to modify most of these permissions afterwards. Therefore, preventing the risk of private information from being leaked to unauthorized users is a major concern for today's Android platform and applications.

To enforce the security of the Android platform and its applications, many researchers have proposed a variety of systems to detect, evaluate and mitigate Android privacy leakage Yu, Xuming, Xiao, Lin & Jingzhao (2020); He, Yang, Hu & Wang (2019); Liu, Liu, Zhu, Wang & Zhang (2019); Onwuzurike (2019). The systems reviewed in the present work can be classified in three major categories, as per the following: The first category consists of purely static Enck *et al.* (2014); Mann & Starostin (2012) or dynamic Rasthofer *et al.* (2014); Yang *et al.* (2014); Rasthofer *et al.* (2015) approaches that analyze Android applications code to detect any malicious activities before or at runtime. In some situations, these approaches can scan applications for potential attacks without the need for executing the application; also, they can scan applications while accessing sensitive information at runtime. Yet, most of these approaches can only detect intra-procedural data flow potential attacks. Also, a dedicated malware author can still find ways to circumvent these approaches' confinement. For instance, malware can ex-filtrate private information through side channels, such as implicit flows and timing channels. Further, the reviewed techniques cannot handle native components and Java reflective calls in a general way.

The second category provides solutions that require Modifying the Android Platform to enforce flexible security policies Hornyack *et al.* (2011); Feth & Pretschner (2012); Arena *et al.* (2013). For instance, Feth et al. Feth & Pretschner (2012) has proposed a fine-grained security system to enhance the standard permissions of Android operating system. Also, SecureDroid Arena *et al.* (2013) provides an extension of Android security framework to enforce context-dependent policies, such as defining context policy to restrict the use of a resource to a maximum number of times per day, or to deny access to the resource after a certain time of day. The main advantage of this category is that it does not require enforcing policies for each installed application independently; all policies enforced by the system will be applied directly and easily on all applications. However, this category has a number of major drawbacks such as the need of building different versions of firmware and platform codes, where applications will be limited by the security policies supported by the modified Android platform.

The third category includes solutions that are based on Rewriting Android Applications Xu *et al.* (2012); Zhang & Yin (2014). These solutions require no modification to the Android platform and can be easily deployed. Moreover, they provide access control mechanisms that monitor and enforce policies on sensitive APIs. For example, Zhang et al. Zhang & Yin (2014) was among many other researchers who developed a bytecode rewriting approach of Android applications by implementing a prototype called Capper to trace and mitigate privacy leaks in Android applications. Moreover, Davis et al. Davis *et al.* (2012) designed and implemented a rewriting framework called I-arm-droid which embedded In-App reference monitors into

Android applications. However, this approach also has some limitations, mainly; rewriting can only be applied for each application separately.

The above three categories have provided some security frameworks (more details and comparisons can be seen in section 8). These frameworks can enforce context-related policies, but they cannot provide inter-apps context-aware policies. Additionally, though some frameworks allow many parties to define security policies, but those security policies cannot satisfy all users' needs. Therefore, we need a policy specification language to allow regular users and any company to easily interpret and enforce their complex rules on their Android mobile applications.

Our work falls within the third category which is Rewriting Android Applications to effectively enforce different behaviors and policies to each application individually without the need to modify the underlying platform. The modified applications communicate with a centralized applications monitor that we propose in order to avoid any malicious activities happening without the user's awareness. This work also address applications that are not primarily malicious but can be extremely dangerous in particular contexts.

2.1.1 Challenges

Rewriting and monitoring context-aware inter-app policies of sensitive APIs on Android applications presents several challenges:

- Extracting permissions and their respective invoked API methods from the byte code of Android application for reinforcement and instrumentation procedures.
- Context-aware inter-app policies are difficult to predict as they frequently get changed and needs to be updated accordingly for accuracy and correctness.
- Beside the difficulty of representing the security policies in a logical language which can contain user contexts and semantics, a key challenge is how to design and develop effective and efficient algorithms to monitor private information leakage on semantics levels.

- There are significant differences between the register-based Dalvik Virtual Machine and the stack-based Java Virtual Machine, these differences resulting in information loss when converting bytecode from one format to the other.
- Due to the resourced constrained mobile devices, we have to decide, in early stages the instrumentation and monitoring location, whether to be on device, external PC or App market.

2.1.2 Contributions

The main focus of this work is to elaborate context-aware inter-app policy enforcement framework for Android applications by rewriting these applications to communicate with our centralized applications monitor in order to have access to any sensitive API calls. Moreover, users will be able to enforce their own policy using our anticipated policy engine. However, providing policies and solutions to these issues are very challenging. To the best of our knowledge, the current approaches in the literature are not fully satisfactory, since, these approaches do not provide a centralized control over the inter-app behaviors while they are running. They only provide an independent control for each single application.

Our main contributions in this work are:

- Providing a rewriting framework. This framework will force any Android applications to communicate with our centralized applications monitor before calling any sensitive API calls. Also, it will enforce many security polices at run time on the same application and across different applications that might share the same device resources.
- Providing a centralized applications controller. This will allow users to manage all API calls performed by the applications installed on the device.
- Providing a simple and effective way to introduce security policies. Security policies typically change more often being tailored to the privacy needs of the users. Thus, we need a policy engine with a user interface to allow regular users and enterprises to interpret and enforce their complex rules on their Android mobile applications. Policies have to be considered at different priority levels to decide which to enforce in case of conflicting policies.

- Providing a Policy Enforcement Framework for con-text-aware inter-app access control to mitigate privacy leakage. By using this framework, users should be able to define fine-grained context-aware policies depending upon various context attributes, such as time, location, date, application etc. Moreover, policies can be remotely defined and enforced using Bluetooth, Wi-Fi or Cloud

2.2 Approach Overview

The API methods in the Android system allow developers to use the most advanced features. Applications can thus access various sensitive properties provided by the Android system. For example, sending text messages, accessing the camera, accessing GPS coordinates, etc. These features must therefore be monitored to ensure more secure execution of applications.

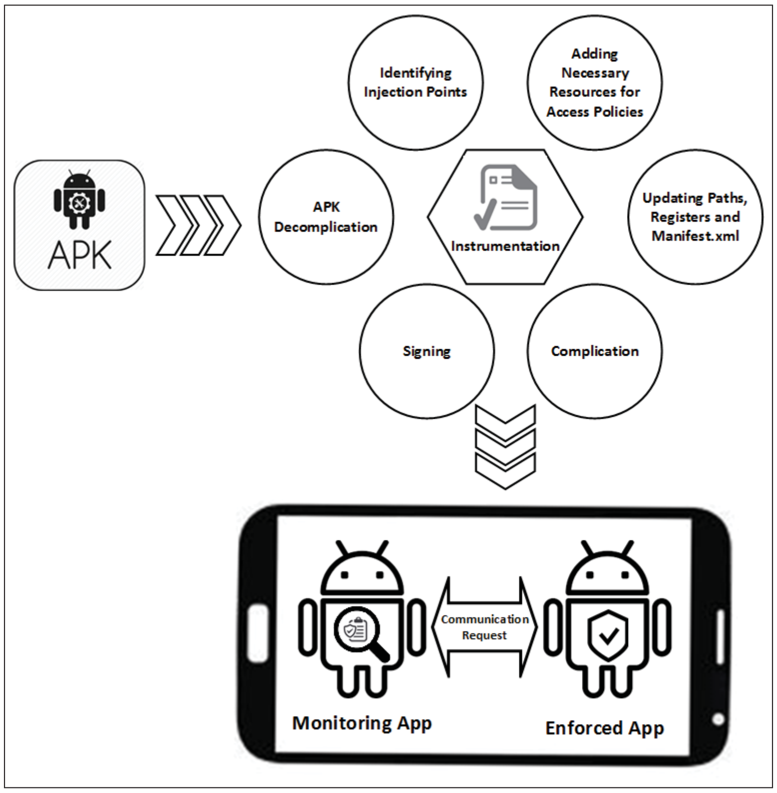

APK
APK Decomplication
Identifying Injection Points
Adding Necessary Resources for Access Policies
Instrumentation
Updating Paths, Registers and Manifest.xml
Signing
Complication
Communication Request
Monitoring App
Enforced App

Figure 2.1 Overall-architecture

In this article, a solution to modify the behavior of Android applications is provided to have a control mechanism on any sensitive API calls. It also allows the users of these enforced application to define and monitor a set of context aware security policies. Thus, when an enforced application attempts to execute an API method, that call will be intercepted to retrieve the parameters needed to control it. With these parameters as well as the user-specified security policies, the decision on the authorization or prohibition of the execution of the method can be taken. And to allow the interception of the calls, some instructions will be added to the application to communicate with our developed application controller. In this case, the enforced application will send the collected parameters from the call to the application controller, which uses the list of security policies, makes the appropriate decision and sends the response to the application. Accordingly, the controller will allow, prohibit, or pause the execution of this API method. Figure 2.1, shows the overall architecture of the present research work, where the user provides the APK file of the application to be reinforced. Then the application will go through several steps to obtain a reinforced version capable of communicating with the controller. The nature of the instrumentation and the instructions to be added to the application as well as their locations will be explained later in this article.

2.3 System Design

The objective of this work is to implement a flexible system allowing the enforcement of context aware security policies on Android applications. Several alternatives designs have been considered in this study to have an efficient solution that can balance execution time and resource consumption.

- *Modification of the Android platform*: This approach is based on the modification of the Android system which can offer several advantages such as, it does not require rewriting applications after each download. Also, this approach can integrate the security policies in the Android system and allows the control of the information flow throughout the platform. However, this approach contains a number of disadvantages such as, the operation of solutions based on this approach requires administrative rights, which implies the need to root the device. This may result in loss of user warranty. In addition, this approach provides different solutions that allows having several versions of the Android framework. In this case, compatibility issues may arise between applications and the installed system, as well as between frameworks and device types. Additionally, security checks may be limited, as applications are limited to the security policies support-ed by the modified Android framework. Therefore, this approach will not be adopted in this study.

- *Decompilation to Java source code*: Android applications are developed in Java language, which will be compiled to Java bytecode, then to Dalvik bytecode to run it on the Android devices. While the Java virtual machine has been used for much longer time than the Dalvik virtual machine, the tools for decompiling the Java bytecode are more mature. However, the Dalvik virtual machine is different from the Java virtual machine, since it is based on registers not on stacks. Several tools like dex2jar pxb1988 (2014) and ded Lab (2014) are developed for allowing the conversion of the Dalvik bytecode to Java byte code. But, this conversion phase might cause loss of some critical application functionality. Therefore, in this study, we have adopted a solution based on the decompilation of the Dalvik bytecode to Smali bytecode JesusFreke (2013). This is an intermediate byte code, which is more complex, but offers a much higher rate during the recompilation phase.
- *Application Rewriting Framework*: We have considered two approaches on running our developed system: The first approach represents a remote application, running on a web server. The user then accesses a website that allows him to introduce his application and download the enhanced APK. The second approach provides a third-party Android application. This application is developed in Java and can be installed on a device running the Android operating system. Then, the user can use this application on his device to reinforce any APK file of targeted applications.
- *Integration of the application controller*: The solution presented in this study is based on a controller for managing security policies. To achieve this target, two approaches have been investigated: First, allowing the integration of the controller within the Android applications. In this case, the control code will be injected into the targeted application code. This method does not require Inter-Process Communication (IPC) between the enforced application and

the controller. The second approach is based on a third-party controller that offers the advantage of managing multiple applications in parallel. Also, will mitigate the risk of collaboration between multiple applications to create a malicious context. Therefore, we have adopted this approach to allow more flexible management of Inter-app security policies through a useful user interface.

2.4 Application Rewriting Framework: Technical Details

Once a user downloads an application on his device, the APK file will be passed through several steps to obtain an enforced APK that can communicate with the controller. In what follows, we present the main steps of the proposed Application Rewriting Framework.

2.4.1 Decompiling Applications

Android application developers write their source code in java. The compilers are subsequently used to convert the Java source code into dalvik (.Dex) executable codes to be executed by the dalvik virtual machine. Therefore, in order to make changes to an application, the source code should be modified and then recompiled into a new dalvik executable file. But unfortunately, the user does not have access to the Java source codes of all applications. The user can have access to the dalvik executable file, which is not human-readable. Hence, we need to convert dalvik executable files into a Smali intermediate byte code JesusFreke (2013), which provides a more comprehensible reading while supporting modifications using the APKTOOL Tumbleson (2010). In our study, the targeted APK file will be decompiled and the generated Smali files will be scanned line by line to undergo the necessary modifications, which are mainly portions of code allowing enforced applications to communication with the controller.
In Smali code, there are three main types of methods: static methods, instantiation methods, and constructors.

- *Static methods:* Static methods are called in the Smali code by these two statements: Invoke-static and invoke-static/range. Also, some parameters are necessary to invoke the requested method by the virtual machine. Methods are written in Smali code in the following format:

$$(p1), Lpackage/Name/ObjectName; \rightarrow MethodName(parameter_1, .., parameter_N)Z$$

Where:

- ◇ $Lpackage$: represents the package.
- ◇ $Name$: is the name of the application.
- ◇ $ObjectName$: is the name of the object that uses the requested method.
- ◇ $MethodName$: is the name of the requested method.
- ◇ $Parameter_1...Parameter_N$: are the types of parameters expected by the requested method. Example for the String type, the parameter is in the format: $Ljava/lang/String$.
- ◇ Z: represents the type of the object returned by the method.
- ◇ $(p1)$: represents the register containing the value of $parameter_1$. In addition, a register is reserved for each parameter expected by the requested method.

- *Instantiation methods:* These methods are called in the Smali code by the invoke-virtual statement. These methods are called by the same format as the static methods, but in addition to the registers that contain the parameters expected by the method, another register is required as a reference to the object wherein the method is going to be called.
- *constructors:* The constructors are called by the Invoke-direct statement. The method call has the same format as the other types of methods, however, the difference is in the registers, where the first register is reserved as a reference to the new instance of object that will be created.

Each Smali file of the decompiled application will be inspected to locate the points of injections of the communicators. The injection is mainly focused on the API methods that are part of Android libraries and to allow control over these methods. While API methods are called in the same way as other methods, a tracking method is applied to extract only API methods. Once all these methods are localized, the necessary portions of code are injected.

2.4.2 API Tracking Method

At a first step in this method, all Smali files generated by the de-compilation phase will be scanned. A search is made in each file using the invocation keywords for each type of method. This phase allows us to locate all calls to the methods in the application. The next step is the selection phase which will allow extracting the subset of the API methods from the global set of methods found in the application. The algorithm developed in this study refers to a file that contains signatures of calls to the API methods. These signatures consist of the type of the method, the package, the name of the method as well as the class containing the method.

The present work targets the most common API methods in malicious applications as illustrated in Figure 2.2 Sen *et al.* (2017). Where, the red bar represents the percentage of use of these API methods by malicious applications, and the white bar represents their availability in benign applications. Once all the methods in the application are identified, this list is compared with

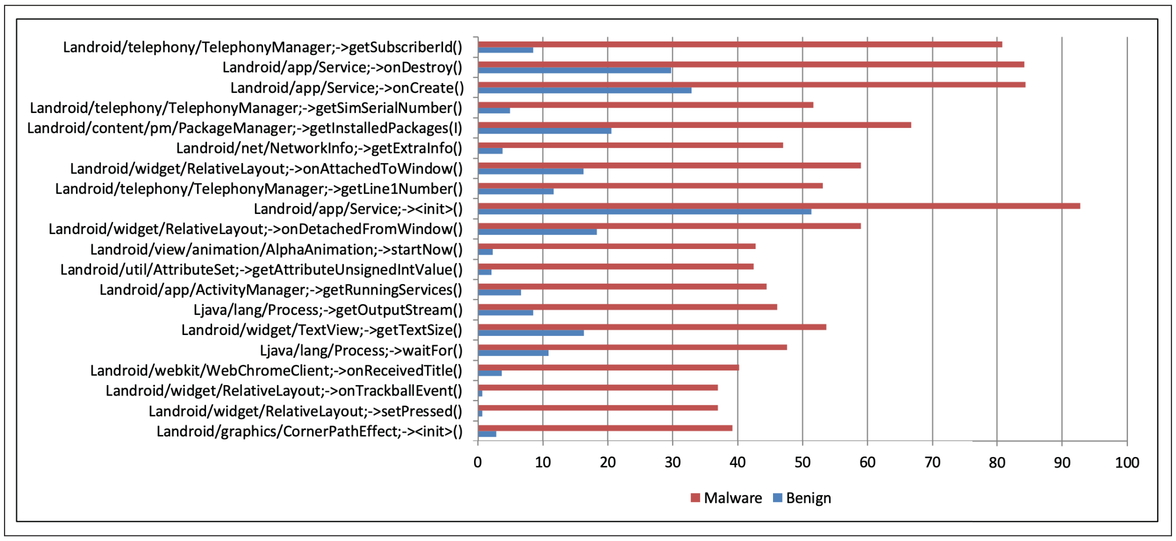


Figure 2.2 Top 20 Most common API methods in malicious applicationsSen *et al.* (2017)

the set of signatures contained in the reference file of the API methods. Thus, at the end of this operation, our algorithm developed keeps track of API methods contained in the application as well as their location in the byte code. It is important to note that our API method reference file is flexible as it accepts any modification or insertion of new API method signatures. The next step is to inject portions of code that allows communication with our controller before each call to an API method identified by the algorithm. Subsequently, the API methods will be forced to go through the application controller, providing several parameters allowing the controller to make the decision on the execution of the requested method. The communication between the enforced application and the controller is provided by several methods. These methods have been implemented and enclosed in a set of classes. However, before presenting these classes, which are added to the enhanced application, we present first the communication mechanism.

2.4.3 Communication Mechanism

The communication between the enforced applications and the application controller is mainly based on implicit Intents. When the application has to contact the controller, a new Intent will be created by the enhanced application. This intent will transfer some information to the application controller. Mainly, the parameters of the invoked API method, two arguments that allow the controller to identify the application that requests the method, as well as the type of the request. Therefore, the Intents are created and used in this study in two situations; when sending a request from the enhanced application to the controller and when sending the response by the controller. The difference between the two queries is specified by the Action field of the Intent. Where, the data field of the Intent, represents the type of data circulated by the Intent. The Extras field of the Intent contains the data to be circulated. The Flags field contains a set of variables that we use to modify the behavior of the intent in case of errors. When a response is sent by the controller, the enhanced applications must retrieve the data communicated by the intent. On the other hand, for Android system to communicate the intent to the right application, each enforced application must declare a filter in its manifest file. This filter will indicate to Android that this application can manage a precise type of actions. Therefore, our algorithm will modify the manifest file of the enhanced application, to add the node:

$$< intent - filter >$$

Also, will contain the action:

$$< This.From\_Monitor >$$

This node and the action, will allow Android to understand that this application can handle the intents received by the controller.

Figure 2.3 illustrates the communication mechanism between the controller and the reinforced applications. Our rewriting framework will inject the necessary code points to contact the controller before each attempt to execute an API method. To achieve this, several classes have been developed and written in Smali bytecode. Figure 2.3, also shows an example of the query control method, contained in the Smali class methods. This method is executed just before the API method is invoked. It should be noted that at the time of this execution, the enhanced application will create a new activity to manage communication with the controller. This activity is demonstrated by the smali communicator file and uses another Smali class called BroadCastReceiver. Then, the activated Communicator will generate a new intent, and send it to the application controller through the BroadCastRecevier class and wait for the response. Once the response is received, the Communicator activity consults the response to make the appropriate decision.

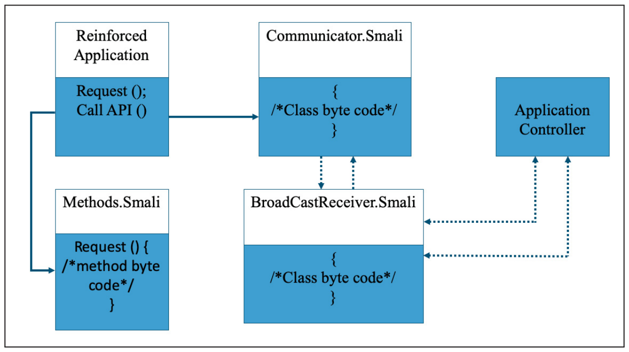


Figure 2.3 Communication Mechanism

Our rewriting algorithm will copy all the above-mentioned classes in a new folder that contains the decompiled version of the reinforced application. The following is mere details about these classes:

- *Methods.Smali:* This class contains two main methods for initiating communication with the application controller. The first method allows the management of the execution of the application. This method will be the first call of the enforced application. It takes as arguments the type of the request "RunApplication" as well as the signature of the application. The type of this request allows the controller to know that the application request is correct and can be executed, and the signature of the application allows the controller to identify the application in question. Once the controller receives this type of request, a check is performed using the security policies introduced by the user, then a response will be sent.
  The second method contained in this class allows the management of the execution of API methods. The call to this method is injected before each call to an API method. This method takes in arguments the type of the "RunAPI" request. Also, allows the controller to know that the application is attempting to execute an API method. Our algorithm identifies the arguments passed to the API method and will be sent to the controller via our control method.
- *Communicator.Smali:* This class is an activity that inherits from the *ActionBarActivity* class. It is instantiated by the control methods contained in the *Methods.Smali* class and used to manage communication with the controller. This class allows to represent the intent that contains the request to be sent to the controller, to wait for the answer and to take the appropriate decision according to the answer received. Three types of responses are identified in this study, namely:
  - ◇ Access Denied: If this type of response is received, the activity will force the application to terminate.
  - ◇ Access allowed: This type of response allows the application or API to run. The activity will then close, unblocking the application to run normally.
  - ◇ Temporarily denied access: If the controller decides that the resource requested by the application is temporarily inaccessible, this response will be sent to the application. The

application will be intercepted by the activity which, in turn, will return a second request to the controller after a period of time has elapsed.

In order for this class to handle communications with the controller, the activity must be executed in a new thread. In this way, the application can be temporarily suspended during the control time while communicating with the controller through the new thread. This communication is done using the *BroadCastReceiver.Smali* class.

- *BroadCastReceiver.Smali:* This class allows the management of intents, by inheriting the *BroadCastReceiver* class and implementing the *OnReceive* method. This method has been re-implemented so that it can identify the intents sent by the controller. The method will then retrieve the signature of the application sent by the controller and compare it with the signature of the application in question. If both signatures are identical, the response sent by the intent will be retrieved and sent to the *Communicator.Smali* activity to perform the necessary action.

2.4.4 Application Control Technique

To strengthen Android applications and to communicate with the developed application controller when needed, we have developed a technique that allows the automatic modification of the applications as well as the generation of the new reinforced APKs. This technique fulfills the following aspects:

- *Injection of resources into the application:* The Smali classes represented above have to be added to the enhanced application. To achieve this, the algorithm will create a new package in the directory containing the Smali files of the application. Thus, the necessary resources will be copied into this new package, while separating them from the other Smali classes of the application.
- *Changing paths:* After copying the necessary resources into the decompiled application, the developed algorithm in this study will modify the paths in the added Smali files. The classes developed in Smali are standard classes, ready to be copied into any APK. However, a very important step is to consider making a link between the application and the copied resources.

Therefore, our algorithm automatically adapts the calls of the instructions added in the Smali files of the enforced application, so that they can call the methods located in the control package that has been added. The signatures of the methods in the invocation instructions will then be modified. The following call to the *query()* method is taken as an example:

$$Invoke - static(p1), LCOM/Methods; \rightarrow query()V$$

This can be adapted to a specific application in the format:

$$Invoke - static(p1), LCOM/Facebook/w/z/ComAPP/$$

$$Methods; \rightarrow query()V$$

- *Alteration of registers:* While, the Android applications are developed to run on a Dalvik machine, also, this mechanism is based on the use of registers. Therefore, to add new calls to control the invoked methods, we required to add registers so that the methods can work. In addition, we should analyze the registers used in the targeted application to locate the register anticipated by each call. For example, the following call is added to the application code:

$$Invoke - static(p1), LCOM/Methods; \rightarrow$$

$$query(LAndroid/content/context;)V$$

  It is noticed that the query method uses a variable containing the context of the application. Thus, context must be provided by the register $p1$. If above line of code is added to the application, it will be necessary to make sure that the register $p1$ represents the context of the application. If this is not the case, the algorithm will execute an analysis of the Smali file in question to retrieve the register corresponding to the context of the application.
- *Editing the AndroidManifest.xml file:* The *AndroidManifest.xml* file is used to describe all the components of the application including: activities, Broadcast Receivers, permissions, libraries used in the application, etc. Hence, it provides information about the application to

the Android system.

During our reinforcement phases, several modifications will affect the enforced application. Some of these changes must be declared in the *AndroidManifest.xml* file. Thus, our developed algorithm, allows the automatic modification of the manifest file as following: First, it allows the declaration of the Communicator activity implemented by the *Communicator.Smali* class. Also, informs the *AndroidManifest.xml* file that this activity must run in a new sequence. Next, it allows the declaration of the *BroadCastReceiver* implemented by the *BroadCastReceiver.Smali*. Finally, it allows adding a filter called *This.from_Monitor* which provides the application the ability to recognize the actions of the intents sent by the controller.

- *Compiling the Enhanced Application:* Once all necessary changes are made to the enhanced application, it must be recompiled to generate a new APK. In our algorithm, we compile all the Smali files contained in the application using the APKTOOL tool. Then the set of Smali files will be recompiled in order to generate the .dex file that represents the entire application. The non-compiled resources will be added to generate a new APK representing the enhanced version of the application.
- *Signing the Enhanced Application:* Android system requires for all applications to be signed with a certificate, so they can be installed on Android devices. This certificate will be used by Android system to identify the author of the application. The signature of the application is located in its APK archive in the *WEB-INF* directory. Therefore, we have to sign the new APK generated by the compilation of the enhanced application and update the *WEB-INF* directory. At this step, we get the new enhanced application and its ready to be installed on a device and communicate automatically with our centralized applications monitor.

2.5 Centralized Control

The second part of the present work presents the implementation of a third-party application, which enables centralized control of the actions performed by the enhanced and installed applications. For this purpose, the application controller can make the necessary decisions by modifying the behavior of the applications if necessary. This controller is mainly based on context-aware security policies introduced by the user to make the appropriate decisions for each action performed by the enforced applications. Also, users will have accesses to the list of all enforced applications through the controller's user interface, where they can define set of rules for each application individually, any set of API methods or/and set of applications. Therefore, when an application tries to execute any API method, it sends a request of type "Execution" to the controller. Then, the controller will make the appropriate decision based on the predefined access control policies.

Each enforced and installed application must be registered in the application controller. The controller ensures this mechanism, so when it receives a request for execution from an application, it will check if the application is already registered in its list. If the application is not already registered, the controller will register this application while keeping track of its signature. The controller uses a catalog file that contains a list of registered applications with their signatures. This file is loaded at the start of the controller in a data structure as a "table" and will be retained throughout the execution of the controller. Figure 2.4 illustrates the process of registering an application to the controller.

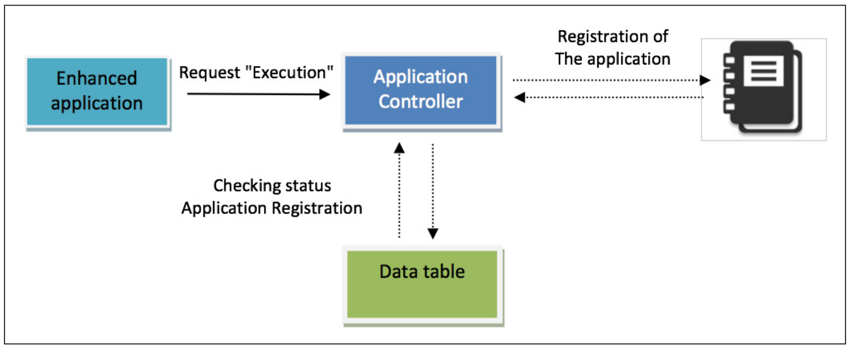

Enhanced application
Request "Execution"
Application Controller
Registration of The application
Checking status Application Registration
Data table

Figure 2.4   Registration of an application to the controller

2.5.1   Security Policies

The user will define the security policies to control the behavior of the enhanced applications. The security policies are primarily defined using rules and conditions. The identification of the conditions must be carried out by the user. These conditions are primarily used to create security rules. We have defined five types of basic conditions which will be used to construct more general conditions. These basic conditions are:

- Time: This condition allows the user to define a time interval during which an application set or API methods can be executed.
- Date: The date condition is used to manage the execution of applications or API methods on specific dates (i.e., in a dates range).
- Location: This condition allows the user to prevent API applications or API methods from running in defined locations. These locations are based on GPS coordinates.
- Applications: This condition allows the definition of a set of applications, using the logical operators "OR" and "AND". If the user wants to apply a security rule that depends on a set of applications, then this condition would be in the format "APP_1 AND APP_2 AND APP_3, etc.".
- Battery: This condition allows the user to manage the execution of applications or API methods while taking battery status into account.

Furthermore, the application controller allows the combination of conditions using logical operators such as "AND" and "OR". Then the user can define any logical scenario as shown in Figure 2.5.

2.5.1.1 Creating Policy rules

The security rules are primarily defined using the conditions created by the user. These security rules are combined to define the scenario desired by the user. If, for instance, the user wants to apply the following security rule: "It is prohibited to run a set of applications if the device user is in a meeting." In order for this security rule to be created, the user must first create the conditions that define the rule. If, for example, the meeting is held twice a week, from 2 pm to 3 pm, every Monday within the company. Several conditions must be defined, namely:

- C_1: The time interval type condition containing the start and end time of the meeting.
- C_2: Date condition containing the date of the meeting.

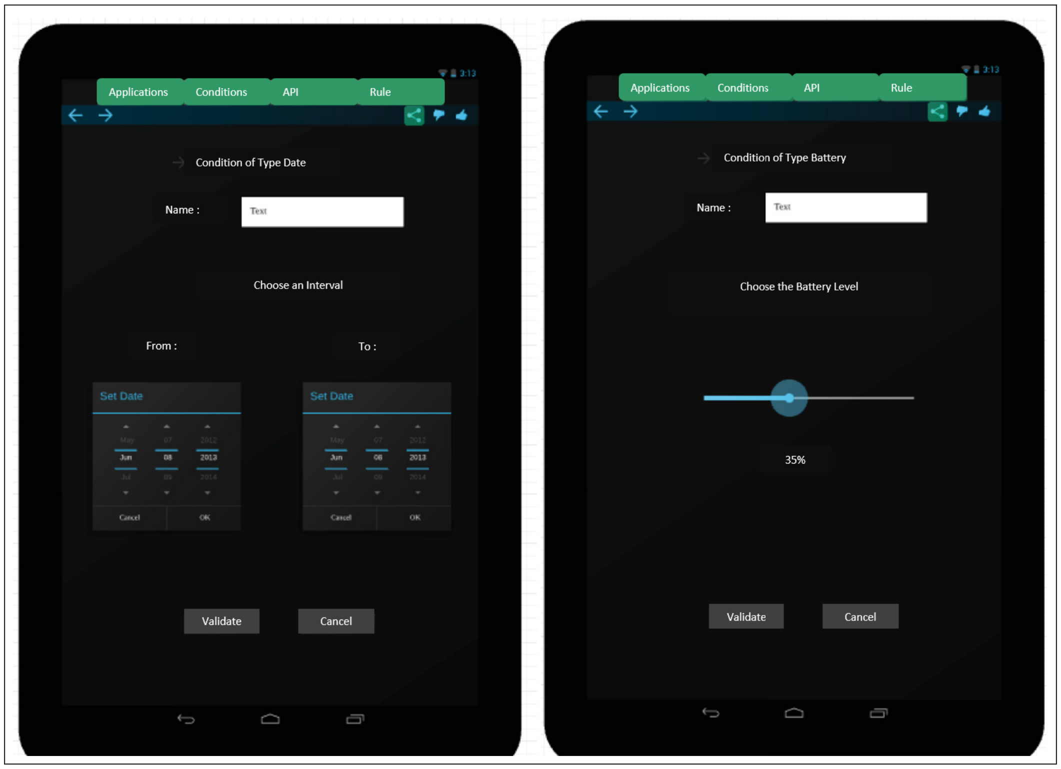


Figure 2.5 Examples of creating conditions

- Location type Contains the GPS coordinates of the company where the meeting room is located.
- C_3: Type condition Location containing the GPS coordinates of the company where the meeting room is located.
- C_4: Application-type conditions containing the set of applications, eg.: ((Application 1 AND Application 2) OR Application 3 OR Application 4). These four conditions are taken into account in order to define the rule that prevents the scenario defined by the user. It should be noted that no application or set of applications of rule C_4 is executed if the three conditions (C_1 & C_2 & C_3) are satisfied.

2.5.1.2 Applying the Policies rules

Security rules can be built to define runtime scenarios. These policies are not only dependent on applications, but also on API methods. Our developed application controller provides a flexible way to construct these rules based on API methods. The interface of the application controller displays all applications installed on the device which are already registered within the controller. In this case, the user can either manage the API methods specific to each application or manage them to all applications. When the user chooses a specific application (also valid for all applications), the API methods used by this application will be displayed. This list of methods is retrieved during the reinforcement phase of the application. Our framework identifies the API methods from Smali files, and the list will be sent to the application monitor during the application authentication phase. The user can prohibit the execution of an API method by the chosen application (or set of applications) by deactivating the method using the developed user interface. If the application tries to execute a user-deactivated method, the application controller will prevent this action because the "All Time" condition is enabled by default. The user can customize the rule by changing the "All the time" condition and replacing it with a specific condition already created. If, for example, the user wants to apply the following security rule: "An application cannot record a video clip if the device user is in a meeting". The user will create the necessary conditions to define this rule. These conditions will be combined to define a more generic condition that will be applied to the user-deactivated video recording API method. In this way, if the user is not in a meeting, the application controller will allow the API method of video recording to be executed by the application in question (or by the set of applications). Several rules can be defined, depending on the context defined by the conditions. These rules can then be activated or deactivated by the user as required. In addition, the application controller provides the user with the ability to define API methods that are exclusive to one or a set of applications. The user assigns API methods exclusive to the chosen applications via the controller's graphical interface. Thus, all applications that do not belong to the defined set cannot execute these API methods. In this case, the user could always apply conditions to define scenarios that are more specific to this exclusion. These conditions are created in the same way explained earlier.

2.5.1.3 Saving the Policies Rules

Once the security rules are defined by the user, a file will be created for each application which contain all security rules. The controller uses two other files, the first contains the conditions created by the user and the second contains the security rules that are applied to all applications. These files are updated with each modification made by the user. The security rules are saved using the following syntax: Action 1 to be prohibited, Action 2 to be prohibited, etc.: Condition 1 AND / OR Condition 2 AND / OR, etc. " This rule states that action 1 and action 2 are forbidden if condition 1 AND / OR condition 2 is verified. Two real examples can be cited in this case:

- Exec, SendTextMessage: C1 AND (C2 or C3).
- BNC, RBC: Facebook or ScreenShooter.

The first rule prevents the execution of the two API methods: Exec and SendTextMessage if condition C1 and one of conditions C2 or C3 are checked. As for the second rule, it prohibits the execution of both BNC and RBC applications if one of the Facebook or ScreenShooter applications is running.

2.5.2 Reference Tables

The application controller is based on the security rules introduced by the user and saved in backup files. However, in order to speed up processing, these files are not solicited whenever a request is received by a reinforced application. The application controller loads the backup files when it is started then proceed to the processing stage, where all rules will be analyzed. The algorithm allows the controller to break down rules into basic elements. These are mainly API methods, application names, and conditions. Once the rules are analyzed by the application controller, processing will be performed by combining the logic of all user-activated rules. At the end of this process, the application controller will generate array data structures. These reference arrays will be used throughout the execution of the controller. As shown in Figure 2.6, the first reference table will contain the list of API methods prohibited by all applications. As for the second reference table, it will contain in each box indexed by the name of an application; A structure represented by a Boolean which makes it possible to know if the execution of the application is allowed or not. The second table will also contain the list of API methods prohibited by the application in question.

When an application sends a request to the controller, the controller consults the two reference tables to see if the application where the requested API method is allowed to run. This information will be transmitted to the application in question. The reference tables are updated

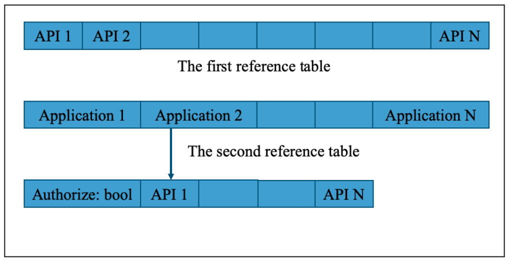


Figure 2.6 Reference Tables

by the application controller in two cases. The first case is the modification or addition of the security rules by the user. The second case is the change of state of one or more conditions. For example, if a security rule prevents access to an application that is dependent on a time-slot condition. When this condition is verified, that is, when the current time will be included in this interval, the application must be prohibited. Thus, the Boolean in the second reference table will have to be set to "false".

To perform this mechanism, the controller developed in this study defines certain services allowing the control of the conditions and to be up to date. To achieve this, we have implemented a service for each type of condition, such as (Time interval, Date, Location, Application, Battery). Each of these services controls the conditions belonging to its type. When a user-activated security rule depends on one or more conditions, they will be sent to the various services. Each service intercepts the conditions homogeneous to its type:

- Time Service: Intercepts the time interval conditions and controls the time change.
- Date service: Intercepts date conditions and controls changes in dates.
- Location Service: Intercepts location conditions and controls changes in GPS coordinates.

- Application Service: Intercepts application-like conditions and controls application execution.
- Battery service: Controls the battery status.

When one of the five services detect a condition changes state (valid, invalid), the service sends a notification to an observer class previously implemented. This allows the monitoring of the conditions as well as the updating of the two reference tables. This class is registered as an "observer" of the events sent by the four services. Figure 2.7 illustrates the control mechanism of conditions by the application controller.

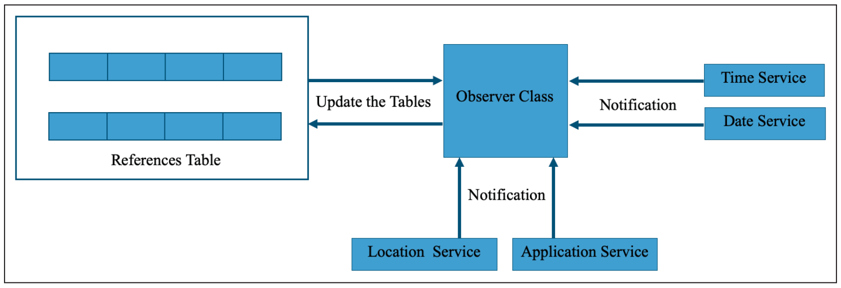


Figure 2.7 The Mechanism of controlling conditions

Once the application controller makes the appropriate decision to the query sent by an enhanced application, referring to the reference tables. This response must be transferred to the enhanced application. The controller will then create a new intent, and apply the action *This_From_Monitor*. In this way, the intent could be intercepted by the reinforced applications. The transferred response via the intent consists of three fields: The first field is the signature of the application requesting access to the resource. This application will then retrieve this signature and compare it with its own, in order to verify if the message is intended for it. If the signature is identical, the application will retrieve the second field which represents the number of the request sent by this application. In this way, if the application requesting access consists of several threads that solicit the controller at the same time. This application will know which answer belongs to which query sent. When the Thread that sent the request is identified, the enhanced application will retrieve the decision. The decision represents the right to access the requested resource

(Access denied, Access authorized, Access temporarily refused). The user, will be notified if the application will be closed in accordance with the decision of the application controller.

2.5.3 Adding API methods

As previously mentioned, this work focus on the top 20 most common API methods in malicious applications. Therefore, the developed application controller only processes these methods. However, the user can always add more methods to intercept using the graphical interface of our controller. The user then begins the call to the API method in JAVA language. The monitor then takes care of the conversation from the java instructions to the Smali instruction and adds the requested call to the list of calls already entered. In this way, the new call will be intercepted by the framework in the reinforcement phase.

2.6 Experimental Analysis

We have conducted various tests in order to evaluate our rewriting framework and the application controller on a set of real-world Android applications. The experiments were carried out on a Samsung Galaxy S3 phone running the Android operating system 4.1.1, with 2G of RAM, a dual-core processor running at 1.5 GHz and an internal memory of 12G. In this study, the framework variant, running on the web server, is installed on a server running under the Linux operating system Ubuntu 14.04, including a core processor i5-4200U, 8GB DDR3 RAM. The application runs under the Apache 2 web server.

2.6.1 Rewriting Effectiveness

To evaluate the efficiency of our application rewriting framework, we performed tests on a dataset of 950 Android applications. These applications were randomly selected from Google Play and added to 450 malicious applications Mila (2011). Furthermore, all selected applications have been automatically rewritten using our Framework. The malicious applications used in our study belong to different categories such as: games, music applications, social networks, etc. From malware characterization viewpoint, these applications contain malware belonging to different families. Also, the size of these applications varies between 1 MB and 50 MB.

Table 2.1 illustrates the rewriting success rate of our framework when applied on the aforementioned applications dataset. Note that, the logs provided by our framework have been retrieved in order to analyze error messages in the case of failure during the recompilation phase. After the analysis of these logs, it was found that the errors are generated by apktool Tumbleson (2010) and have no relation to the modifications made to the applications.
Hence, in some cases (around 0.23%), Apktool does not recompile an application even if not modified. Normally, the generated errors are related to the resources of the application.

Table 2.1 Applications rewriting success rates.

Applications Type	Number of applications	Success of rewriting rate
Apps from Google Play	950	99,78% (948)
Malicious applications	450	99,77% (449)
All applications	1400	99.78% (1397)

As shown in Table 2.1, our Application Rewriting Framework provides a success rate of 99.78% during the recompilation phase. In other words, only the compilation of 3 applications failed among a set of 1400 applications. This success rate is considerably high compared to other studied approaches. As an example, the study presented in El-Harake, Falcone, Jerad, Langet & Mamlouk (2014) which is an application rewriting approach based on an AspectJ compiler, shows a success rate of 90% during the recompilation phase. As another example, the "Urasium", is an approach proposed in Xu *et al.* (2012) , uses the same compiler apktookTumbleson (2010) adopted in the development of our Framework. This approach ensures a success rate of 96%; which is similar to the compilation rate of the Capper approach presented in Zhang & Yin (2014).

2.6.2 Rewriting Time

The application rewriting approach adopted in this study proposes two alternatives. The first being a third-party Android application while the second being a solution implemented on a web server.

In the following, we present the experimental results related to the processing time required for the rewriting of the applications by the two variants of the Framework. A set of 1350 applications was rewritten by the two variants where the size of these applications varies between 1MB and 100MB.

Table 2.2 shows a sample of ten applications where the time to rewrite each application by the two Framework variants has been measured. The table also illustrates the difference in execution time between the variant implemented on the mobile device and the one hosted on the web server.

Table 2.2 Applications rewriting time.

Applications	Applications size(MB)	Run time on server (s)	Run time on device (s)	Run Time (Server vs Device) (s)
The world	1	40	62	22
ConnectTheDots	1	30	50	20
MassengerPlayer	4	79	104	25
ITubePro	4	38	61	23
VLC	12	60	85	25
PafLeChien	12	75	94	19
GamerZ	26	76	96	20
SubwaySurf	28	79	96	17
TempleRun	48	81	101	20
ClashRoyale	84	75	101	26

It can be seen from Table 2.2 that the rewriting time does not vary according to the size of the applications, this can be explained by two reasons: The first reason is that apktool recompiles only bytecode files, where the resources of the application like images, XML files, etc. will not be taken into account. As a result, an application may be of considerable size due to non-compiled resources. However, this one does not include a large number of Smali files, which accelerates its compilation and decompilation by apktool. The second reason is that our framework deals only with Smali files. Therefore, the rewriting time only takes the size of the application. Figure 2.8 illustrates the ten samples of table 2.2 rewriting time for each variant of our Framework as a function of the size of the applications. It can also be noticed that the application rewriting framework requires more time on the mobile device than on the web server. This is due to the device's limited resources.

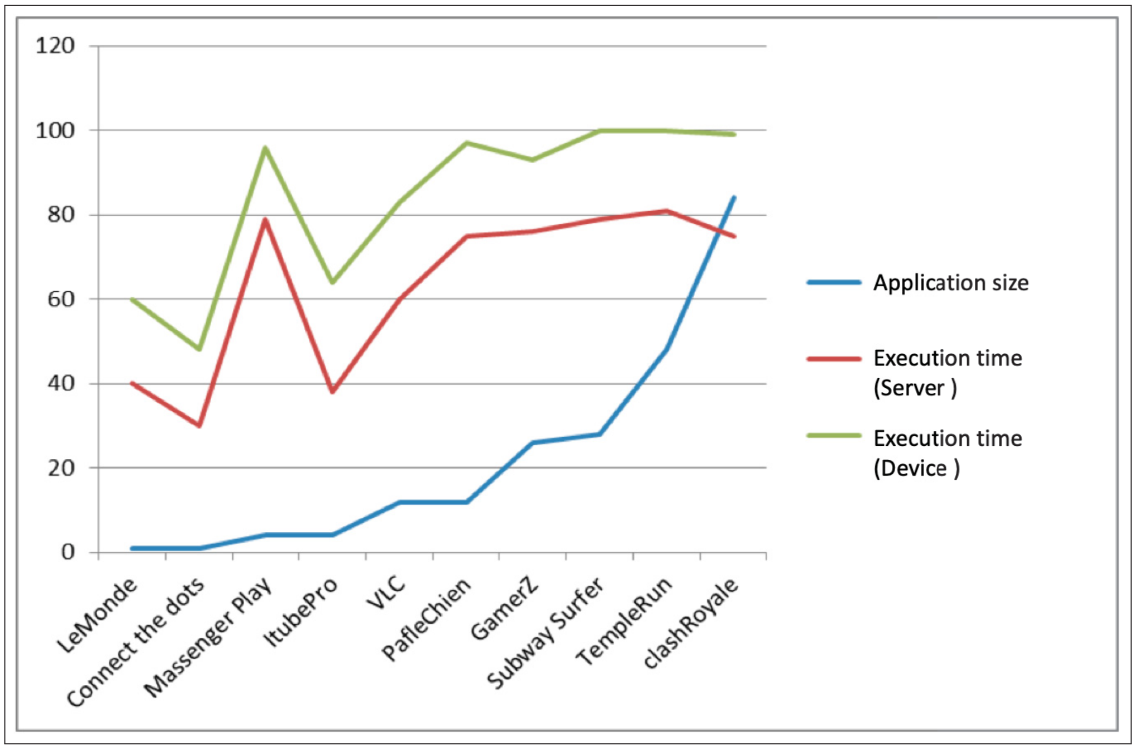


Figure 2.8 Variation of the rewriting time according to the size of the applications

In the present study, for all tested 1350 applications, the average of the time difference between the two variants was calculated and it is around 21s. Particularly, the framework executed on the mobile device requires, on average, 21s more than the variant implemented on the web server. In addition, for all the 1350 applications, the average of rewriting time was also calculated. The latter is 51s for the web server variant and 72s for the variant installed in the device. Figure 2.9 illustrates the rewriting time for all applications and for each of the two variants. Further, this figure shows that only 20% of the 1350 applications require a rewriting time of the order of 80s on the server and 100s on the mobile device. Other applications require less rewriting time.

the work Zhang & Yin (2014) has reported an average processing time of 60s for the Capper tool. However, Capper does not include Dex2jar decompilation time, which may take several seconds.

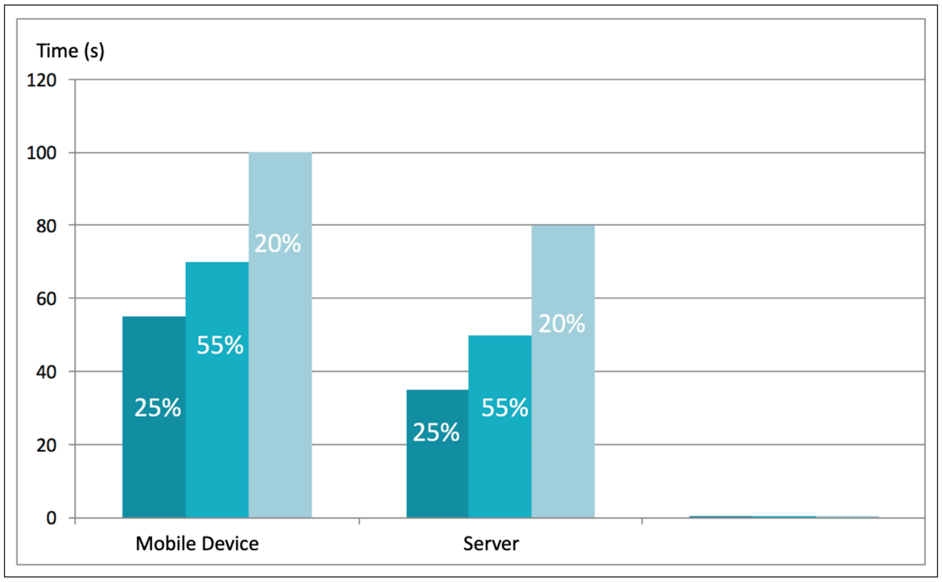


Figure 2.9 Time to rewrite all applications

This task represents on average more than 40% of the rewriting phase of our Framework. The time for the Framework to be enhanced could then be reduced by 40% if the decompilation time is not taken into account.

2.6.3 Application Size

Mobile devices suffer from source constraints. It is therefore important to measure the impact of application rewriting on the system. Note that apktool Tumbleson (2010), being the tool used by our Framework to compile and decompile applications, does not maintain the same size of the application after its recompiling. In fact, the new compilation can increase or decrease the size of applications. Thus, in order to have a better illustration of this size variation, we performed an experimentation where each application was decompiled using apktool and then recompiled without applying any changes to it. The results of this operation are illustrated in Table 3.2. In the present study, the code portions added by the Framework vary from one application to another. First, our framework injects in the rewritten application the Smali classes needed to communicate with the application controller. It then traverses the bytecode of the application and proceeds to insert the instructions calling the methods of these classes according to the

Table 2.3 Sizes of applications rewritten by the Framework.

	Original	Recompiled with Apktool (o)	Rebuilt (Framework) (o)	Reinforced-Recompiled (o)	Added Size (o)	Percentage of byte code size added	Percentage of added size
FBMessenger	23494355	23671070	23 673 491	2421	179136	0.01%	0,76%
Traffic Rider	94019694	94472732	94 502 477	29745	482783	0.03%	0,51%
World Chef	64631472	65798804	65 981 462	182658	1349990	0.28%	2.09%
BatterySaver	8295601	8344668	8 357 203	12535	61602	0.15%	0,74%
Color Switch	14051589	14099235	14 102 189	2954	50600	0.02%	0.36%
Instagram	11322148	11877285	11 996 209	118924	674061	1.05%	5,95%
Netflix	15761539	15993453	16156718	163265	463265	1.03%	2,51%
SpotifyMusic	27368605	27985605	27 986 774	1169	51020	0.004%	2.26%
Snapchat	34545629	34629675	34688877	59202	59202	0.17%	0,41%
Piano Tiles 2	18869921	18912502	18 935 851	23349	65930	0.12%	0,35%

type of API requested. As a result, some applications may require more byte code to be added than others. This variation affects the size of the applications. In order to get a clearer idea of the variation of sizes, the set of 1350 Android applications were rewritten using our framewok, observing their sizes before and after the rewrite phase.

Table 3.2 shows the different sizes for a set of ten applications. Also, shows the difference in size between the original APK of the application and the APK generated by the Rewriting Framework. This value includes the size change brought by apktool. The percentages shown in Table 3.2 represent the percentage of size added to each application (by the Framework) relative to its original size, including the size added by apktool, as well as the percentage of size added by the Framework excluding the size added by apktool. Furthermore, it shows the size of the byte code that allows call control to API methods For the set of 1350 applications selected, the average added size was calculated. The latter is 98822 bytes (0.098 MB), which is a relatively small size compared to application sizes. In addition, the average percentage of size added for all 1350 applications was 1.2%. Benjamin Davis in Davis *et al.* (2012) estimated the percentage of size added by I-arm-droid to 2%. While Xu et al. in Xu *et al.* (2012) estimated the size added by Aurasium to 52000 bytes. Figure 2.10 illustrates the different sizes of the ten applications studied, in each of the three states: original, after recompilation with Apktool and after rewriting by our Framework.

Figure 2.11 shows the different sizes (original, after compilation and after rewriting) for a set of 134 applications processed. From the figure, we notice that the size added by our Framework is very small compared to the original size of the applications.

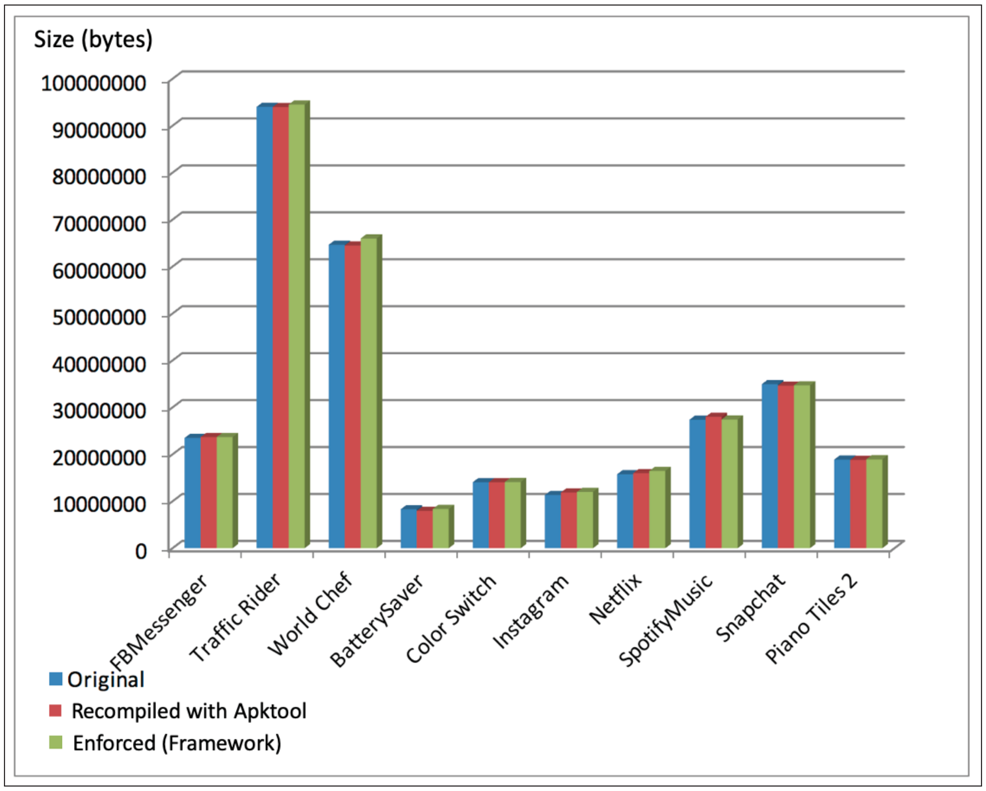


Figure 2.10 Sizes of ten applications rewritten by the Framework.

2.6.4 Inserting Control Calls

The Application Rewriting Framework developed in this study has two main tasks, namely, the addition of Smali classes for communication with the controller and the insertion of API control calls (requesting classes). To verify the integrity of these insertions, ten applications have been rewritten by the Framework. Thus, the versions generated by the Framework were analyzed manually in order to check whether the control calls were added before each call to the API methods in the entire byte code of each application. Our analysis, found that the algorithm of insertion of the control methods developed in our study ensures a success rate of 100%. That is, for each call to an API method, the framework has added the call to the API-specific control method. As already explained in the previous sections, the evaluation is based on the 20 most common API methods in malicious applications Sen *et al.* (2017). Table 2.12 shows the APIs found in the ten randomly selected malicious applications. The table contains 12 API

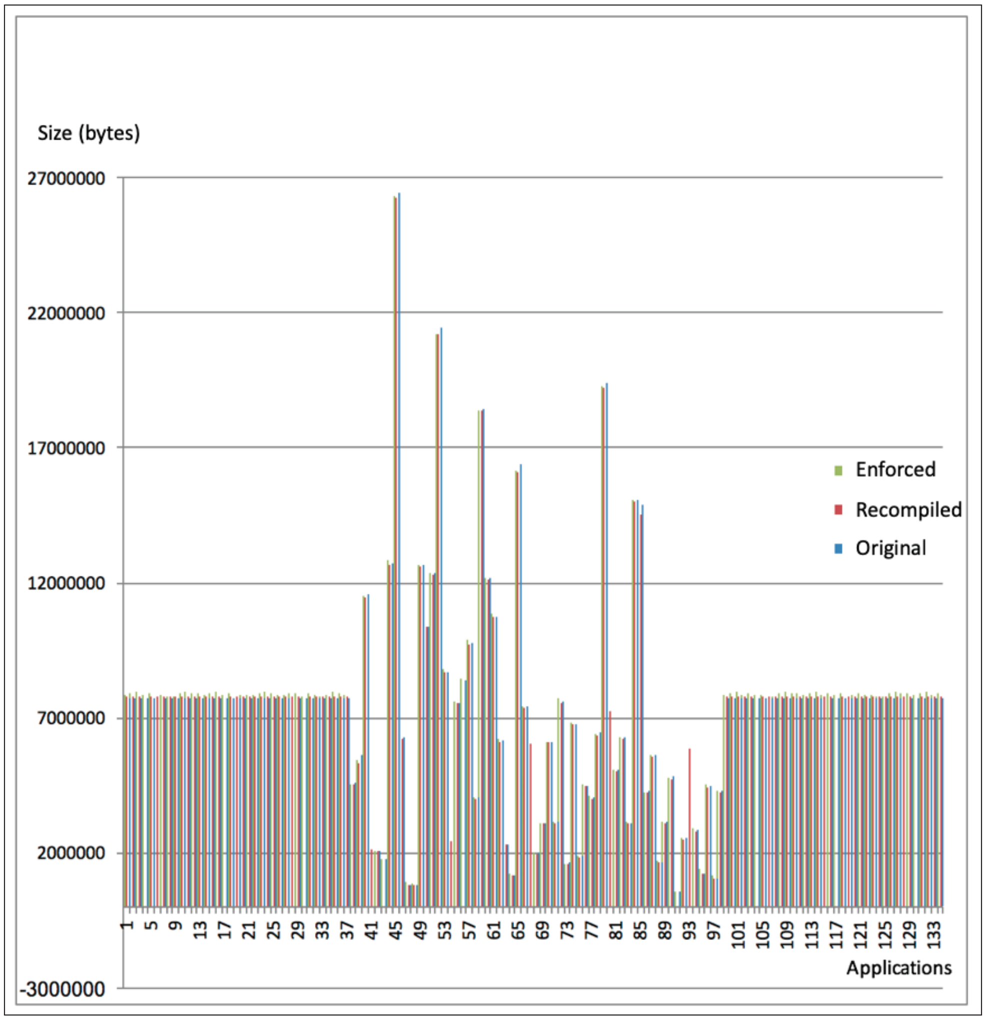


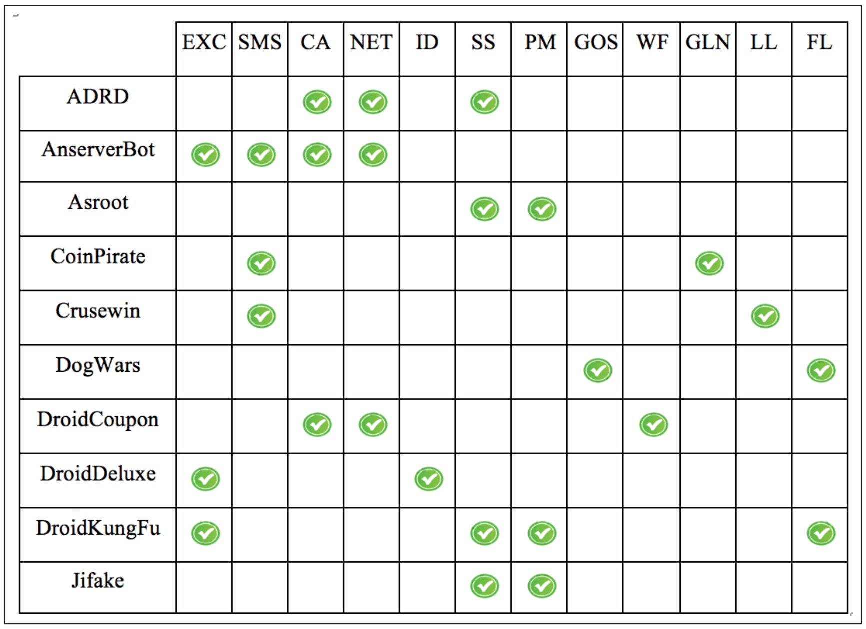
Figure 2.11 Sizes of 134 applications rewritten by the Framework

methods solicited by the applications in question. The abbreviations of these methods and their explanations are as follows:

- EXC: Ljava / Lang / RunTime → Exec (), allows you to execute a shell command.
- SMS: Landroid / telephony / SmsManager→ SendTextMessage (), allows you to send a text message (SMS)
- NET: java.net.HttpURLConnection→ openConnection (), to open an HTTP connection.
- ID: Landroid / telephonyManager / getDeviceId (), retrieves the device identifier.
- SS: Landroid / content / Context→ startService (), starts a service.

- PM: Landroid / content / pm / PackageManager→ GetPackageManager (), returns an instance of packageManager containing information about the global package.
- GOS: Ljava / lang / Process → GetOutputStream (), returns the OutputStream connected to the standard InputStream.
- WF Ljava / lang / Process→ WaitFor (), allows to wait for the end of the execution of a process.
- GLN Landroid / telephony / TelephonyManager→ GetLine1Number (), returns the phone number of the first phone line.
- LL Landroid / content / pm / ApplicationInfo→ LoadLabel (), retrieves the current text tag associated with an object.
- FL Ljava / io / FileOutputStream→ Flush (), allows to write all the data in memory to the output (OutPutStream, etc.)

The tests performed were performed on the API methods cited in the previous sections. Methods not shown in Table 2.12 have not been found in all ten selected applications. Manual analysis to search for them confirms that these methods do not exist in the application code.



	EXC	SMS	CA	NET	ID	SS	PM	GOS	WF	GLN	LL	FL
ADRD			✓	✓		✓						
AnserverBot	✓	✓	✓	✓								
Asroot						✓	✓					
CoinPirate		✓								✓		
Crusewin		✓									✓	
DogWars								✓				✓
DroidCoupon			✓	✓					✓			
DroidDeluxe	✓				✓							
DroidKungFu	✓					✓	✓					✓
Jifake						✓	✓					

Figure 2.12 API Methods Used by Malicious Applications

2.6.5 Controller Functionality

To ensure that the changes made by the Framework do not interfere with the functioning of the applications, a set of twenty applications have been rewritten by the Framework. Also, the applications are executed in the test device while observing the logs generated by these applications and the controller. The logs generated by the applications are mainly traces of the API calls requested by these applications. The controller, in turn, generates a set of logs representing the intercepted API calls. The security policies implemented in the controller have been disabled at the beginning to observe the normal operation of the applications. Applications installed in the device run their events with Monkey Developers (2015). This is an application running Android to generate random user events, like buttons clicks or different interactions with other components of the application. In this study, the Monkey application was used in order to stress the applications in a random way. The logs helped to verify that the applications are functioning normally or stop working due to the changes made. Based on these logs, we have observed that the modifications made by our Framework of rewriting do not in any way influence the functioning of the applications. In addition, the logs generated by the applications were compared with the logs of the controller to ensure that all calls to the API methods solicited by the applications were indeed intercepted by the controller.

2.6.6 Testing the Security Policies

The application controller developed in this study is based on a set of security policies introduced by the user to control applications installed on the Android platform. Functional tests have been carried out to ensure that our controller analyzes and implements the security policies. The tests were carried out on a set of applications rewritten by the Framework. These were installed on the test device and their events were launched in a random manner to Monkey Developers (2015). A set of security policies were chosen and implemented in the controller. Thus, the logs, generated by the controller and the applications, were analyzed. The aim of this maneuver is to ensure that the API methods requested by the applications have been checked in accordance with the security policies. The security policies introduced in the controller depend on the applications being executed as well as on the API methods requested, the time, date, location, and battery constraints. Here are some security policies used in testing:

- Prohibition of launching application A if: (Application B and Application C) or (Application B and Application D) are executed
- API_1 and API_2 prohibited by applications A and B, if: (API_1 is used by any application, and AP_2 is used by applications (D or G)).
- API_1 cannot be used by all applications if: ((API_2 is used by application A and geolocation coordinates = X), or (API_3 is used by applications C and D and F and (Date interval = D or Time interval = T)).
- Prohibited to use API_1 and API_2: if (Battery < 50%, and Application A is executed).

The list of policies presented illustrates only 4 policies on a set of 30 security policies implemented for carrying out the tests. Based on our analysis of the logs, the behavior of each application could be identified. It has been found that the controller manages all the applications executed and makes it possible to require a check on any call to API methods in order to ensure that the API methods comply with the user-defined security policies.
The following is a case study for the introduction of the security policy: "It is forbidden for any applications to take a screen-shot if the RBC and BNC banking applications are executed, and that from Monday to Thursday, from 1 pm to 4 pm" First, the security policy in the application controller is introduced by defining the following three conditions:

- T_1: time interval from 13h to 16h
- D_1: interval from Monday to Thursday
- E_1: The BNC application is executed or the RBC application is executed

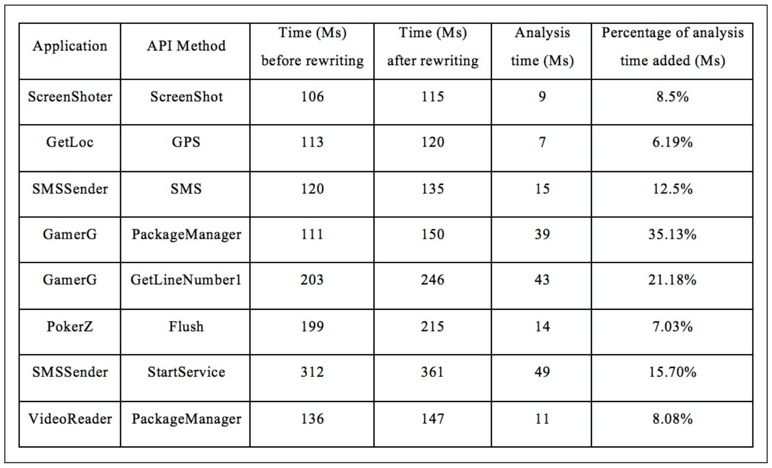
Thereafter, the global condition is defined:

- G: T_1 & D_1 & E_1

The "ScreenShot" API is subsequently banned on all installed applications if condition G is verified. The "ScreenShotTaker" application has been developed to take a screenshot every 15 seconds. Thus, this one has been rewritten using our Framework. We save the logs during the execution of the controller as well as the two applications: ScreenShotTaker and RBC. The controller and the application "Screenshoter" are launched at 12:59 pm. Thirty seconds later, the "BNC" application was executed. Also, the "ScreenShotTaker" application has not been stopped since condition T1 is not checked.

2.6.7 Controller Run Time

The application controller intercepts any call to an API method to analyze it. An analysis time is added to the execution time of each API method for the rewritten application. To measure this execution time, six applications using different API methods were created. Thus, the execution time required for the execution of each method has been calculated. In this study, instructions to calculate the execution time are placed in the byte code of the applications before and after each call to an API method. To calculate the execution time added by the analysis phase, six applications are launched initially by requesting a set of API methods (before rewriting). Table 2.13 shows the difference between the execution time of API methods before and after rewriting applications by our framework. It can also be noticed through this table that for this set of six applications, the analysis time varies between 7ms and 49ms. This represents a relatively negligible time. The percentage representing the analysis time added when calling API methods was calculated and shown in Table 2.13.



Application	API Method	Time (Ms) before rewriting	Time (Ms) after rewriting	Analysis time (Ms)	Percentage of analysis time added (Ms)
ScreenShoter	ScreenShot	106	115	9	8.5%
GetLoc	GPS	113	120	7	6.19%
SMSSender	SMS	120	135	15	12.5%
GamerG	PackageManager	111	150	39	35.13%
GamerG	GetLineNumber1	203	246	43	21.18%
PokerZ	Flush	199	215	14	7.03%
SMSSender	StartService	312	361	49	15.70%
VideoReader	PackageManager	136	147	11	8.08%

Figure 2.13 API Methods RunTime

Figure 2.14 shows the execution time of the API methods by the six applications studied before and after the application rewriting. We note that the execution time added by the analysis phase of our application controller is relatively small compared to the time of execution of the call without control. It represents 14.37% for all eight API methods illustrated in Figure 2.14. In addition, tests were carried out on a set of 20 API calls in order to calculate the execution time overhead added by the application controller and which is 20% relative to the execution time of the API call without control. Xu et al. in Xu *et al.* (2012) have also been interested in controls of API methods, estimating the overhead added by Aurasium to 35%.

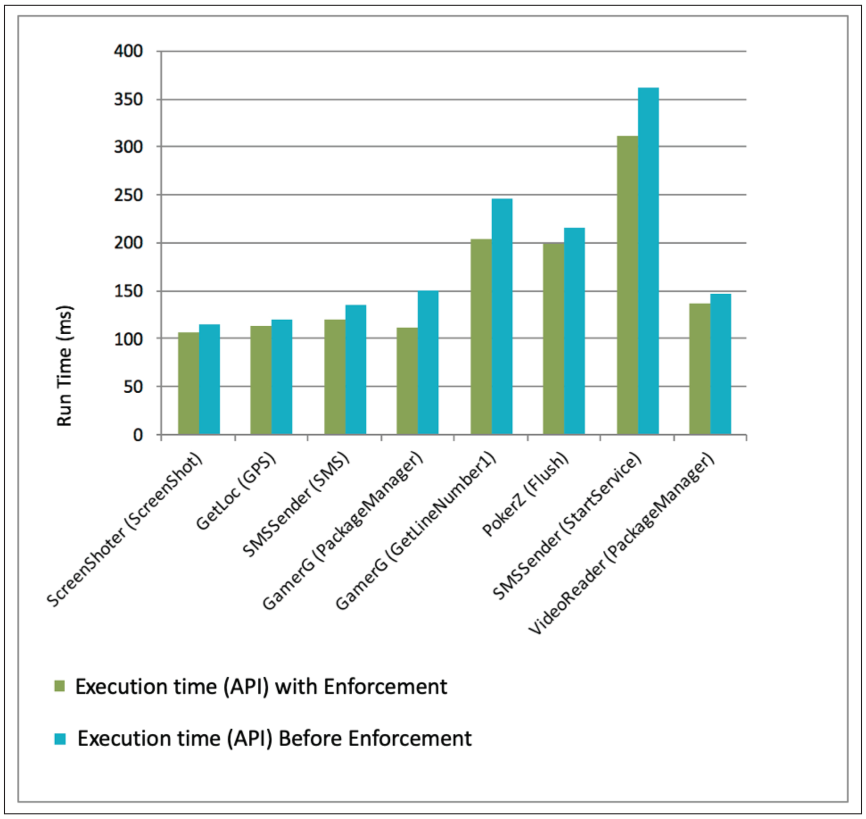


Figure 2.14 API Methods Execution Time

2.7 Discussion

The approach we have proposed to control the execution of API methods consists of two main axes, namely the Application Rewriting Framework and the Application Controller. The conducted in-depth experiments have proved the efficiency of both. Our framework ensures a success rate of rewriting of 99.78%, which is considerably high compared to the other approaches studied. The study presented by El-Harake *et al.* (2014), which is an application rewriting approach based on an AspectJ compiler shows a success rate of 90%, whereas Urasium Xu *et al.* (2012) and Caper Zhang & Yin (2014) guarantee a rate of 96%.

Application rewrite time has been calculated for both variants of our Framework. The latter is 51s for the variant executed on the web server and 72s for the variant implanted in the device. The authors in Zhang & Yin (2014) illustrate the processing time of Capper. The latter being 60s. However, the latter do not include decompile time by Dex2jar, which may take several seconds. This task represents on average more than 40% of the rewriting phase of this study. The time for the Framework to be enhanced could then be reduced by 40% if the decompilation time is not taken into account.

Mobile devices suffer from the constraint of limited resources. It is therefore important to measure the impact of application rewriting on the system. We have therefore calculated the average size added by our rewriting framework, that was only 98822 bytes (0.098 MB), which is a relatively small size compared of the application sizes, and which represents an average percentage of 1.2% To the original size of the application. The work in Davis *et al.* (2012) estimates the percentage of size added by i-arm-droid to 2%. While the work in Xu *et al.* (2012) estimates the size added by Aurasium to 52000 bytes

Our application controller requires control over any call to API methods, so an analysis time is added to the execution time of API methods. Tests have therefore been carried out on the set of 20 API calls studied Sen *et al.* (2017) in order to calculate the execution time overhead added by the application controller and which represents 20% of the time of the PLC non-control call. It should be noted that Xu *et al.* (2012) have, in turn, also been interested in controls of API methods, who estimated the overhead added by Aurasium to 35%.

Our Application Controller provides a simple and effective way to introduce security policies. A graphical interface is available for the user, based on the decomposition of security policies in basic conditions, and offers the possibility of combining the conditions in order to define complex security rules. Moreover, after studying the literature, our solution is the only one able to overcome any collaboration between applications with the intent to create malicious contexts. The security policies introduced in our controller can be defined on a set containing several API applications and methods. Because the controller is a third-party application, it can monitor and stop any attempt to collaborate unhealthily between applications. Unlike the approaches studied, our solution could stop attempts to execute native codes by malicious applications, since the execution of these codes must solicit API methods. The definition of security policies that depend on these API methods can be an effective solution against this type of attack. Table 6 provides a comparison of our approach with the different approaches studied.

2.8 Conclusion

While being the dominant operating system and due to its popularity, Android is one of the main targets of attackers. In this work, we focused on android applications that are malicious when only running in particular context. We proposed a rewriting framework that provides an instrumentation technique for security policies enforcement in android applications. The framework allows users to define high-level and context-aware inter-app policies which are then automatically refined to fine-grained enforceable policies. The results showed that the resulting instrumented applications have only 1.2% additional code compared to their original version, which is relatively small. The results also showed that the rewriting time takes no more than 51s and most importantly the success rate was 99.78%, which is significantly high compared to other existing approaches.

CHAPTER 3

CAPEF: CONTEXT-AWARE POLICY ENFORCEMENT FRAMEWORK FOR ANDROID APPLICATIONS

3.1 Introduction

Context awarenes service is a key driver for the modern mobile operating systems which are commonly prompting users by showing authorization dialog boxes asking for allowing or denying access to some functionalities. These services opened a big interest in defining, managing, and enforcing context-aware policies especially for those scenarios that put users under the risk of leaking or misusing their credential information. Yet, thousands of malicious applications are developed on the Android store and affecting millions of Android users worldwide. To safeguarded Android users, Google is frequently announcing the cracking down of such malicious applications. For instance, Google has removed over 700,000 malicious applications from the Play Store in 2017 only Maring (2018). Based on Goggle statistics, this is 70% more than what Google removed in 2016. Very recently in 2022, Google has removed 16 bad apps that missuses mobile data and draining batteries. Surprisingly, these apps have been downloaded by more than 20 million users around the world Rathore (2022).

Android system protects sensitive APIs by granting them permissions to amplify application privileges on the device, including access to stored data and services, such as network, memory, and so on. All permissions required to access the protected APIs in each application's manifest file (AndroidManifest.xml) man are necessarily set by the Android app developers. System permissions are divided into two categories, normal and dangerous. Normal permissions do not pose a direct threat to the privacy of the user, although dangerous permissions may allow the application to access the user's confidential data. Existing application authorization system in Android allows you to control only the permissions that are classified as dangerous, whereas our developed policy approach, offers the ability to control all monitored permissions as any application may cause a risk or conflict within a specific context without user awareness. Also, our model will mitigate malware collusion in which two or more malicious apps combine to accomplish their goals. For example, a user can choose to allow a camera app to access the camera, but not to the contact information without his consent or awareness. Another example, where normal applications can be granted permissions to collecting user's contacts, photos, videos, locations, or banking information then sending it over the internet to a remote server and taking into consideration pre-defined context aware access control policies. Therefore, a flexible policy-based access control system is needed to monitor APIs functions in Android applications, especially those requiring access to the user's sensitive and crucial information. The current permission system of Android still has some limitations, where users must grant most permissions requested by an application to install it, without being able to automatically manage most of these permissions based on the user's context afterwards.

This drawback has motivated the researchers to propose context-aware policies and/or define policy languages to enforce the current Android permission system either by modifying the Android platform such as in Arena *et al.* (2013); Nauman *et al.* (2010); Zhou *et al.* (2011); Hornyack *et al.* (2011); Feth & Pretschner (2012) or by instrumenting the Android Applications Xu *et al.* (2012); Jeon *et al.* (2012); Davis *et al.* (2012); Davis & Chen (2013); von Styp-Rekowsky *et al.* (2013); Zhang *et al.* (2013); Pearce *et al.* (2012); Shekhar *et al.* (2012); Zhang & Yin (2014); Falcone & Currea (2012) and more recently in Riganelli, Micucci & Mariani (2019); Alhanahnah *et al.* (2020); Grace & Sughasiny (2022) (more details and comparisons can be seen in the background and literature review section). While, existing works have demonstrated significant effectiveness in protecting users against threats, these approaches are still impeded by several drawbacks.

3.1.1 Challenges

Defining and monitoring context-aware inter-app policies of sensitive APIs on Android applications presents several challenges. Especially, when we are trying to defend applications collaborating to create malicious contexts:

i Context-aware inter-app policies are difficult to predict as they frequently get changed and need to be updated accordingly for accuracy and correctness.

ii Beside the difficulty of representing the security policies in a logical language which can contain user contexts and semantics, a key challenge is how to design and develop effective and efficient algorithms to monitor private information leakage on semantics levels.

iii There is a need for a policy language that can provide certain agreements that empower users with the ability to prioritize specific mobile resource and specify the amount and kind of information that can be shared within particular contexts. For instance, a user should be able to share a personal data with a specific service provider based on his location or at a specific time of the day to ensure his privacy. In this case, the user must agree on a trade-off between data privacy and the needed service. As a result, a policy should be defined to ensure privacy, while certain context-based information can be shared.

iv Android Sandboxing is introduced in the recent Android version 13.0. Sandboxing protects apps data and permission from getting access from other apps. This new feature will have an impact on our inter-apps policy model, but our main goal still effective which is to allow the user to define his policies to work automatically depending on the context update, to the running apps etc. Therefore our, framework can fully protect the Android OS versions lower than 13.0 in which the installed apps can communicate between each others.

v Due to the resourced constrained mobile devices, we have to decide, in early stages the instrumentation and monitoring location, whether to be on device, external PC or App market.

3.1.2 Contributions

This article is contributing solutions for the above-mentioned challenges and limitations by introducing:

i A formal context-aware policy specification framework for Android applications that effectively describe users defined consents.

ii A design and implementation of an instrumentation framework to mitigate privacy leakage across different Android applications.

iii Providing a centralized applications controller. This will allow users to manage all API calls performed by the applications installed on the device and to mitigate malware activities.

iv Effectively enforce different behaviors based on automated context-aware policies for each Android application individually without any modification to be entailed in the underlying platform.

v Experiments conducted on our CAPEF in terms of performance by analyzing the size of the enforced application after the instrumentation, also, the execution time of the policy decision, and the policy size which affects the complexity of the applied rules and conditions.

3.2 System Overview

This section gives an overview of our approach that automatically enforces user specific context-aware policies for android application and monitors all API calls that occur due to the interaction between enforced applications. Our developed system works in the application level of the Android framework, and its main components are illustrated in Fig. 3.1.

i From left, the first components represented the instrumentation of the targeted Android application (byte code or source code) by injecting monitoring code before each selected API method's call to intercept it at run time.

ii After the instrumentation, the applications will be forced to communicate with our controller that monitors the targeted context-aware inter-app calls.

iii Then, users will use CAPEF interface to create context-based rules and conditions in the form of security and privacy policies. More precisely, the policy represents a rule or set of rules based on a set of conditions and save it in the policies Database.

To motivate and illustrate our approach, we present the following scenario for vulnerability pattern that consider context-awareness policies. In this scenario, a user is using a public WiFi network in a coffee shop and he is trying to consult his credential banking information through his banking application. As the public network is not secure and there are other people who use the same network, so there is a risk of sending requests to attackers and thefts of private data. Android system checks only if the user has been previously granted the permission of accessing WiFi network and doesn't provide any context-awareness policies to mitigate such dangers scenarios. CAPEF can provide more effective access control not only on the permissions declared in advance by the user but also at run time based on the context of the user, device, and resources. Thus, as a solution to the above-mentioned scenario, a user can use CAPEF to define a policy that prevent the use of banking applications while connecting to a public WiFi network. When an enforced bank application attempt to get public WiFi access, our application controller will first check if the permission is declared in the application manifest file. Then, will check if the user has defined any policies related to this permission in the policies database. Subsequently, the controller access decision will be based on the predefined policies for that specific API. As a result, the controller will notify the user for not being allowed to connect to public WiFi for banking activities.

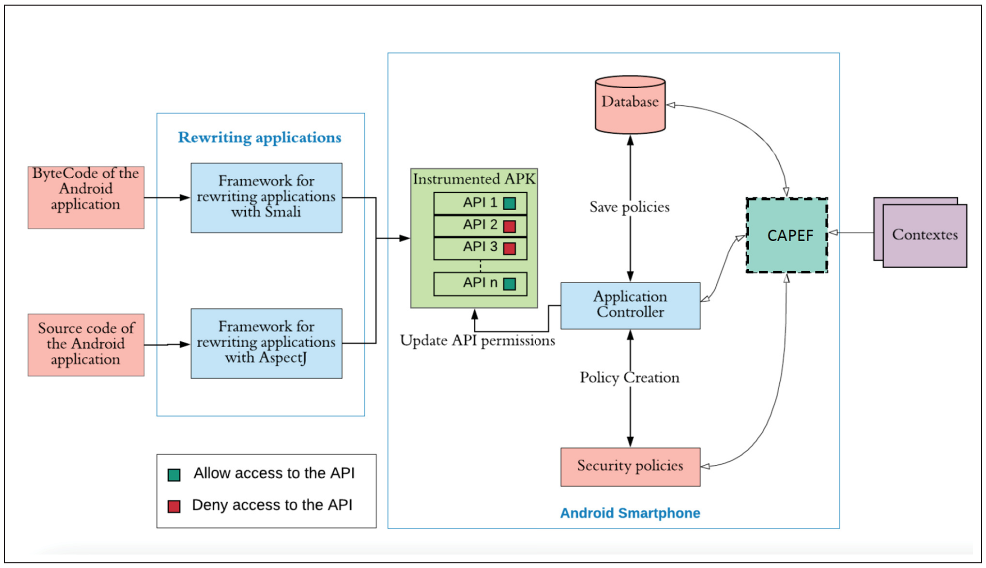


Figure 3.1 System Overview

3.3 CAPEF

Context-Aware policies are not static and might be changing over time to fulfill users' needs. Therefore, these policies could be used to control the behavior of the applications during run-time, which in our case, means monitoring and controlling all sensitive activities across different applications according to user's context. To achieve this goal, we provided a native Java-based CAPEF that allow regular users and enterprises to interpret and enforce their complex context-Aware policies. Contexts will represent various parameters including time, location, identity, activity, application, device status, resources etc. Moreover, these policies can be exported in multiple formats such as XML and JSON as they are widespread use today for data interchange and structured stores.

To develop the CAPEF language that allows the user to define contextual policies and transform them into security controls, we must rely on a flexible design that varies with the complexity of the policies, rational, able to execute all the conditions and easy to add new contexts.

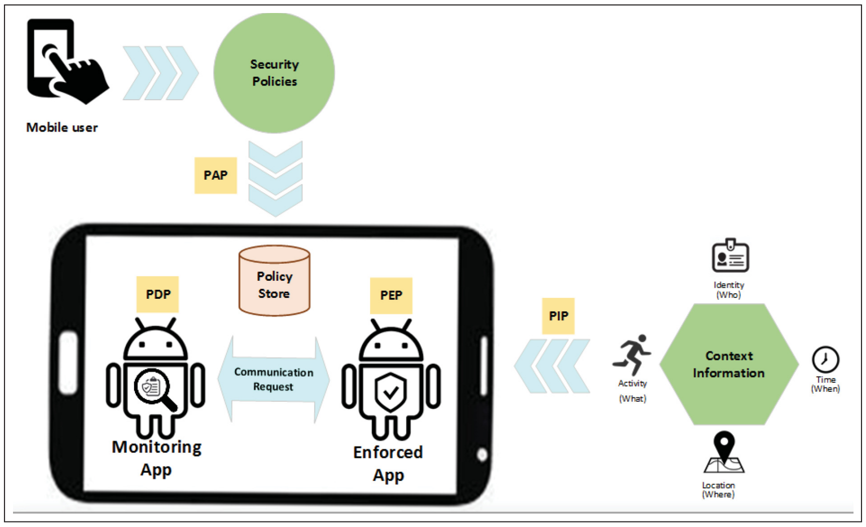


Figure 3.2   CAPEF Architecture

3.3.1 CAPEF Architecture

The main components of CAPEF are shown in Fig.3.2. which represents the following:

i **Security policies**: which is an interface for the user to define context aware policies and save them at the policy store. This component will act as a policy administration point (PAP) which is the source of the policies.

ii **Enforced Application**: Which plays the role of policy enforcement point (PEP) that receives the access request and move it to the Monitoring application for making access decision based on the predefined context aware policies.

iii **Context Information**: Provides context information in a form of attribute values about the targeted applications, resources, activates, actions and so on. This component will play the role of policy information point (PIP) in our system.

iv **Monitoring Application**: This component plays the role of policy decision point (PDP). It takes the access request from the PEP then interacts with PAP and PIP that capture the required context information to identify the appropriate policy. Then evaluates the request according to the applicable policy and returns the decision to the PEP.

3.3.2 Formal Definition

Hereafter, we formally define our ABAC policy model, which is composed of three main entities:

1. A, P, and C : sets of application, permissions (resources) and contexts, respectively;
2. AA , PA ,and CA are the pre-defined sets of attributes for applications, permissions, and contexts, respectively.

   I An application *app* is a represented by a tuple as follows:
   app = <name, visibility, class, APIs>, where:
      i visibility ∈ {background, foreground}
      ii class ∈ { banking, communication, recording, games, media, location}
      iii APIs: a set of APIs that can be invoked during execution

   II A permissions *per* embeds the access to the resource, it is a represented by a tuple as follows:
   per = < name, resource, securityLevel >,
      i resource ∈ { personal data, calendar, camera, wifi, account, calls, sms,Audio, GPS},
      ii securityLevel ∈ { Normal, Dangerous },

   III A context *c* is a represented by a tuple as follows:

c =< time, location, fgApp, BgApps availableCPU, availableMEM, availableNRG >,

i fgApp indicates which application is running in foreground;

ii BgApps is the set of the applications running in background.

3. Attr(app),Attr(per), and Attr(c) are attribute assignment relations for application app, permission per and context c, we have respectively

   I Attr(app) $\subseteq$ name $\times$ visiblity $\times$ class $\times$ APIs;

   II Attr(per) $\subseteq$ name $\times$ resource $\times$ securityLevel;

   III Attr(c) $\subseteq$ time $\times$ location $\times$ fgApp $\times$ BGaps $\times$ availableCPU $\times$ availableMEM $\times$ availableNRG.

For the value assignment of each attribute, we use the following notation: **entity.attribute= value** ,

For example, for an application **app**, a permission **per** and a context **c**, we have the following assignments:

app.visiblity = 'background', per.securityLevel = 'Dangerous', c.location = 'Montreal'.

4. The ABAC policy rule that decides whether or not an application **app** is allowed to get the permission **per** under a particular context **c**, is denoted as a predicate **PR** over the attributes of **app**, **per** and **c** as follows:
   Rule : Allow(app, per, c) $\longleftarrow$ PR(Attr(app), Attr(per), Attr(c))

Given all the attribute assignments of *app*, *per*, and *c*, if the predicate's evaluation is true, then the application **app** is allowed to get the permission **per** under the context **c**; otherwise, the permission is denied. Using the formal definition, we can have different types of policies for the *app*, for example:

1. A rule that dictates that " When a banking applications is being used, so the TakeScreenshot actions should be prevented from running" can be written as:
   Allow$_{screenShot}$(app, per, c) $\longleftarrow$ (TakeScreenshot $\in$ app.APIs) $\wedge$ (per.resource==screen) $\wedge$ (c.fgApp.class != banking)

2. A rule that dictates: "RecordVoice and RecordCall applications should be prevented from running when the user is dialing Skype from 9:00 to 10:00" can be written as:

   i Allow$_{recordVoice}$(app, per, c) $\leftarrow$ (RecordVoice $\in$ app.APIs) $\wedge$ (per.resource==voice) $\wedge$ ((c.fgApp != 'skype') $\vee$ (c.time <9:00 am $\vee$ c.time > 10:00am))

   ii Allow$_{recordCall}$(app, per, c) $\leftarrow$(RecordCall $\in$ app.APIs) $\wedge$ (per.resource==call) $\wedge$ ((c.fgApp != 'skype') $\vee$ (c.time < 9:00am $\vee$ c.time > 10:00am))

   Or simply by combining the two rules as following:

   Allow$_{record}$(app, per, c) $\leftarrow$(RecordVoice , RecordCall $\cap$ app.APIs!= ϕ) $\wedge$ (per==voice $\vee$ per==call) $\wedge$ ((c.fgApp != 'skype') $\vee$ (c.time < 9:00am $\vee$ c.time > 10:00am))

3.3.3 CAPEF Policy Specification

CAPEF language is based on the definition of a policy that consists of a user ID, a policy name, a policy execution state, a control rule, and a list of applications to control. Each of these applications is characterized by a name, package name, execution status and a list of permissions. The permissions consist of a name and a execution state. The security rule consists of a list of objects that can be Parentheses, Conditions, Logical Operators, Conditional Operators, CPU, Time, Resource Used, Location and Battery.

As shown in Fig.3.3, user-defined security policies and its rules can contain multiple conditions, different contexts, multiple logical operators, and parentheses to specify priority between conditions. Also, these policies can be executed simultaneously in different inter-app activities across different applications. In this case, policy decision becomes more complex. Therefore, to facilitate and accelerate the execution of any compound policy, our algorithm will receive the current contexts and the control rule as parameters then returns the policy decision of the controller.

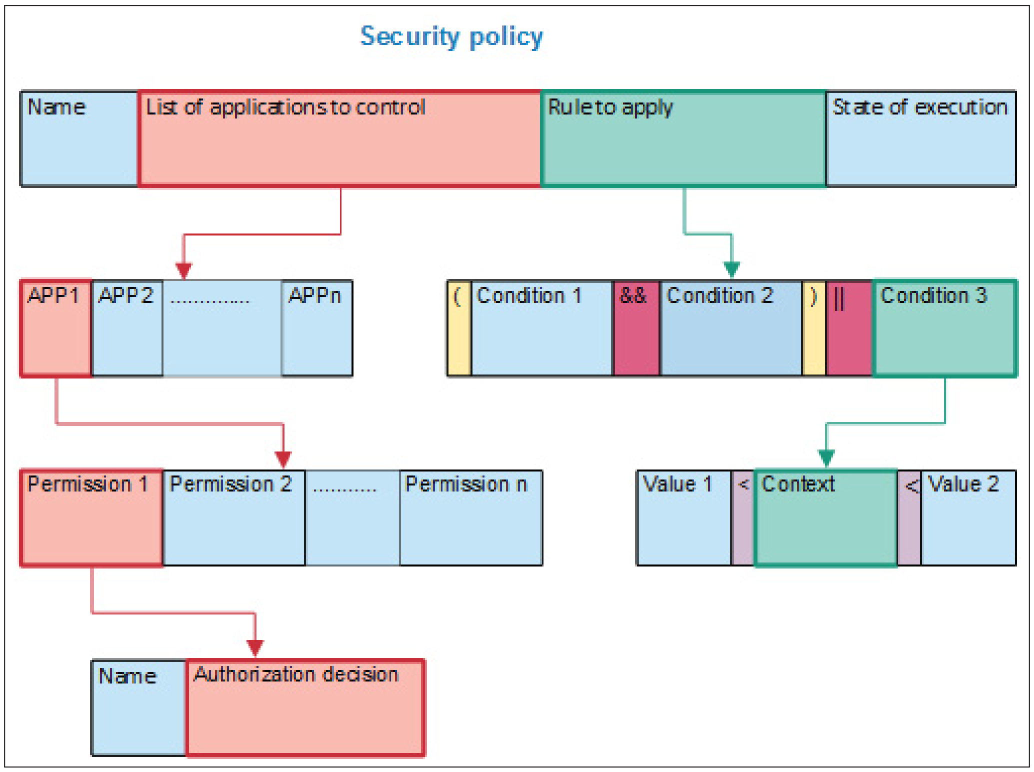


Figure 3.3   CAPEF Policy Execution Structure

In addition, the security rule might have sub-nodes of other rules, and themselves are sub-rules of the main rule. Therefore, we have adopted a recursive technique to reduce the complexity of the composed security rules. In this case, the algorithm will call itself, and recursion stops condition must be checked, otherwise the program will be stuck in an infinite loop.

To show the usefulness of our solution, we have chosen two examples of dangerous scenarios and we will translate them into security policies.

I Critical scenario 1: If the user uses his banking application to check his data and deposit a check in his account, while a TakeScreenshot application is installed on his smartphone.This application that takes screenshots automatic present a risk on banking data that is personal life data.

   i Solution: You must prevent the TakeScreenshot application and any screenshot function from running when the user is using their banking application.

    - ii Security Policy: If [BankApp] is running, then stop the [TakeScreenShot] application.

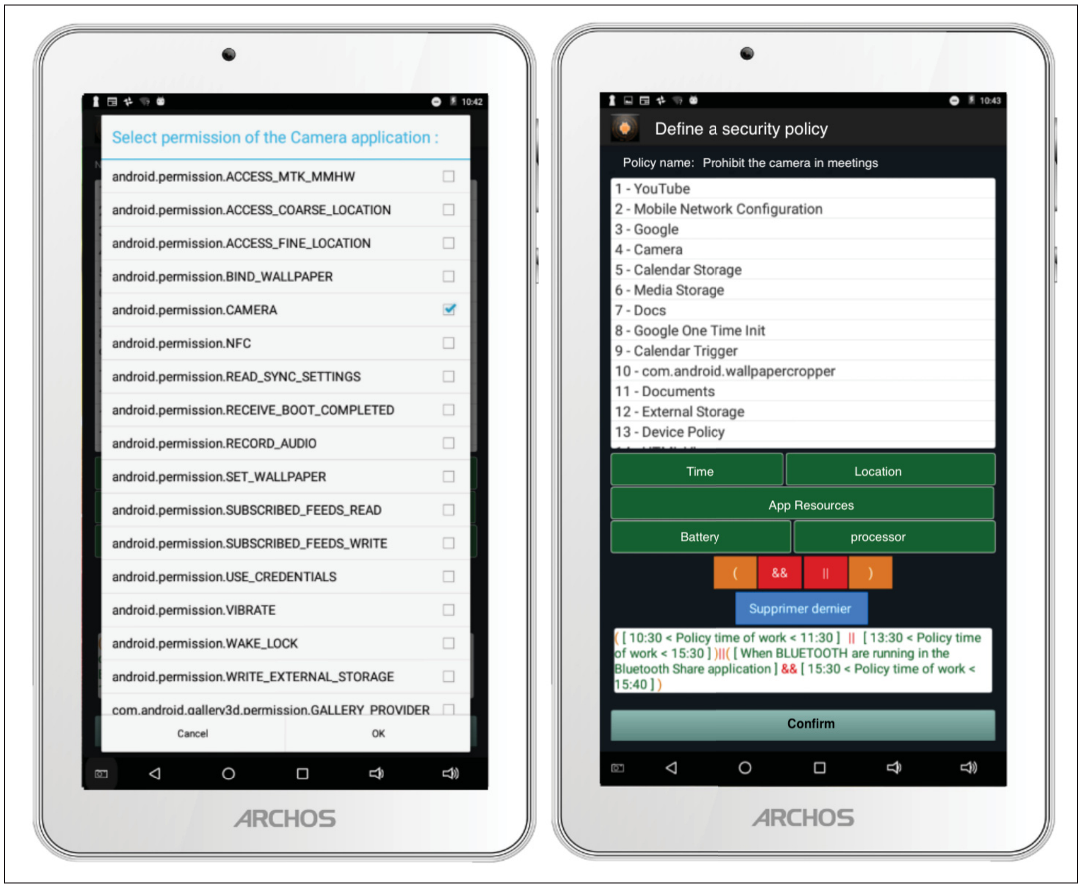
- II Critical Scenario 2: If the user is in a private work meeting every Monday from 9:00 am to 10:00 am by Skype while many other apps are able to record his speech and share it in public as RecordVoice and RecordCall applications.
    - i Solution: RecordVoice and RecordCall applications should be prevented from running when the user is dialing Skype from 9:00 to 10:00.
    - ii Security Policy: If ((CALL_PHONE in [Skype]) && (9: 00 <= Current_Time <= 10: 00)), then stop or prevent (if not yet executing) the application [RecordVoice && RecordCall].

To apply and evaluate the defined context aware policies, an application controller has been developed to allow users to define policies depending on different types of contexts and conditions.

3.4 Centralized Application Controller

The developed controller provides a user interface to translate the dangerous scenarios into security policies using the CAPEF. Based on the defined policies, the controller will make the adequate access control decision to allow or block applications from using certain permissions. Moreover, provide centralized control of installed applications which capture mandatory decisions that are automatically dependent on the current context.

Fig. 3.4, shows an example of how to define a security policy with our controller. The control scenario is to block the execution of the Camera resource in the Camera application if the user is in a meeting from 10:30 to 11:30 or from 13:30 to 15:30 otherwise it is in a meeting from 15:30 to 15:40 and that Bluetooth is enabled in the BluetoothShare Application.



Select permission of the Camera application :

- android.permission.ACCESS_MTK_MMHW ☐
- android.permission.ACCESS_COARSE_LOCATION ☐
- android.permission.ACCESS_FINE_LOCATION ☐
- android.permission.BIND_WALLPAPER ☐
- android.permission.CAMERA ✓
- android.permission.NFC ☐
- android.permission.READ_SYNC_SETTINGS ☐
- android.permission.RECEIVE_BOOT_COMPLETED ☐
- android.permission.RECORD_AUDIO ☐
- android.permission.SET_WALLPAPER ☐
- android.permission.SUBSCRIBED_FEEDS_READ ☐
- android.permission.SUBSCRIBED_FEEDS_WRITE ☐
- android.permission.USE_CREDENTIALS ☐
- android.permission.VIBRATE ☐
- android.permission.WAKE_LOCK ☐
- android.permission.WRITE_EXTERNAL_STORAGE ☐
- com.android.gallery3d.permission.GALLERY_PROVIDER ☐

Define a security policy

Policy name: Prohibit the camera in meetings

1 - YouTube
2 - Mobile Network Configuration
3 - Google
4 - Camera
5 - Calendar Storage
6 - Media Storage
7 - Docs
8 - Google One Time Init
9 - Calendar Trigger
10 - com.android.wallpapercropper
11 - Documents
12 - External Storage
13 - Device Policy

([10:30 < Policy time of work < 11:30] || [13:30 < Policy time of work < 15:30])||([When BLUETOOTH are running in the Bluetooth Share application] && [15:30 < Policy time of work < 15:40])

Figure 3.4 Screenshots of a policy definition within our controller

3.4.1 Managing Security Policies

The Application controller interface has been developed in a way that the user will be able to define any security policies in a few simple clicks. We chose our solution to be ergonomic, personalized, and user-centric design to have a convenient and easy-to-use service. It has also been taken into consideration that our user interface must reduce the search effort and limit data entry. In addition, all policies created by the user have been saved in a database. With the database, the user will be able to import, create, view, and modify security policies.

3.4.2 Export/Import Security Policies

Our developed solution allows the user to extract his defined policies and share them with other users of the controller or send them to nay server or cloud database. Therefore, our CAPEF flows same related policy language's architecture and structure. As discussed in the literature XACML is one of the good examples to extract our policies to its format. While Android system does not compile XACML language and all reviewed languages, our policies will be translated into java language to execute them, then will be extracted on different languages such as XML, JSON etc. Therefore, in order for our language to be compatible with other languages, we kept the same generic policy structure, objects and attributes applied by other languages as as shown in Table 3.1. Similar translation procedure will be applied when importing policies from other languages.

Table 3.1 Generic policy description

Element	**Description**
Policy-set	Presents a table that groups the list of policies.
Policy	Presents the policy object that contains the" Target and" Rule Sub-objects, as well as the attributes:" Combine "which presents the role of the policy, the" AUTHOR-KEY-CN attribute the author identifier of the policy and the attribute AUTHOR-KEY-FINGERPRINT" presents the fingerprint key of the author of the policy.
Target	This is the object that contains the definition of the target applications to control.
Rule	It is the object that defines the security rule, the attribute" effect "presents the decision of the control to give or withdraw the authorization.
Condition	Contains the permissions to remove and the contexts.
Resource-match	Contains different attributes:" attr "which can be an application to block or an API permission," subject-match "contains the application to control and" match "contains the permission to remove.

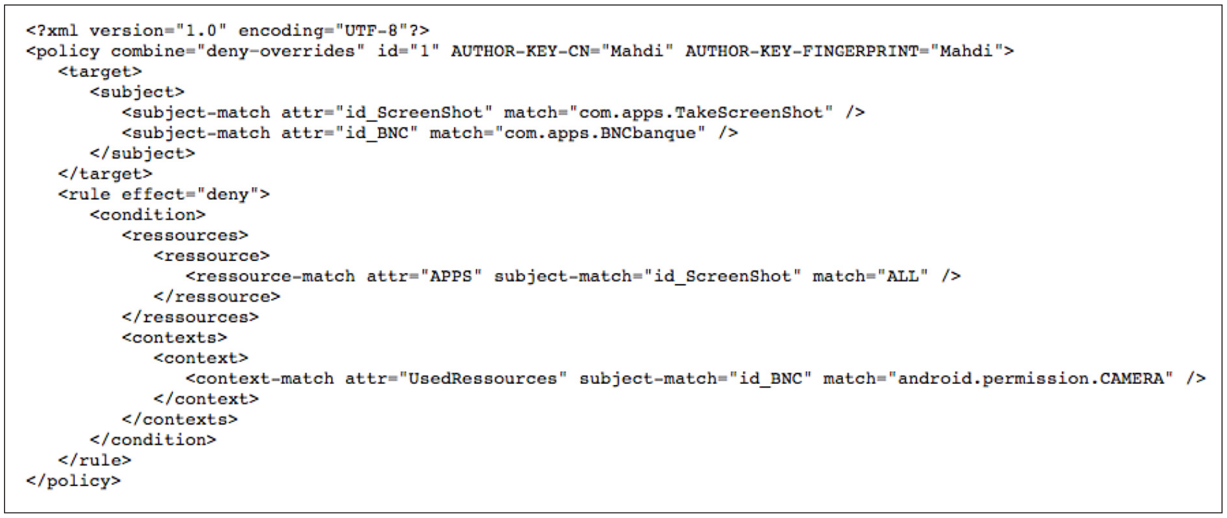
Among the values that can be assigned to attributes such as the names of applications, permissions, etc., we have defined symbols that allow us to simplify the rules, for example:

i ANY: it means no, for example a rental context, we do not need permission in this condition.

ii ALL: it means that we want to control all the permissions or all the applications it depends on the attributes used.

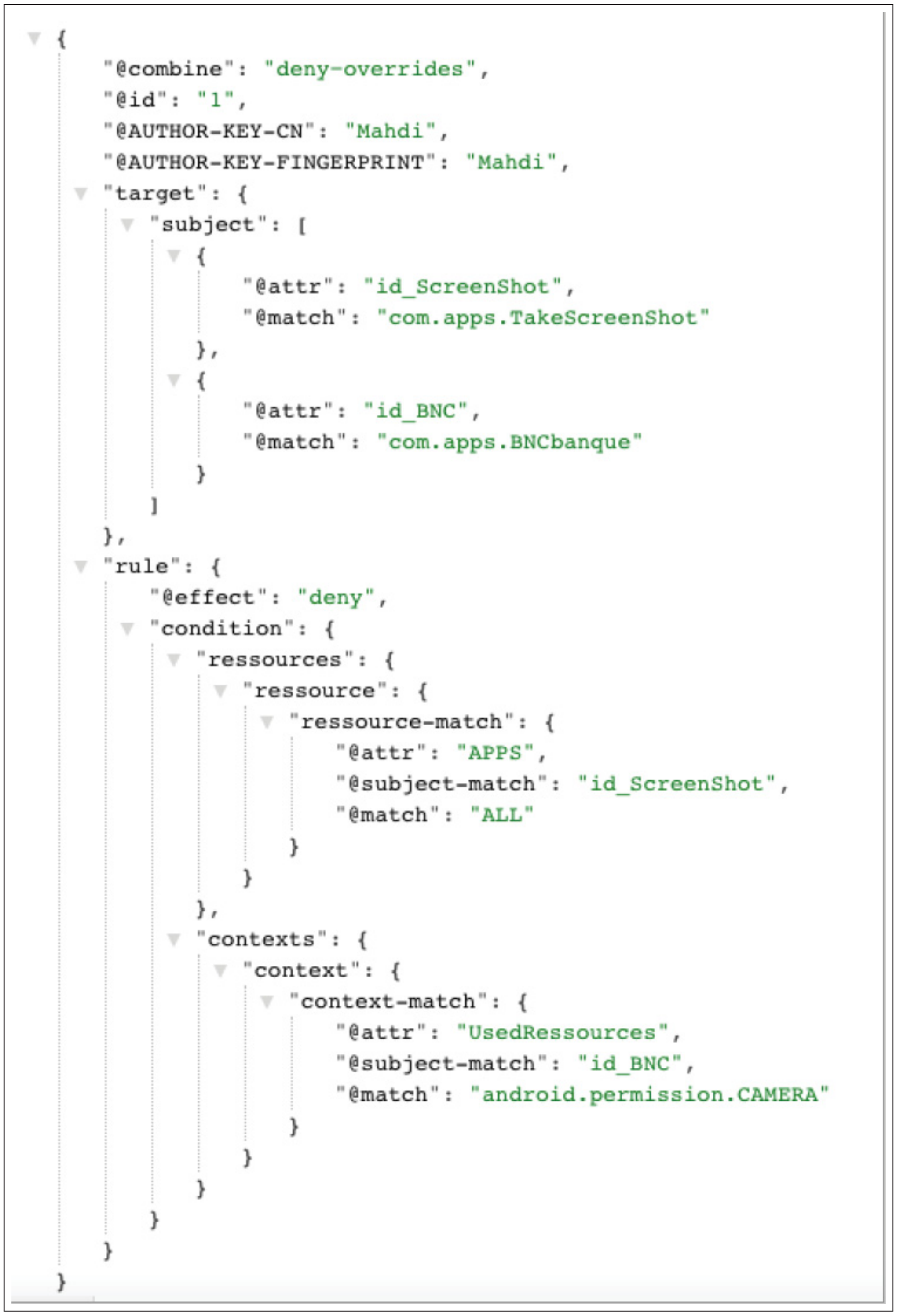
iii APPS: it means that we want to force the shutdown of an application.

iv API: that means we're going to apply the control on an API permission.



```
<?xml version="1.0" encoding="UTF-8"?>
<policy combine="deny-overrides" id="1" AUTHOR-KEY-CN="Mahdi" AUTHOR-KEY-FINGERPRINT="Mahdi">
   <target>
      <subject>
         <subject-match attr="id_ScreenShot" match="com.apps.TakeScreenShot" />
         <subject-match attr="id_BNC" match="com.apps.BNCbanque" />
      </subject>
   </target>
   <rule effect="deny">
      <condition>
         <ressources>
            <ressource>
               <ressource-match attr="APPS" subject-match="id_ScreenShot" match="ALL" />
            </ressource>
         </ressources>
         <contexts>
            <context>
               <context-match attr="UsedRessources" subject-match="id_BNC" match="android.permission.CAMERA" />
            </context>
         </contexts>
      </condition>
   </rule>
</policy>
```

Figure 3.5 An example of a security policy presented in the form of XML



```
{
    "@combine": "deny-overrides",
    "@id": "1",
    "@AUTHOR-KEY-CN": "Mahdi",
    "@AUTHOR-KEY-FINGERPRINT": "Mahdi",
    "target": {
        "subject": [
            {
                "@attr": "id_ScreenShot",
                "@match": "com.apps.TakeScreenShot"
            },
            {
                "@attr": "id_BNC",
                "@match": "com.apps.BNCbanque"
            }
        ]
    },
    "rule": {
        "@effect": "deny",
        "condition": {
            "ressources": {
                "ressource": {
                    "ressource-match": {
                        "@attr": "APPS",
                        "@subject-match": "id_ScreenShot",
                        "@match": "ALL"
                    }
                }
            },
            "contexts": {
                "context": {
                    "context-match": {
                        "@attr": "UsedRessources",
                        "@subject-match": "id_BNC",
                        "@match": "android.permission.CAMERA"
                    }
                }
            }
        }
    }
}
```

Figure 3.6   An example of a security policy presented in the form of JSON

The following scenario is established to extract CAPEF policies to communicate with other policy languages such as XML and JSON:

i Scenario: Prohibit launching the TakeScreenShot application that allows you to take automatic screenshots when the user opens the camera in his BankApp application to send a check.

ii Policy: if (CAMERA in [BNCApp]), then stop the application [TakeScreenShot].

Fig.3.5. shows the extraction of the above security policy scenario to XML and Fig.3.6. shows the extraction of the same scenario to JSON security policy.

3.5 Experimental results

This section presented the evaluation of our CAPEF and the application controller in terms of performance by analyzing: (1) the size of the enforced application after the instrumentation, 2) the execution time of the policy decision (3) The policy size due to the complexity of the applied rules and conditions.

3.5.1 Enforced Application Size

To measure the effect of the instrumentation method on the original size of the applications, we have instrumented a set of 109 applications using our rewriting framework. Table 3.2 shows a sample of eighteen applications, the original size and the new size after the instrumented. Indeed, this percentage represents the size of the code added during the control of APIs calls for each application individually

For all instrumented 109 applications, the average size added was 705 bytes, which is about 0.063% of the size of the original applications. Also, as shown in Fig.3.7, it is very clear that the size added is very small and will have a very small impact on the size of the original applications.

Table 3.2 The size of applications before and after instrumentation

Application	Original (Bytes)	Instrumented (Bytes)	Size added (Bytes)	Percentage
Contact Identicons	246904	247965	603	0.24%
GPS tracker	22420823	22421668	845	0.0037%
Show web view	483839	484460	621	0.12%
Contact Search	368610	369278	668	0.18%
Contacts Widget	227386	228034	648	0.28%
Beta Updater for WhatsApp	321673	322448	775	0.24%
Contact loader	2701541	2702019	478	0.01%
Photo Manager	366100	366668	568	0.15%
Wi-Fi setup	2177239	2177747	508	0.02%
Time tracker	1477841	1478711	870	0.06%
Calender Trigger	119863	120732	869	0.72%
Calender Color	629361	630232	871	0.13%
Calender Import Export	45823	46772	949	2.07%
CamTimer	1580753	1581393	640	0.04%
OpenCamera	226585	227420	835	0.36%
Microphone	1905254	1905910	656	0.03%
SMS backup	26071	26984	913	3.50%

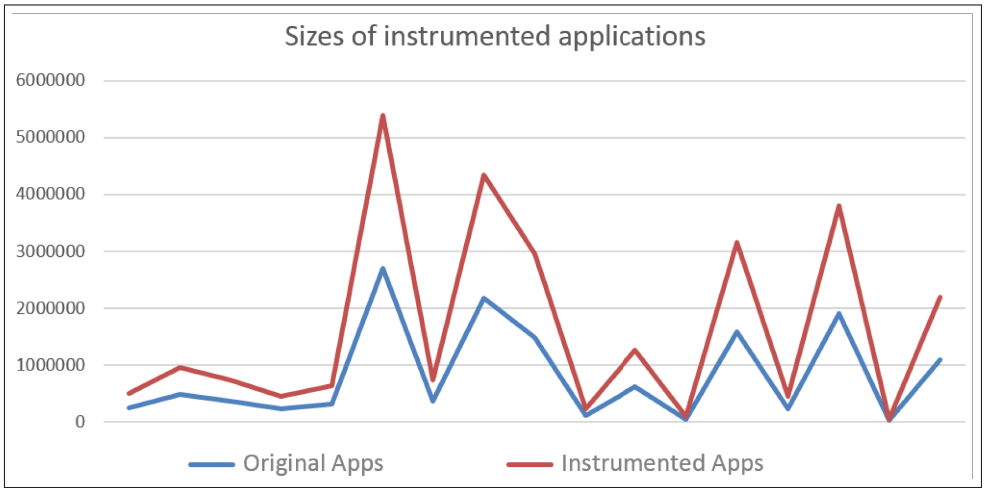


Figure 3.7 Average added size for 109 instrumented applications

3.5.2 Execution time of the policy decision

To calculate and evaluate the execution time of our defined policies, we have calculated the decision execution time for several context aware policies with different rules and conditions based on some selected scenarios. The following is a set of scenarios that has been chosen among many others used during our tests. These scenarios will be ranked in ascending order according to their level of complexity.

- **Scenario 1**: Prohibit launching the TakeScreenShot application that allows you to take automatic screenshots when the user opens the camera in his BankApp application to send a Check.

  Policy: Deny$_{TakeSceenShot}$(app, per, c) $\longleftarrow$ (TakeScreenshot $\in$ app.APIs) $\wedge$ (per.resource==screen) $\wedge$ (c.fgApp.class = banking)
- **Scenario 2**: Prohibits the RecordAudioMedia application from recording when the user is making a phone call through the PhoneCall application.

  Policy: Deny$_{RecordAudioMedia}$ (app, per, c) $\longleftarrow$ (RecordAudioMedia $\in$ app.APIs) $\wedge$ (per.resource == microphone) $\wedge$ (c.fgApp.class = (callPhone $\vee$ receivePhoneCall))
- **Scenario 3**: Prohibit the FakeGPS application from changing the user's location when using one or more of these BankApp, Uber, and Google-Map applications.

  Policy: Deny$_{FakeGPS}$ (app, per, c) $\longleftarrow$ (FakeGPS $\in$ apps.APIs) $\wedge$ (per.resouce == (accessCoarseLocation $\wedge$ accessFineLocation) $\wedge$ (c.fgApp.Class = (BankApp $\wedge$ UBER $\wedge$ GoogleMap))
- **Scenario 4**: Prohibition of the BankApp application to access the Internet or use the camera when the user is at TimHortons knowing that its longitude = 45.491318 and its latitude = -73.727987.

  Policy: Deny$_{internet \wedge camera}$ (app, per, c) $\longleftarrow$ ((internet $\wedge$ camera) $\in$ apps.APIs) $\wedge$ (per.resource == (GPS = [45.491318, -73.727987]) $\wedge$ (c.fgApp.Class = BankApp)
- **Scenario 5**: When the user is at the meeting at ETS from 8am to 9am. Prohibit Facebook, Instagram and Gmail applications from accessing the Internet, the camera and the location.

Prohibit Message application from receiving SMS and MMS. Also, Prohibit the recording feature of RecordAudio application.

Policy: Deny$_{all}$ (app, per, c) ← ((Facebook ∧ Instagram ∧ Gmail∧ RecordAudioMedia ∧ SMS ∧ MMS) ∈ apps.APIs) ∧ (per.resouce ==(GPS = [45.491318, -73.727987] ∧ internet ∧ microphone ∧ phoneCall ∧ receiveCall)) ∧ (c.fgApp.Class = meeting) ∧ (c.time >= 8am ∧ c.time <=9am)

The decision execution time has been calculated for each policy individually as following:

i For the Policy 1 and Policy 2, the test results were fixed because the context does not vary when entering random test values. The execution time for the first policy is 116 ms and for the second policy is 234 ms.

ii For the policy 3, the context is related to three different running applications, but it remains fixed. The execution time for the whole policy is 307 ms.

iii For the policy 4, our context is the location, so the results were more or less close, but they vary according to the change in GPS values. In this case the execution time of the whole policy is 314 ms.

iv For the policy 5, two different contexts were used time and location. The average execution time for notifying each application also was calculated. For the Skype application the execution time is 549 ms, for the Messages application is 592 ms, for the Instagram application is 634 ms, for the Gmail application is 758 ms and for Facebook is 814. Also, the average execution time for the whole policy is 818 ms.

Fig.3.8, shows the different policies execution times according to the complexity for each policy. All calculations and testes where repeated several times to ensure accuracy. As a result, we have noticed that as more the policy becomes complex the execution time becomes bigger.

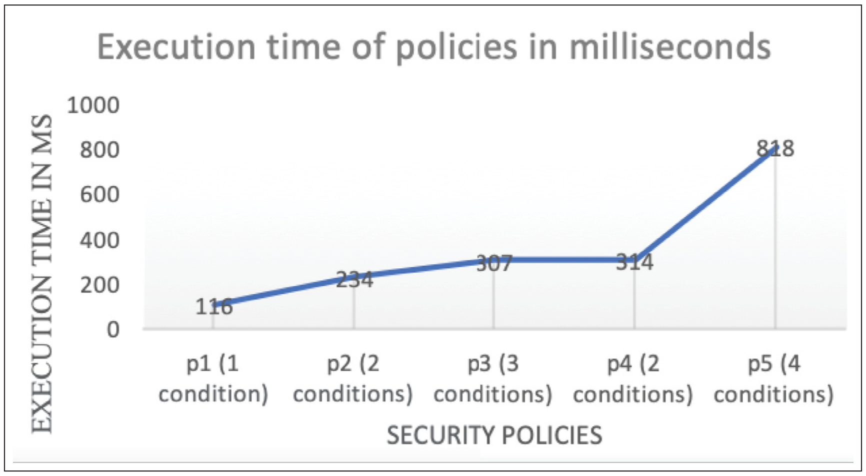


Figure 3.8 Policies execution times

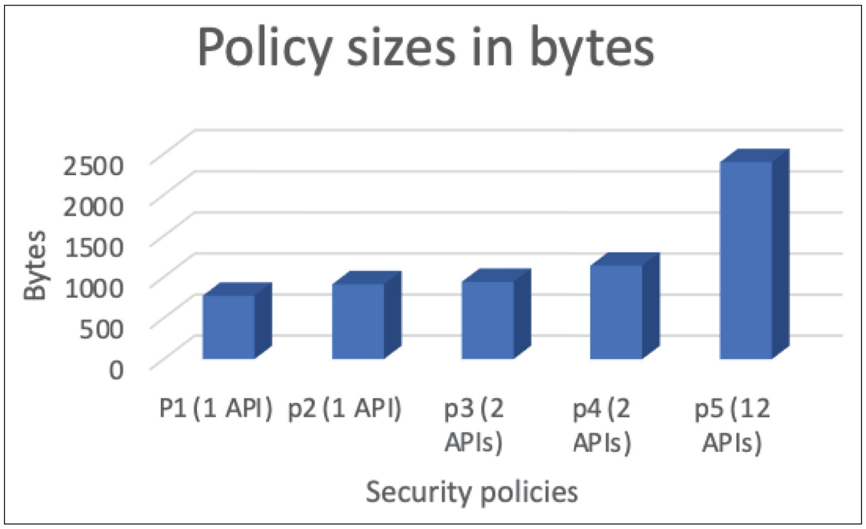


Figure 3.9 Policies sizes according to their complexities

3.5.3 Policy Size

The policy size is also changing due to the complexity of the applied rules and conditions. We have calculated the policy sizes on the list of 109 enforced applications. Also, we took the same fife scenarios and their security policies mentioned above in the previous section to make our simulation. Fig. 3.9 shoes the progression of policy sizes according to their complexities.

3.6 Conclusion

This work addressed problems related to context aware policies for Android applications as its one of the main targets of attackers. We, therefore developed CAPEF, which is a policy specification framework that enforces context-aware inter-app security policies to effectively describe users defined consents. Thorough experiments we have performed a study on the efficiency of CAPEF with respect to the size and execution time of the enforced applications. The evaluation results demonstrated the feasibility of our framework and the effectiveness of our policy specification language in enforcing complex context-aware policies on different Android applications.

CHAPTER 4

LCA-ABE: LIGHTWEIGHT CONTEXT-AWARE ENCRYPTION FOR ANDROID APPLICATIONS

4.1 Introduction

New wave of technological advancements in the era of Internet of Things (IoT) is yet to be unleashed along with new business models for future of smart and self-adaptive applications. According to Gartner, up to 8.4 billion IoT devices will be connected through machine-to-machine (M2M) communication and this number will significantly increase to reach 20.4 billion by 2020 Gar. These interrelated devices are provided with unique identifiers and the ability to transfer data over the network with improved efficiency, accuracy, and economic benefit of the IoT environments. While the success of all new self-adaptive technological advancements is evident, the complexity that they add to both identity and access management is crucial. Particularly, the massive volume of data generated by these devices raises the need for context-aware, adaptive access control solution to avoid any leak or exploitation of confidential information by unauthorized party. Further, securing end-to-end communication becomes crucial, which necessitates an encryption mechanism to ensure data security. In this scope, several encryption-based access control schemes have been proposed and adopted for privacy and security preserving in smart environments. Starting with symmetric key encryption and moving to the Public Key Infrastructure (PKI) which provides data signature to ensure integrity and session keys to ensure confidentiality. However, PKI still requires huge infrastructure (i.e., certifying authorities, registration authorities, repository, archives and end entity) to manage and maintain the certificates Ssh Jancic & Warren (2004). On the other hand, Goyal et al Goyal *et al.* (2006) introduced an asymmetric encryption technique called Attribute-Based Encryption (ABE) which is one of the most recent access control-based encryption scheme. Also, Yao et al Yao, Chen & Tian (2015) proposed a lightweight version ABE encryption and decryption scheme for smart environments. This technique allows the definition of fine-grained access control policies for data privacy and security. Yet, none of these existing works fulfils the needs of context-aware, smart, secure and adaptive solutions.

4.2 Architecture

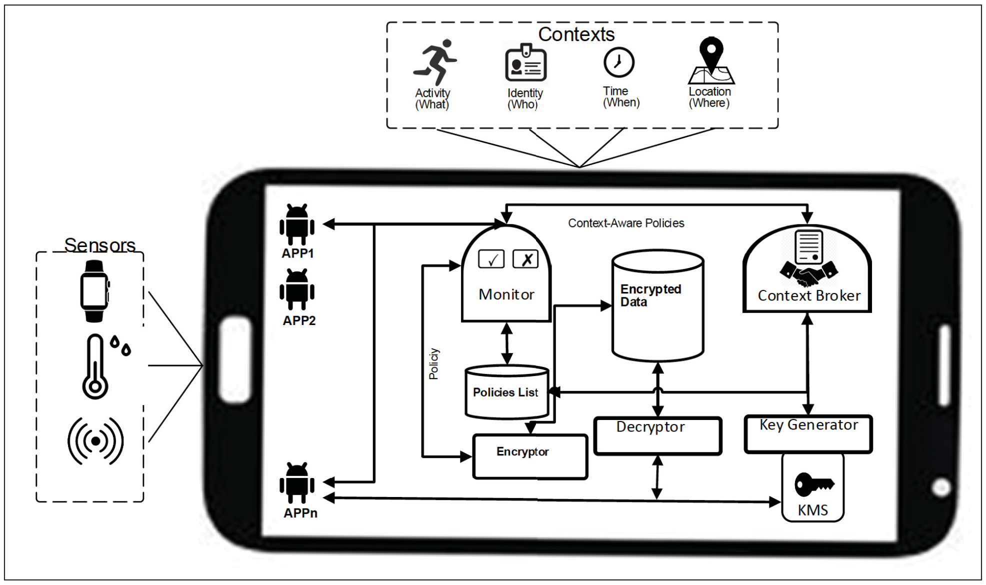


Figure 4.1 Architecture

Figure 4.1 shows the proposed overall architectural framework of the LCA-ABE and the interaction among its modules:

- **Context Broker** which assigns the user-defined policies in the system. This module utilizes the user-defined policies and transforms them into context-aware policies using the policy language proposed in Elarbi (2018). Next, these context-aware policies are stored in the *Policy storage* along with the parameters required by the *Key Generator*.
- **Key Generator** is responsible the generating the ABE Public Key, Master Secret Key, and the Secret Keys for the different applications based on the attributes provided by the *Context Broker*. The Public Key is transferred to the *Encryptor*. Whenever an Application Secret Key is generated, it is sent to the *Key Management Service*(KMS). The Key Generator generates the keys and stores them in the secured storage of android using the KMS.
- **Monitor** acts as an entry point for the sensors data as well as the applications accessing the the latter. It is also responsible of converting the context-aware policies into ABE policies for the *Encryptor*. After receiving sensors' data, the *Monitor* sends the data along with the ABE policy of the current context of the sensor from the *Policy storage* to the *Encryptor*.

Meanwhile, if an Application wants to access a sensor data, the *Application* queries the *Monitor* for the needed data. The *Monitor* verifies the *Application*, which then retrieves data from the storage and uses the *Decryptor* along with its Secret Key to decrypt it.

- **Encryptor** is a black-box which encrypts whatever data coming from the *Monitor* along with the access policy using CP-ABE and stores it in the secured storage.
- **Decryptor** decrypts the encrypted data being sent from the *Application* along with its Secret Key. If the decryption is successful the plain data is sent back to the *Application*.
- **Key Management Service** stores and manages all the cryptographic keys and also distributes the secret key to the designated Applications. It works in collaboration with the *Key Generator*.

In the sequel we emphasize on the three main operations in the proposed architecture:

- When users want to setup their policy for the applications and sensors, they first interact with the *Context Broker* to initiate the process of setting up the policies, context-aware policies, key generation, etc. The full sequence diagram of this process including the relevant interactions with other modules are shown in Figure 4.2.
- Figure 4.3 shows the sequence diagram for the interactions that take place when a sensor wants to send data to an application. The *Monitor* is responsible to request the data from the sensor and then passing it to the *Encryptor* for encryption and then storage in the database.
- The full sequence of operations for an Application to access sensor data is illustrated in Figure 4.4. In this framework, applications cannot access the sensor data directly, it has to go through the *Monitor*. Applications request the needed data from the *Monitor*, which verifies if that application is granted access before sharing the encrypted data. Afterwards the application transmit the encrypted data along with its key to the *Decryptor* for decryption. The *Decryptor* decrypts the data with the secret key provided and returns the data to the application.

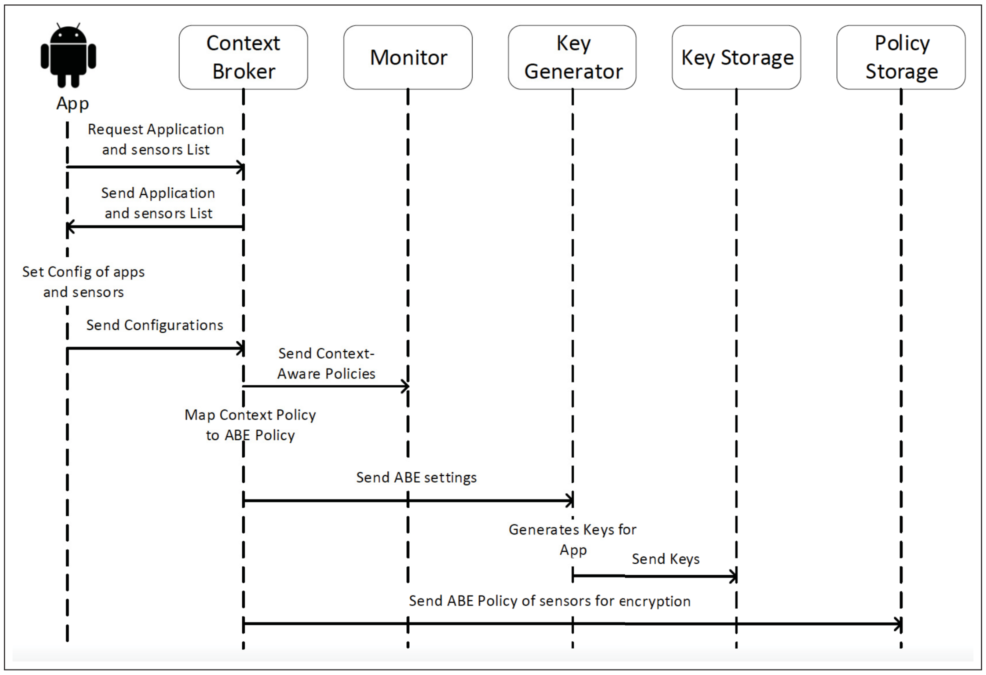

App
Context Broker
Monitor
Key Generator
Key Storage
Policy Storage
Request Application and sensors List
Send Application and sensors List
Set Config of apps and sensors
Send Configurations
Send Context-Aware Policies
Map Context Policy to ABE Policy
Send ABE settings
Generates Keys for App
Send Keys
Send ABE Policy of sensors for encryption

Figure 4.2 Broker Sequence diagram

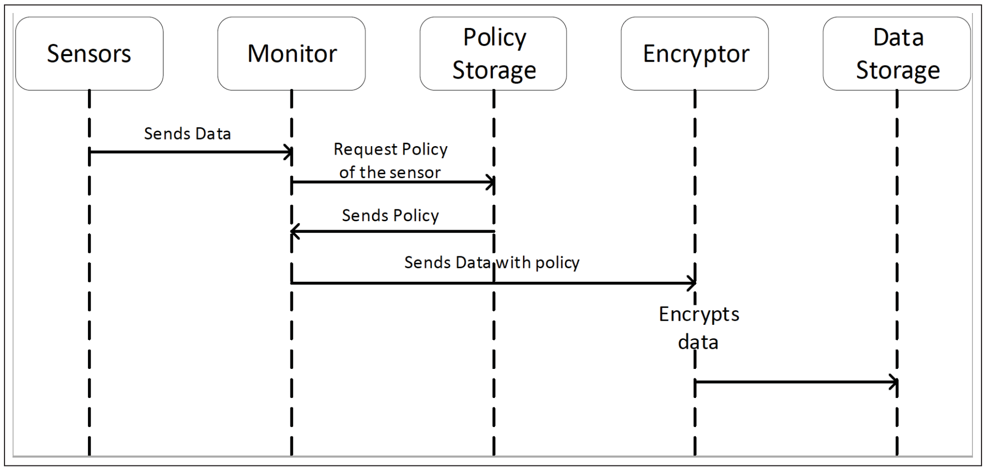


Figure 4.3 Sensor Sequence diagram

4.3 Implementation

For the implementation of the the proposed framework, we first modified the implementation of the CP-ABE by Clo to allow the encryption algorithm to run on Android and also to make it compatible with the policy language implementation. As for the Context-Aware Policy

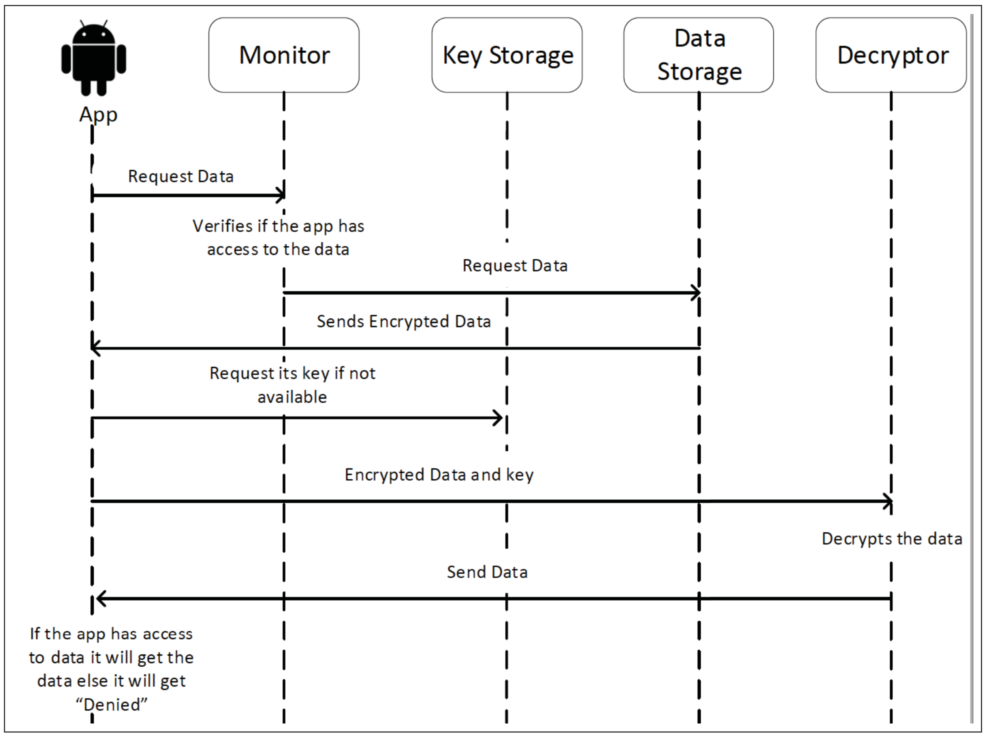

App
Monitor
Key Storage
Data Storage
Decryptor
Request Data
Verifies if the app has access to the data
Request Data
Sends Encrypted Data
Request its key if not available
Encrypted Data and key
Decrypts the data
Send Data
If the app has access to data it will get the data else it will get "Denied"

Figure 4.4 Application Sequence diagram

Language we have used Elarbi (2018) for the implementation. Regarding the conversion of Context-Aware Policy to Attribute Based Encryption policy is achieved by using mapping for this implementation. The device used is Acer Iconia One 7. Its hardware and software specification are shown in Table 4.1.

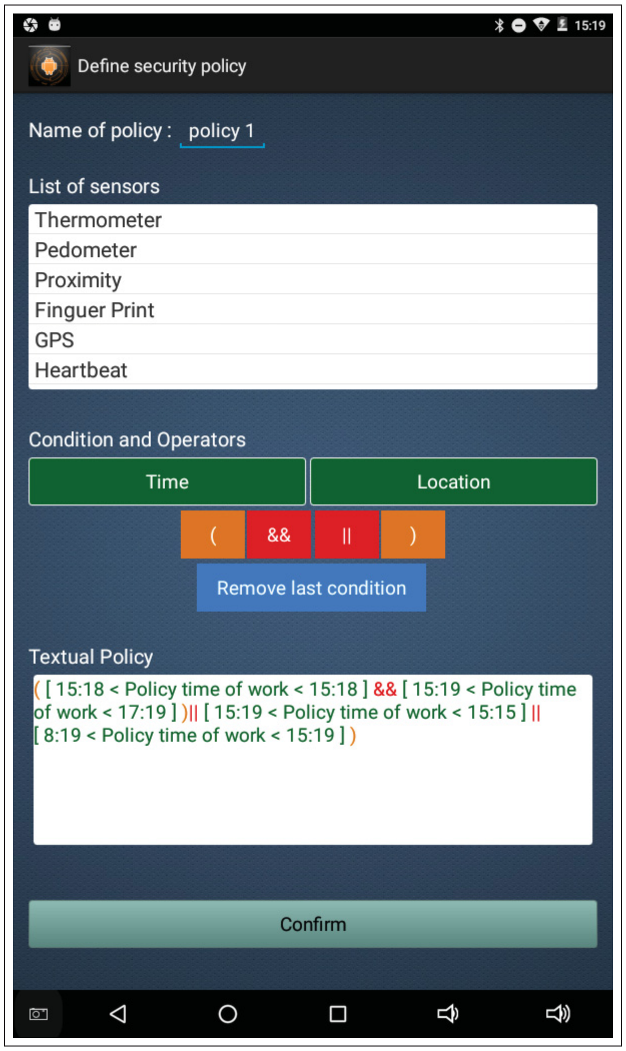
Table 4.1 Hardware and software Specification

Processor	1.6GHz dual core Intel Atom Z2560
RAM	1 GB
Storage	8 GB
Android Version	7.0

4.4 Case Study and Experimental Analysis

4.4.1 Case study

Patient context awareness is an important concept for application services in mobile health systems. The use of wearable monitoring devices allows continuous monitoring of patients according to their context information. In this paper we investigated a multi-sensors system that uses context-aware policies to enhance the accuracy of potentially dangerous Heart Rate variability by taking into account patients' context information. We have designed and implemented an automated, user-friendly, and context-aware access control model that allows receiving data from several sensors and provides context decision accordingly. Figure 4.5 shows a screen shot of an example of our implementation for defining and assigning context-aware policies for list of sensors. Based on context information such as time and location, access can be granted/revoked.


15:19

Define security policy

Name of policy : policy 1

List of sensors

Thermometer
Pedometer
Proximity
Finguer Print
GPS
Heartbeat

Condition and Operators

Textual Policy

([15:18 < Policy time of work < 15:18] && [15:19 < Policy time of work < 17:19])|| [15:19 < Policy time of work < 15:15] || [8:19 < Policy time of work < 15:19])

Figure 4.5 Screen shot of Policy assignment

4.4.2 Experimentation and Evaluation

We have performed thorough experiments to evaluate and measure the performance overhead of the implementation. The evaluation analyses the resources utilization, size of the ciphertext as well as the execution time of the encryption, decryption and key generation processes. The experiments were performed based on the number of ABE attributes varying from 1 to 50. For Secret Key Generation, the parameter is the number of attributes. As for Encryption, the main parameter is the length of access policy. Finally for Decryption, the secret key is the crucial parameter.

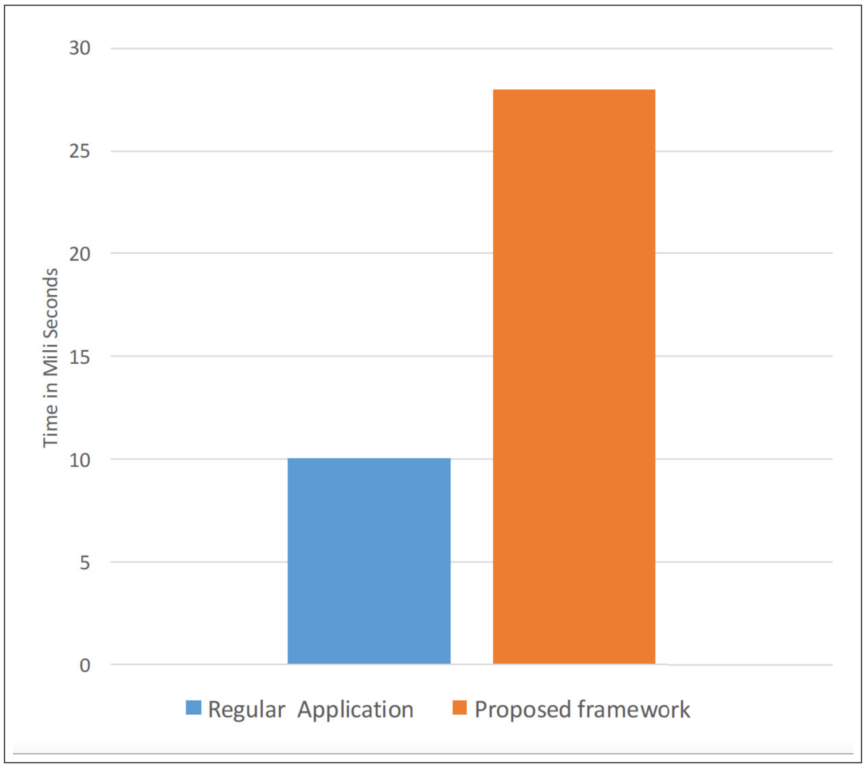


Figure 4.6 Comparison of Application with the Framework

Figure 4.6 compares the execution time of the original application with the time needed in the proposed framework. The results show slight increase of 18 ms in the execution time for the policy length of three.

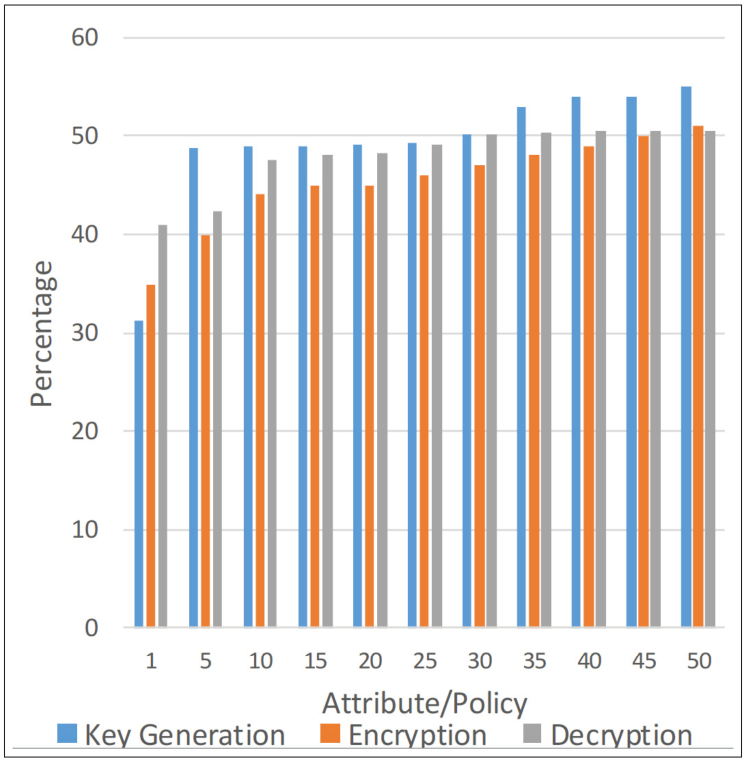


Figure 4.7 CPU utilization

Figure 4.7 displays the CPU utilization of the device for Key Generation, Encryption and Decryption. The CPU utilization for Key Generation is the highest, whereas the Encryption requires the least CPU. On the other hand Decryption has almost the same CPU utilization when the number of policy attributes varies between 30 and 50.

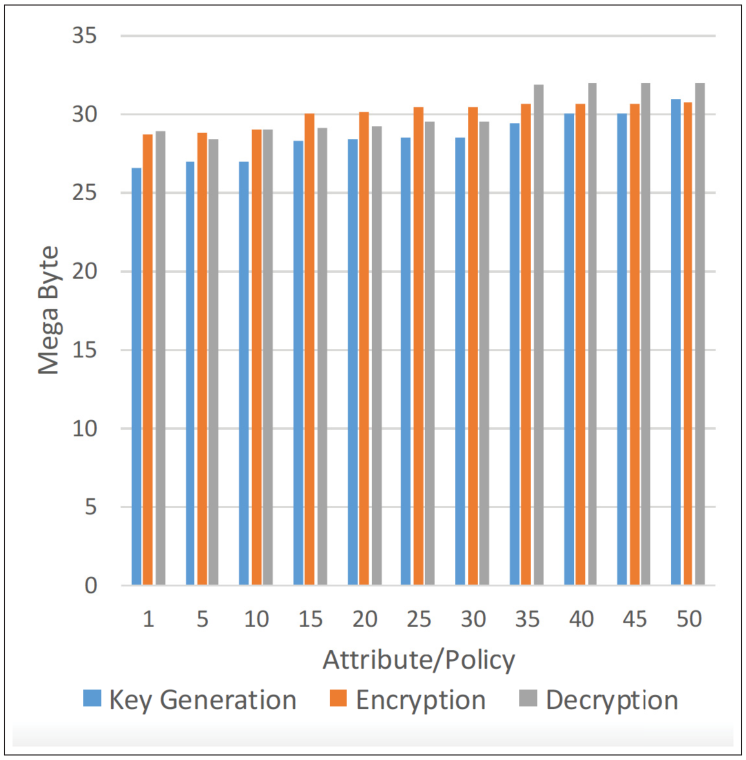


Figure 4.8 Memory utilization

Figure 4.8 shows the memory utilization during the Key Generation, Encryption and Decryption. This experiment shows that the Key Generation process requires less memory usage than the other operations and the Decryption has the highest memory usage values. Another interesting observation is that the memory utilization does not go above 32 MB.

Figure 4.9 illustrates the size of the data generated for the Secret Key and the ciphertext after encryption. In this experiment we kept the size of the plain text constant to 160 bytes. It is clear that the file size increases linearly for the Secret Key as well as the ciphertext when the number of attributes are increased.

The execution times needed for the Secret Key Generation, Encryption and Decryption are shown shown in Figure 4.10. The execution time for all these operations gradually increases as the number of attributes increases. In addition, Key Generation and Decryption have almost the same execution time and encryption has the lowest execution time.

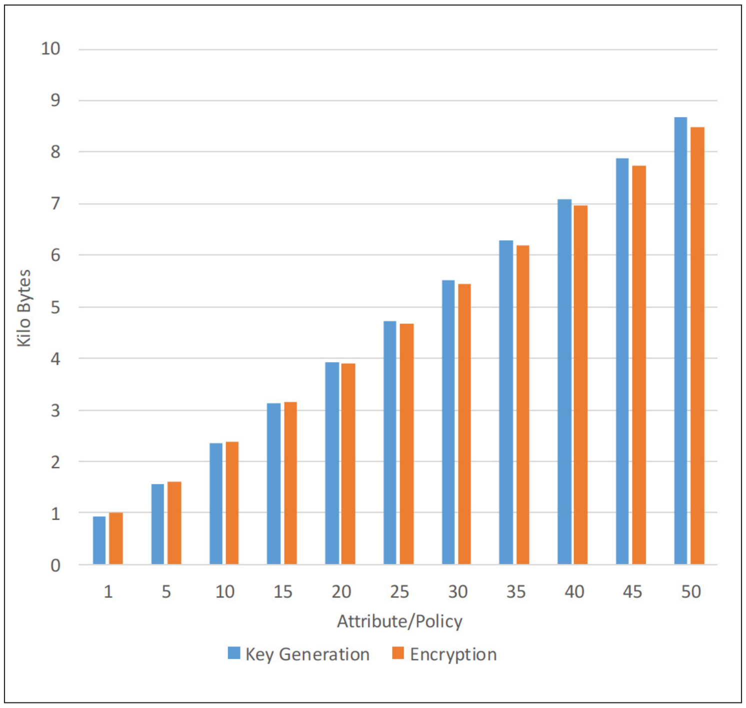


Figure 4.9 File Size

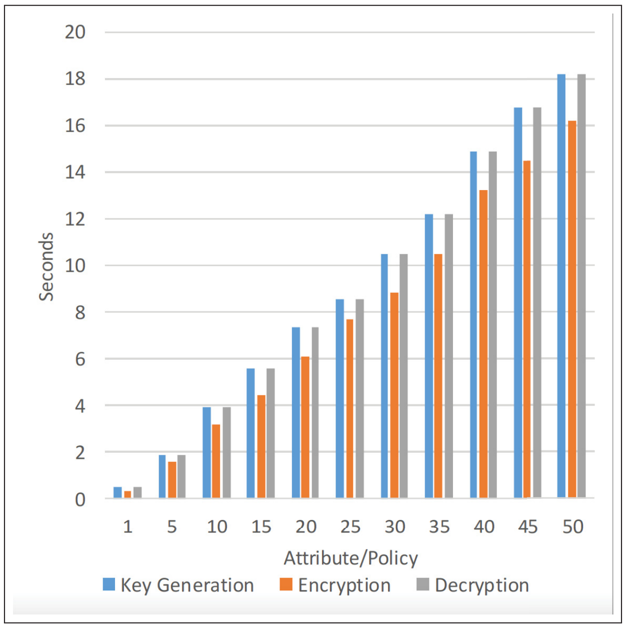


Figure 4.10 Execution Time

4.5 Conclusion

We proposed in this paper a novel framework for defining and enforcing context-aware security policies for Android applications. An extension of Android's security framework is presented to improve the standard permission control provided by the Android system, by leveraging an asymmetric encryption technique called Attribute-Based Encryption (ABE). The proposed technique provides an effective lightweight adaptable context aware encryption model. Moreover, the framework has adopted a policy language which allows customized context-aware policies. The policy language provides the ability to add or edit a policy through a dedicated system service based on context-aware information. Further context monitoring considers various parameters including time, location, device status, etc., while provides the ability to define which permissions to grant, and others to deny, for each of the defined contexts. Furthermore, the implemented security framework is capable of granting a secure inter-application data communication, by encrypting all requested sensor's sensitive data and making it available only for the authorized applications according to the context-aware policies defined. Thorough experiments have been performed demonstrating the efficiency of the proposed solution.

CHAPTER 5

SECURE ADAPTIVE CONTEXT-AWARE ABE FOR SMART ENVIRONMENTS

5.1 Introduction

Security and privacy have become a headache for the industry to cope with the current technological trends and also with the future vision of 6G networks which will impact every interconnected device by including intelligent connections. Context-awareness and encryption can be used to tackle these challenges coming from the industry. Context-awareness is a term that is used for representing the case where computer and embedded devices sense and react according to the changes in their environment. First introduced by Schilit and Theimer (Schilit & Theimer, 1994) in 1994, a context-aware system acquires, understands, recognizes the context, and takes an action according to that precise context. Context-awareness evolved from desktop applications, web applications, mobile computing, pervasive/ubiquitous cloud computing, to the Internet of Things (IoT) over the last years. Moreover, the evolution of the 6G networks is continuously evolving to match the needs of the future IoT smart, self-adoptive applications. Nowadays, billion of IoT devices are connected through machine-to-machine (M2M) and these interrelated devices are provided with unique identifiers and the ability to transfer data over a network with improved efficiency, accuracy, and economic benefit of the IoT environments. Therefore, new applications and business models of the future IoT require new performance criteria such as big data, multi-devices, limited access control, security, privacy, and regulations.

Beside the massive scale of connected end devices, new technologies of multi-devices have been advanced, making identity and access managements more complex while bringing new security and privacy challenges. Smartwatches are good examples of devices that complement Smartphones with the capabilities to check time, messages, emails, notifications, and many more functionalities with easier accessibility. Another example is vehicles equipped with sensors and internet access and connected to other devices, both inside and outside of vehicles, to provide additional benefits, such as traffic alerts and emergency assistance. Section 5.4 covers more of these solution examples and use-cases in detail.

While the success of all these technological advancements is obvious, the complexity that they add to both identity and access managements is crucial. The massive volume of data generated by these devices raises the need for context-aware, adaptive access control solution to avoid any misuse or leak of confidential information to unauthorized parties.

Moreover, this rapid growth will evolve the exchange of big data to build these smart and self-conscious autonomous environments. According to a study by Zaslavsky et al.(Georgakopoulos, Zaslavsky & Perera, 2012), it is expected that the total amount of data on earth will reach up to 35 ZB in 2022. Thus, big data becomes more challenging and raises new IoT and 5G requirements for higher levels of access control, context-awareness, privacy, and security.

Besides the big data challenges, the protection of personal data becomes very relevant for the adoption of these technologies. Therefore, it is critically important to properly understand the main aspects of current regulations which have an impact on 5G and IoT security. The new regulations are mainly proposed to enhance, unify and protect individuals' data. The European Union (EU) has formulated and planned to implement and enforce a new General Data Protection Regulation (GDPR)(Text, 2012) . GDPR is anticipated to protect the export of personal data within and outside the EU. Also, the USA is working to protect costumers, maintain competitions and advance organizational performance by forming the Federal Trade Commission (FTC) and Federal Information Security Management Act (FISMA). FTC and FISMA follow dynamic and effective law enforcement with principle mission of protecting consumer privacy (ftc, 2012) (cis, 2022). All These regulations impose that service providers cannot collect personal information that is not required while data collection, storage and processing have to be done securely. A service provider (SP) should keep the data of the users only during the business period and the user is the one who has the right to grant access to his/her data. Hereby the continuous success and future development of 6G and IoT will depend on the ability to adopt and comply with these regulations.

Accordingly, as discussed in the literature section, several studies have covered many context-aware aspects (Sarker *et al.*, 2020; Nawrocki *et al.*, 2020; Mshali *et al.*, 2018; Alkhresheh *et al.*, 2018; Sarker & Salah, 2019; Phung *et al.*, 2017; Inshi *et al.*, 2020; Selvan & Mahinderjit Singh, 2022; Kim & Chung, 2020; Michalakis & Caridakis, 2022; Kavitha & Ravikumar, 2021) of which two main aspects are still challenging: The first aspect is the lack of a context-awareness broker that can manage the complex identities of interconnected sensors from different devices. In such situations, context-aware policies will play the main role in deciding what and when data needs to be processed or shared, and much more. In addition, the current smart IOT environments need such context-awareness broker as central points of control, as different middle-ware solutions developed by different parties will be employed to connect to sensors for collecting, modeling, and reasoning context information.

The second aspect is updating contexts based on the context information, while the data aggregated by the sensors needed to be shared with the authorized providers or observers, consents are not fixed and can be updated not to share date at certain contexts. In our study as shown in figure 5.1, our main contexts are Activity, Identity, Time and Location. Also, we are collecting data from multiple devices like watch, car and tablet which are the viewed as input for the contexts. For example, consent can be updated to share data when switching between devices, switching between users, or not to share data based on location and time (meeting, enterprise, home). Also, consent can be updated to share data again based on events, such as an insurance company needing to restore data to know what caused an accident. Therefore, to effectively perform this aspect, we need an intelligent dynamic adaptation of contexts, which can be achieved through smart learning techniques. In such situations, machine learning techniques will take the context information as inputs to learn models able to adapt to the consents accordingly.

Therefore, the context broker will coexist with different middlewares by managing the complex identities of interconnected sensors with multi-devices for different providers. Where in these situations, context-aware policies will play the main role in deciding what and when data needs to be processed or shared and much more. In addition, the current smart IoT environments need such context broker as central point of control, as different middle-ware solutions developed

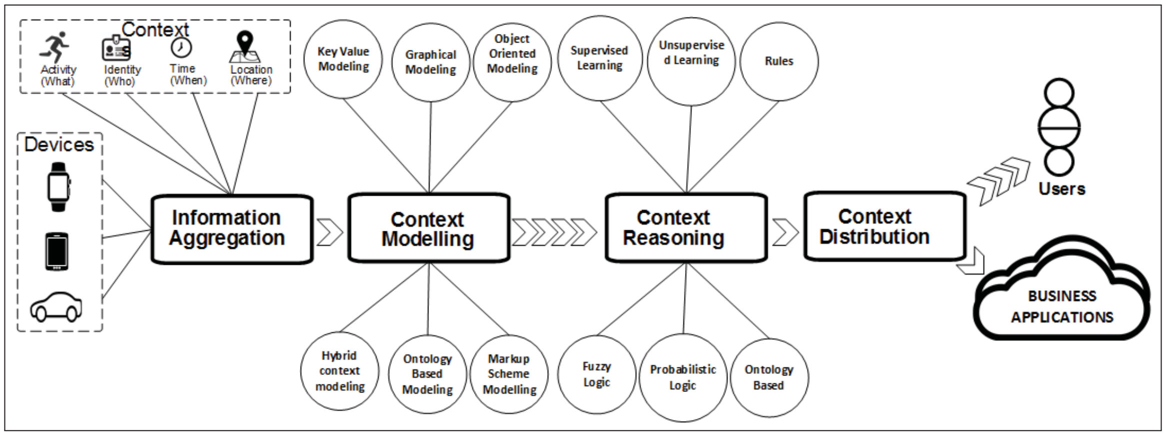


Figure 5.1 Context-awareness Life-Cycle

by different parties will be employed to connect to sensors, collect, model, and reason context. When a subscriber device wants to send data to the business application or application server, it has to authenticate itself to the broker to get the decryption key from the KMS. When the service Provider (SP) wishes to decrypt the data, it has to authenticate itself to the broker to get the decryption key from the KMS. The Broker will check the context and the access privileges for that data and the SP. If the SP is authorized, he will be able to see the data. The details of these processes are shown in the section 5.2.

Besides all above challenges, security is a key requirement to the evolution of the new automated smart IoT environments that provide security services and identity management. This growth raises the need to protect data exchange between different entities by many ways (e.g., authorization, automated access control, and data encryption).

Ultimately, smart Context Awareness Encryption is the key solution to the new secure smart systems evolution. The present novel solution discloses a Context Aware, Adaptable, Intelligent and Lightweight Security Solution. The solution is extending our previous work LCA-ABE (Inshi *et al.*, 2020) to deliver our adaptable data consent, automated access control and secure end-to-end communications between different users, network operators and service providers. By aggregating, modeling, and reasoning context information and then updating consents accordingly in autonomous way. Finally, the solution is fulfilling the new imposed privacy regulations, considering the 5G technologies, leveraging full power of IoT security, Multi-Environments, and access control of big data.

The main contributions of the research are as follows:

- A Context-Aware, Adaptable, Intelligent and Lightweight Security Solution.
- A novel intelligent dynamic creation of context aware policies, which has been achieved through smart learning techniques.
- Providing a context Awareness dynamic encryption model, by leverages Attribute-Based Encryption (ABE).
- Formulating the ABE context-aware policies based on machine learning techniques.
- A solution that fulfills the new imposed privacy regulation.

5.2 Architecture

In this research, we propose a new approach for fine-grained privacy and confidentiality using context awareness and ABE for access control and data security. We are extending our previous work Inshi *et al.* (2020) and adding extra components to make the framework dynamic and adaptive with the context. Figure 5.2 shows our architecture for smart-environment systems. The architecture is divided into multiple modules, which are as follows:

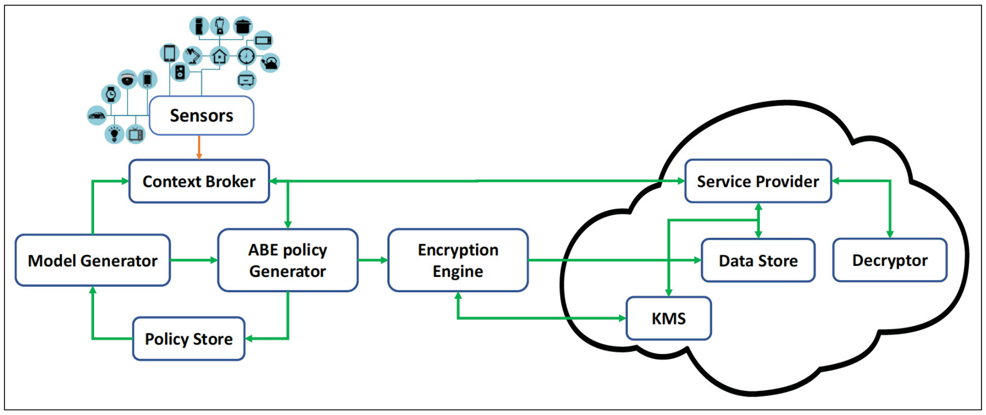


Figure 5.2 Proposed Architecture

- The **context broker** is the entry point for all the data coming from the sensors and different services, and it also acts as a medium for the services to access the sensors and service data. The *context broker*has two functionalities:
  - When the *context broker* receives data, it generates a context based on the location, time, activity, etc., as attributes, and transfers the data and the attributes to the *ABE policy generator* for further processing.
  - When the *context broker* receives a request from a *service provider* , it retrieves the context from the environment and uses the classification algorithm to decide whether the *service provider* has the rights to access the data. If the *service provide r* has access, then the *context broker* gives the *service provider* access to the database to access the data; otherwise, it does not. Once the decision is completed, the context and the decision are both sent to the policy storage for future use.
- The **ABE policy generator** uses the attributes received from the *context broker* and uses the model generated by the *model generator* to generate a CP-ABE policy, and then forwards the data and the policy to the *encryption engine* . Furthermore, it sends the policy to the *policy store* as a dataset entry for future use.
- The **encryption engine** encrypts the data using the policy received from the *ABE policy generator* using CP-ABE and saves it in the database.
- The **policy store** is a database that stores the context and decision for the service providers, as well as the ABE policy generated for each context, all of which will be used by the *model generator.*
- The **model generator** periodically collects the data stored in the *policy store* to generate machine-learning models, which will be used by the *context broker* and *ABE policy generator* . For the *context broker* it uses the naive Bayes classification model, while for the *ABE policy generator* it generates a Markov chain.
- The **key management system (KMS)** generates and stores all the necessary keys, such as the master key, public key, and private key, for the whole ecosystem.
- The **decryptor** is a decryption module used by the service providers to decrypt the data they requested. The *decryptor* will try to decrypt the data using the secret key from the service

provider, and if the service provider has access to the data, then it will return the data, else it will return denied access to the data.

- The **data store** is storage in the cloud, or can be in the device itself, where the data are stored after the encryption process in complete.
- The **service provider** is the application or cloud service that the user has subscribed to.

Figures 5.3 and 5.4 explain the sequence of operations for our framework. Figure 5.3 shows the sequence diagram for the data life cycle when it arrives to the context broker from the sensor. Figure 5.4 shows the steps of operation being performed when a service provider wants to access a sensor's data.

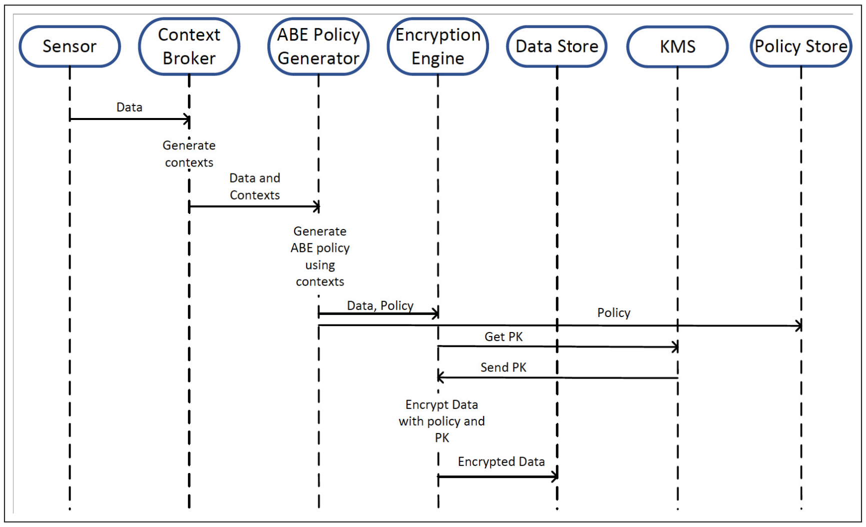


Figure 5.3 Sequence Diagram for sensors generating data.

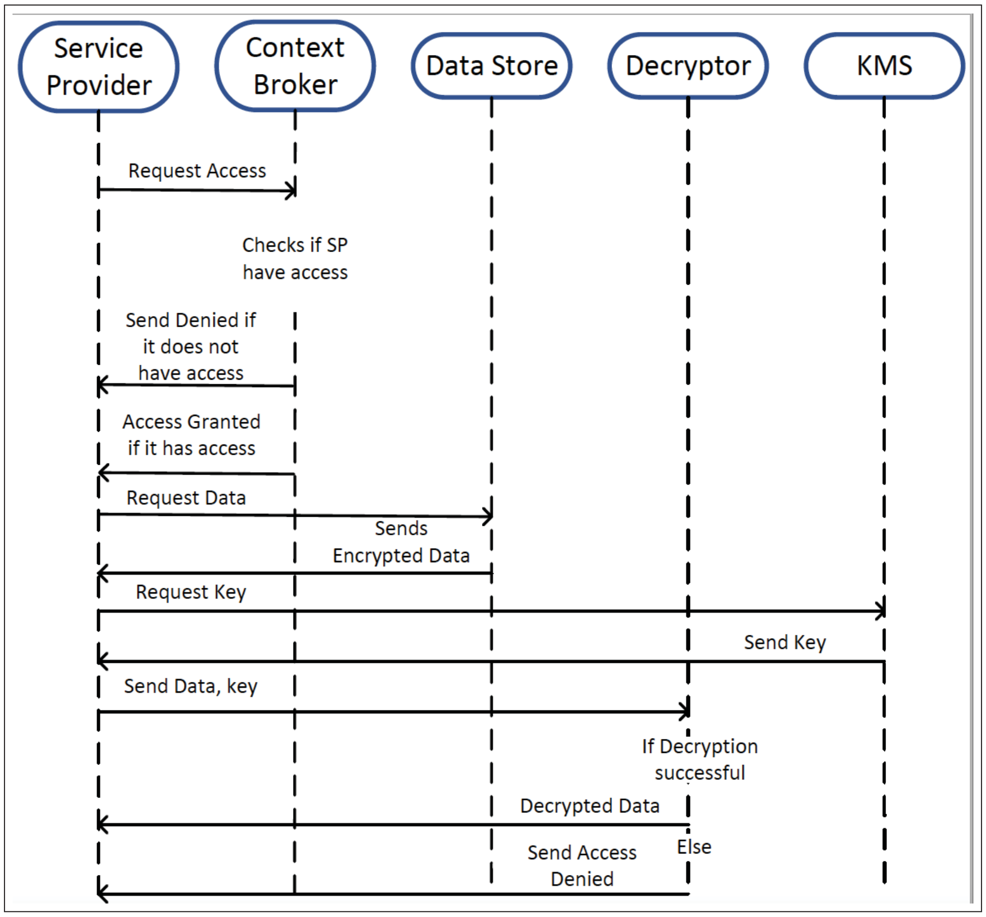


Figure 5.4 Sequence Diagram for Service Provider requesting data.

5.3 Problem Definition

IoT smart applications must be adaptable, intelligent, and secure in order to fulfill the newly imposed privacy regulations, defining and updating consent according to the user's context information in autonomous ways. Furthermore, the execution of the context-aware policies for IoT services and devices must be dynamic, lightweight, and platform-specific. In order to tackle the dynamic nature of the IoT environment, we need to solve the autonomic access control problem and formulate an adaptive policy that can be generated on the basis of context. We present a formal definition of our research problem for context awareness and ABE. In this section, we perform the derivation of formulas for context-aware classification and ABE policy generation. In this research, the problem definition for the context-aware ABE is conducted in two phases, namely:

- The applications need to access the data, but due to security reasons, that needs to be controlled based on the permission, context, source, and application details. The access granted to the application is either “Allow” or “Deny”, which is a binary classification problem wherein the permission, context, source, etc., can be viewed as features, and based on these features, we need to classify whether the application will be granted access or not.
- The context provided by the system needs to be transformed into ABE policy automatically, using operators such as “AND” or “OR” and contexts such as “time”, “location”, etc., which can be viewed as prediction problems. We need to predict the best possible operators and contexts for encrypting the data that can satisfy the user’s behaviour.

5.3.1 Context-Aware Access Control

In a smart environment, there are applications and cloud services that require ambient data generated by the sensors and other applications in order to provide the user with a better experience and ease of use. However, these data need to be shared with the applications and cloud services based on the user permissions.

In this research, we extended the context-aware policy definition of LCA-ABE Inshi *et al.* (2020) for the formulation of our context-aware access control. An application or cloud service *App* contains a list of information, such as name, class, visibility, and API, which we denote as *App* = (name, class, visibility, API). Permission of an *App P*, which is *P* = (name, resources, securityLevel), where name is the name of the permission, resources can be personal data, calendar, camera, etc., and securityLevel is the level of security of the permission, such as normal or dangerous. The context *c* is the circumstances during which the data are generated from the sensors and application. The context can be the time when the data are generated, location of the data, activity of person using the device, etc. The context is represented by *c* = (time, location, activity).

Based on our context-aware policy definition, a context-aware rule for a fitness application can be written as $\text{Allow}_{gps}(\text{Fitness}) \leftarrow (\text{gps} \in \text{Fitness.APIs}) \wedge (\text{P.resources} = \text{gps}) \wedge (\text{c.activity} = \text{exercise}) \wedge (\text{c.time} = \text{7am}) \wedge (\text{c.location} = \text{park})$. Another example, $\text{Deny}_{notifications} \leftarrow ((\text{notification} \in \forall \text{ APIs}) \wedge (\text{notification} \neq (\text{gps} \vee \text{maps}) \wedge (\text{c.activity} = \text{driving})$ will be applied to the user's car, so that all the notifications from the cellphone are blocked except the navigation system. In order to automate the decision process of context-aware policies, the policies were transformed into tables with multiple features that can be used by a machine-learning algorithm, as shown in Table 5.1. Using a simple classification algorithm, we are able to solve the context-aware control problem. For instance, as shown in Inshi *et al.* (2023b), if the user has to include all the rules for a device's sensor data for access control, then it will become a hassle and be infeasible, so we need to use an automated system that will grant the access for the device data based on the user's behaviour. So in order to automate this process, we need to use machine learning, where the ML algorithm can automatically grant the access permission and generate the access policy on behalf of the user.

Table 5.1 Sample Transformation of Context aware policy

Application				**Permission**			**Context**			**Source**	**User**	**Decision**
Name	**Class**	**Visibility**	**API**	**Name**	**Resource type**	**Security Level**	**Time**	**Location**	**Activity**			
Fitness	Well-being	Background	Get data	GPS	gps	Medium	700	Park	Jogging	GPS	Bob	Allow
Fitness	Well-being	Background	Get data	GPS	gps	High	900	Office	Meeting	GPS	Bob	Deny
Map	Navigation	Background	All	All	notification	Medium	800	Hollywood Boulevard	Driving	Apps	Bob	Deny
Map	Navigation	Background	GPS	GPS	notification	Medium	800	Hollywood Boulevard	Driving	Apps	Bob	Allow

5.3.2 Dynamic ABE Policy

The data which are being generated from the different sources need to be encrypted before being stored in the device itself or sent to the cloud. Furthermore, based on the context of the user, the data privacy will change. ABE, which provides encryption of data along with access control to the data, is the most suitable solution for the context-aware environment Annane, Alti & Lakehal (2022a); Annane, Alti, Laouamer & Reffad (2022b). In order to encrypt data, ABE needs a policy that contains attributes and operators that combine these attributes, e.g., "Bob AND 7AM AND Exercise AND Well-being" is a policy for encrypting the GPS data that makes sure that the fitness application can access the data for that context. However, the catch for this process is to generate the ABE policy dynamically, which is challenging. In order to hurdle the challenge, we introduce a new procedure for the automatic generation of the ABE policies.

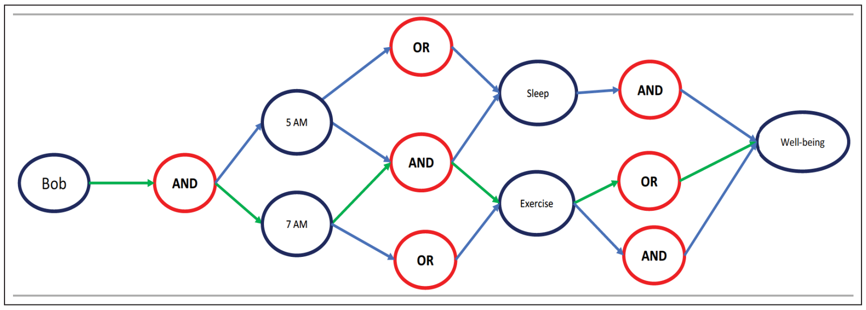


Figure 5.5 A visual representation of ABE policy

Figure 5.5 shows an example of ABE policy based on the context provided by the system. The system has all the attributes, i.e., the contexts, the only thing missing being the operators, which can be easily determined based on the user's predetermined behaviour. This problem can be viewed as a longest path problem, where we need to find the best route from beginning to end.

The application, permission, context, and users, which are attributes denoted by A, are assumed as cities, and the operators 'AND' and 'OR' are the cost for travelling from one city to another. The cost is calculated based on the probability of the operators based on the behaviour of the user, based on the example in Figure 5.5. For reference, the transition matrix will look like Table 5.2. The probability of each state will be calculated using $S_n = S_0 \times Q^n$, where S_0 is the initial state vector, which is Bob, and Q is the transition matrix to move from state i to state j, as shown in Table 5.2. In our research, we did not intend to find the path with the best cost, but searched for the longest one because that is where we will find the best attributes for these contexts, which will eventually become the policy for the data's ABE encryption. Using Dantzig–Fulkerson–Johnson formulation, we can find the path without having subtours or loops. So the equation without loops is $A_{ij} = \begin{cases} 1 & The\ path\ goes\ from\ A_i\ to\ A_j \\ 0 & Otherwise \end{cases}$.

Table 5.2   Transition Matrix

	Bob		5am		7am		Sleep		Exercise		Well-being	
	AND	OR	AND	OR	AND	OR	AND	OR	AND	OR	AND	OR
Bob	0	0	0.3	0	0.5	0	0.02	0.02	0.03	0.04	0.03	0.06
5am	0	0	0	0	0	0	0.4	0.3	0.1	0.05	0.06	0.09
7am	0	0	0	0	0	0	0.15	0.15	0.3	0.2	0.1	0.1
Sleep	0	0	0	0	0	0	0	0	0	0	1	0
Exercise	0	0	0	0	0	0	0	0	0	0	0.4	0.6

By taking $C_{ij} > 0$ to be the cost from attribute i to attribute j, the formulation can be derived using an integer linear programming problem:

$$\begin{aligned} &max \sum_{i=1}^{n} \sum_{j\neq i, j=1}^{n} C_{ij}A_{ij} \quad where, \\ &\sum_{i=1, i\neq j}^{n} A_{ij} = 1 \\ &\sum_{j=1, j\neq i}^{n} A_{ij} = 1 \\ &\sum_{i\in Q} \sum_{j\neq i, j\in Q} A_{ij} \leq Q - 1 \qquad \forall Q \not\subseteq 1, \ldots, n, Q \geq 2 \end{aligned} \tag{5.1}$$

In order to solve this problem, we have utilized a discrete Markov chain to find the best attributes for the policy chain based on the context provided. The Markov chain is a stochastic statistical model, where the future state depends on the past states. In our case, the total number of contexts available in the environment are the states. So, the following equation (Equation (5.2)) is the classical Markov equation of higher order Raftery (1985), and by taking as state space the ordered m-tuples of A, it is used in order to find the optimal policy chain of that particular context for encrypting the data.

$$\mathbb{P}(A_n = a_0 A^{n-1} = a_1, \ldots, A^{n-k} = a_k) = max \sum_{i=1}^{k} \lambda_i q_{a_0} a_i \tag{5.2}$$

where $\sum_{i=1}^{k} \lambda_i = 1$, λ_i is non-negative, k is the total number of attributes, and Q= [q $_{ij}$] is the transition matrix where the sum of the column is equal to one. Using Ching, Huang, Ng & Siu (2013), the generalization of Raftery (1985) as follows:

$$A^{n+k+1} = max \sum_{i=1}^{k} \lambda_i Q_i A^{(n+k+1-i)} \tag{5.3}$$

With Equation (5.3), we will be able to find the best path with the highest probability given any number of attributes and form the policy for the ABE encryption. Using Equation (5.3) and the transition matrix, the result for different states is calculated in the form of Bob 7am (0.3), Exercise (0.3), Well-being (0.6). Then, using the transition matrix and the value, we will find the operators, and the policy will look like Bob AND Time = "7am" AND Exercise OR Well-being.

5.3.3 Algorithm

We have utilized two algorithms to solve the context-aware-based encryption problem. Algorithm 5.1 is designed for dynamic access control and Algorithm 5.2 is for generating the ABE policy for encrypting the data based on the context, where Algorithm 5.1 is hosted in the context broker and Algorithm 5.2 is in the ABE policy generator. We have created the initial transition matrix using a probabilistic method. The descriptions of the algorithms are as follows:

Algorithm 5.1 performs the decision making for the application when it wants to access specific data from the data store. The algorithm takes parameters as shown in Table 5.1 and uses a trained deep-learning model to take the decision, whether it has access to the data or not, with respect to the current contexts.

Algorithm 5.2 is called whenever the system generates data. It automatically generates an ABE policy based on the contexts (attributes) provided by the system. The algorithm takes attributes as shown in Table 5.1, which contains application details, permission of the application, context, data source, and the user who is currently using the device. Furthermore, the algorithm requires a transition matrix that contains the probabilities of each attribute with another attribute. The

Algorithm 5.1 Access Control

```
Input: Parameters[Application, Permission, Context, Source, User], Trained Model
Output:
1 Result = Classify the Parameters using Trained Model
2 if Result = Allow then
3     give access to the data
4 else
5     return "Denied" to the application or service
6 end if
```

transform array contains the probability, which corresponds to the operators. The algorithm then uses Equation (5.2) to calculate the best path for the attribute and stores the value of the path it took to reach the goal. After that, the algorithm uses the values and the transform array to find the operators. Finally, using the path and the operators, it generates the ABE policy for the current context and sends it to the encryption module.

Algorithm 5.2 Policy Generator

```
Input: Attributes[Application, Permission, Context, Source, User], Transition Matrix
       M, Transform Array T
Output: policy
1 Path = Evaluate eq 5.2 using Attributes and M
2 Operators = Get operators using Path and T
3 for All value in Path do
4     Find the value of Path[i] corresponding to T
5     Append value Operators
6 end for
7 for All value in Attributes do
8     Append Attributes[i] AND Operators[i] to policy
9 end for
```

5.4 Experimentation and Evaluation

In this section, we evaluate our framework in terms of resource utilization as well as evaluate the algorithms. We also present four use-cases used during our experimentation. The use-cases represent the different rules that can be used based on the user's predefined context and their representation of the ABE policy.

5.4.1 Use-Cases

Figure 5.6 presents our solution's examples and use-cases. The descriptions of the use-cases are as follows:

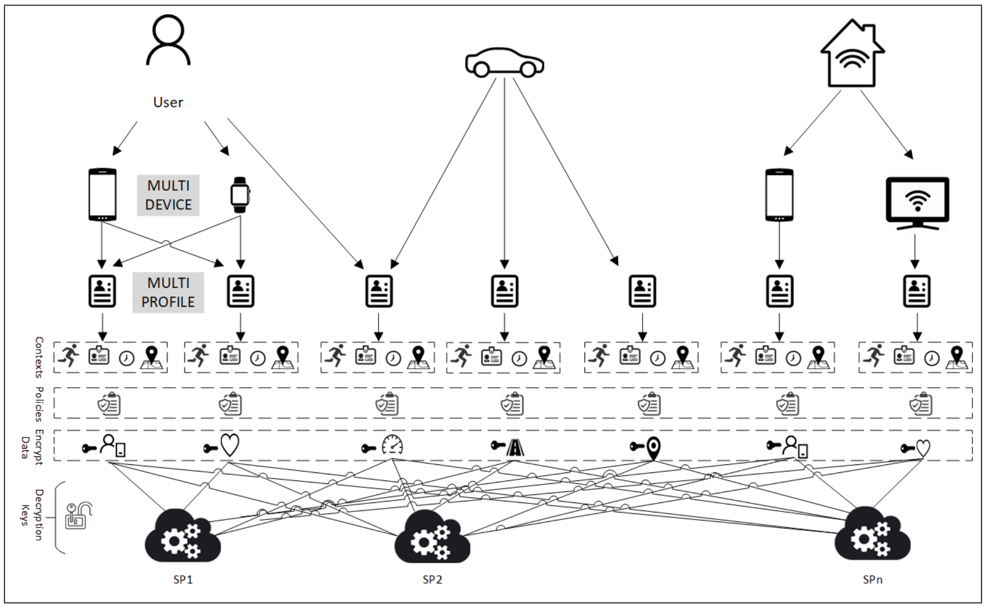


Figure 5.6 Solution Examples Model

5.4.1.1 Scenario 1: Exercising

Bob is going for his regular morning run and is using a fitness application while he exercises. According to normal fitness apps, at the time of installation, Bob was required to allow/deny this app access to some of his smartphone/smartwatch resources. Therefore, using our model, Bob will be able to manage his access policies for while he exercises according to his needs within a specific location and during specific time frames. Examples of the resources that Bob is sharing include location, heartbeat, and accelerator.

Our model defines the application, permissions (resources), and contexts as discussed in Section 5.3. We assume the following are the permissions that Bob has set to control access to his device:

1. Rule 1: The fitness app may have access to heartbeat and accelerator meter sensors while exercising.
2. Rule 2: The fitness app may only access the heartbeat and accelerator meter sensors while Bob is at the park.
3. Rule 3: The fitness app may only access the heartbeat and accelerator sensors at 6:00 am or 6:00 pm.
4. Rule 4: The fitness app may never share/access the location.

The rules are be represented as follows:

1. The context-aware access control policy:
   a. $\text{Allow}_{FitnessApplication} \leftarrow \text{Activity}_{Sport} \wedge \text{Location}_{park} \wedge \text{Time}_{6am} \vee \text{Time}_{6pm} \leftarrow \text{Data}_{heartbeat,accelerator}$
   b. $\text{Deny}_{FitnessApplication} \leftarrow \text{Activity}_{Sport} \wedge \text{Location}_{park} \wedge \text{Time}_{6am} \vee \text{Time}_{6pm} \leftarrow \text{Data}_{location}$
2. The ABE policy at that specific moment, which are the data collected at 6am for different sensors, is as follows:
   a. $\text{Data}_{heartbeat} \rightarrow$ Application = ‘Fitness Application’ AND Location = ‘park’ AND Time = ‘6am’ AND Activity = ‘Sport’
   b. $\text{Data}_{accelerator} \rightarrow$ Application = ‘Fitness Application’ AND Location = ‘park’ AND Time = ‘6am’ AND Activity = ‘Sport’
   c. $\text{Data}_{GPS} \rightarrow$ Application = ‘$\neq Fitness$’ AND Location = ‘park’ AND Time = ‘6am’ AND Activity = ‘Sport’

5.4.1.2 Scenario 2: Meeting with a Supervisor

Bob is a graduate student who has scheduled regular meetings with his supervisor, Alice. According to Alice and the school regulations, all graduate students must respect some policies regarding access to their device resources while attending meetings, during specific time frames, and in specific locations (offices, labs, or meeting rooms). Examples of the resources that Alice could allow/deny a student access to include the microphone, camera, GPS, and notifications.

We assume the following are the policy rules that Bob sets to fulfill Alice's meeting requirements:

1. Rule 1: No app can have access to the voice recorder and camera while in meetings.
2. Rule 2: No app can have access to the voice recorder and camera while in a meeting in the meeting room.
3. Rule 3: No app can have access to the voice recorder and camera while in a meeting in the meeting room during specific time frames.
4. Rule 4: During meetings, no app may ever share/access the location.

Using our model, the rules for this scenario are as follows:

1. The context-aware access control policy:
   a. $\text{Deny}_{Application} \leftarrow \text{Activity}_{Meeting} \wedge \text{Location}_{MeetingRoom} \vee \text{Time}_{\geq 10am} \wedge \text{Time}_{\leq 11am} \leftarrow \text{Data}_{Microphone,Camera,GPS,Notification}$
2. The ABE policy at that specific moment, which are the data collected at 11am for different sensors:
   a. Data_{GPS} → Application = 'All' AND Activity = 'Meeting' AND Location = 'Meeting Room' OR Time = '11am'
   b. Data_{Camera} → Application = 'Personal' AND Activity = 'Meeting' AND Location = 'Meeting Room' OR Time = '11am'
   c. $\text{Data}_{Microphone}$ → Application = 'Personal' AND Activity = 'Meeting' AND Location = 'Meeting Room' AND Time = '11am'

5.4.1.3 Scenario 3: Driving a Car

Bob has a car insured by a well-known insurance company. Bob does not want to receive any annoying notifications while driving. Furthermore, he does not want to share all of his car's/smart device's resource information with the insurance company while driving. Therefore, Bob defines the attributes and context policies to allow/deny the insurance app access to some of his smart car/smartphone/smartwatch resources according to his consent. Moreover, our model allows Bob to update his consent anytime to share data again based on an event, such as the insurance company needing to obtain the data to determine what caused an accident.

Examples of the resources that Bob could allow/deny the insurance company app access to include GPS tracking, navigation, speed, and hours of driving. We assume the following are the permissions that Bob sets to control access to his devices while driving his car:

1. Rule 1: The insurance app has no access to GPS tracking and navigation while driving.
2. Rule 2: No app can send/receive notifications while driving.

The rules for this scenario are as follows:

1. The context-aware access control policy:
   a. $\text{Deny}_{Application} \leftarrow \text{Activity}_{Driving} \leftarrow \text{Data}_{Notification}$
   b. $\text{Deny}_{Insurance} \leftarrow \text{Activity}_{Driving} \leftarrow \text{Data}_{GPS,Navigation}$
2. The ABE policy at that specific moment, which are the data collected at 5pm for the GPS sensor:
   a. $\text{Data}_{GPS} \rightarrow$ Application = '≠ Insurance' AND Activity = 'Driving' AND Location = 'Peel' AND Time = '5pm'

5.4.1.4 Scenario 4: Smart Home

Bob is staying home after finishing his job and is watching a TV show with his family on their smart TV. Most applications installed on smart TVs have access to many resources that can leak some private information, such as location and favourite shows, etc. Bob has to allow/deny these apps access to some of their smart home/smart TV resources. Therefore, using our model, Bob will be able to manage their access policies while staying home according to their needs within a range of specific activities (sleeping, watching TV, and eating) and during specific time frames. Examples of the resources that Bob is sharing: location, GPS, and heartbeat.

We assume the following are the permissions that Bob sets to control access to their smart home resources:

1. Rule 1: All health apps may have access to heartbeat sensors while sleeping.
2. Rule 2: All apps may only access heartbeat and accelerator sensors while Bob is at the park.
3. Rule 3: No app may ever share/access the exact location while Bob is at home.

The rules for this scenario are as follows:

1. The context-aware access control policy:
   a. $\text{Allow}_{FitnessApplication} \leftarrow \text{Activity}_{Sleeping} \leftarrow \text{Data}_{heartbeat}$
   b. $\text{Deny}_{Application} \leftarrow \text{Activity}_{WatchingTV} \leftarrow \text{Data}_{Notification}$
   c. $\text{Deny}_{Application} \leftarrow \text{Activity}_{Sleeping} \leftarrow \text{Data}_{Notification}$
2. The ABE policy at that specific moment, which are the data collected at 11pm for the GPS sensor:
   a. $\text{Data}_{heartbeat} \rightarrow$ Application = 'Fitness Application' AND Location = 'park' AND Time = '11pm'AND Activity = 'Sleeping' OR Activity = 'TV'
   b. $\text{Data}_{accelerator} \rightarrow$ Application = 'Fitness Application' AND Location = 'park' AND Time = '11pm' AND Activity = 'Sleeping' OR Activity = 'TV'
   c. $\text{Data}_{GPS} \rightarrow$ Application = '$\neq Fitness$' AND Location = 'park' AND Time = '11pm' AND AND Activity = 'Sleeping' OR Activity = 'TV'

5.4.2 Implementation

In this section, we explain the implementation of the main algorithms and dataset for our adaptive context-aware encryption framework.

5.4.2.1 Dataset

The principal role of the system is to automatically grant the access control of the sensor and to generate policy for ABE. For this reason, we have collected data by monitoring Fitbit and a person's cell phone applications (mainly the google map data) for a period of one month. The google map data are collected from the historical information stored in the map history, and the Fitbit data are gathered from the personal historical data from their website. The types of data which are collected are as follows:

- Permissions requested by applications include the name of the sensor it is accessing, the type of resources, and the security level;
- Google Map data while driving, which includes the latitude and longitude;
- Permissions of the services, mainly the GPS, and services requested by the Fitbit;
- Activity tracking, which includes sleep, exercise, swimming, heart rate, SpO2, steps, time in heart-rate zone, and calories burned during that time.

After the collection of the data, we had to removed excess unnecessary features and information such as heart-rate zone, calories burned, etc. Then, we aggregated the data and categorized them into respective features, which is similar to Table 5.1. Some of the features, such as the source of the data, permission name, and resource type, are correlated, which was not covered in this article. The data for the first dataset are balanced as we manually labelled the data for the access control that will be used by algorithm 1 for classification. The second dataset contains the ABE policy, which we manually created to test whether Algorithm 5.2 is able to generate the desired policy. This dataset is mainly used to create a transition matrix by using the probabilistic methods, which is similar to Table 5.2.

5.4.2.2 Deployment

To implement our framework, we adopted the implementation provided by Inshi *et al.* (2020) and implemented the classification and policy generation algorithms, which were not available in that framework. We employed Scikit-learn to implement the naive Bayes classifier. We normalized the dataset for the classification and split the dataset for training and testing into 80% and 20%, respectively, and then executed Algorithm 5.1. To implement Algorithm 5.2, we used python. For the algorithm, which needed a transition matrix to generate the policy, we first normalized the second dataset and used probabilistic analysis to generate the operator values for each context. This way, the algorithm would be able to generate the policy properly. We used

Raspberry Pi 3 Model B as our IoT device and the details regarding the hardware and software are given in Table 5.3. We used Raspberry Pi 3 Model B as our IoT device and a desktop PC for the services. Raspberry Pi hosted the context broker, model generator, ABE policy generator, and the policy store. The desktop PC hosted the service provider, data store, decryptor, and KMS. Communication between Raspberry Pi and the desktop PC was achieved using a python websocket. The details regarding the hardware and software are given is Table 5.3.

Table 5.3 Hardware and Software Specification

Processor	1.2GHz 64 bit quad-core ARM Cortex-A53
RAM	1GB
Storage	16GB eMMC flash Storage
Operating System	Raspberian Debian OS

In our experimentation, we used a python script to determine the resource utilization. The latency was measured using the difference between the time when the data were collected and when the encryption was completed. The decryption latency includes the decryption time along with the communication time to ask the context broker whether it has permissions to access the data or not.

5.4.3 Evaluation

The main objective of this experiments was to study the feasibility of our framework as well as the resource utilization. We have compared our algorithms with other ML algorithms, such as logistic regression, decision trees, random forest, and LSTM.

In order to evaluate the performance of our algorithm, we performed various experiments. For the access control evaluation, we experimented with logistic regression, decision trees, and naive Bayes. For the policy generator evaluation, we experimented with random forest, LSTM, and Markov chains. We analyzed the precision, recall, and F1-score of the algorithm, using the equations provided in Dalianis (2018).

We evaluated the algorithms using our dataset to determine the accuracy, precision, and recall, as well as the training time required to prepare the models, which are shown in Table 5.4. For experimentation and evaluation of the algorithm, we split 80% for training the models and 20% for testing the algorithm performances. As shown in Table 5.4 for the access control, the F-score of the logistic regression is higher, but the precision and training are lower. The decision tree has the lowest training time, but the naive Bayes gives the best results in terms of precision, F-score, and training time. For the policy generation evaluation, the lowest performance is provided by the random forest, but the training time is the lowest. LSTM has an average accuracy, but the training time is the highest among the other algorithms. Markov chain is the model in terms of all metrics. To summarize, from Table 5.4, we see that the accuracy of our algorithms is above 90% and their precision is above 85%.

Table 5.4 Evaluation of the Algorithms

Algorithm		Precision	Recall	F1	Training Time
Access Control	Logistic Regression	0.9183	0.8681	0.8925	0.0086
	Decision Tree	0.9248	0.8348	0.8755	0.0024
	Algorithm 5.1 (Naive Bayes)	0.9515	0.9062	0.9283	0.0037
Policy Generator	Random Forrest	0.8426	0.6775	0.7324	0.4581
	LSTM	0.8936	0.7976	0.8429	34.7515
	Algorithm 5.2	0.8926	0.913	0.9027	2.8564

Figures 5.7–5.9 are the experiments performed to find the resource utilization in our framework. In these experiments, we increased the policy length gradually with increments of five attributes, up to fifty attributes. The policy length is the total number of attributes in the policy. In Figure 5.7 is the CPU consumption with varying policy lengths. The CPU consumption is for encryption gradually increases as the length of the policy increases. On the other hand, decryption has almost the same CPU utilization when the number of policy attributes varies between 30 and 50. The CPU utilization does not go above 50% in the worst case, which means that the length of the policy compromises 50 attributes. Figure 5.8 is the memory utilization for encryption and decryption in megabytes. The memory utilization for encryption is lower than the memory utilization. Another interesting observation is that the memory utilization for decryption stabilizes after 35 attributes and is 25 attributes for the encryption . Lastly, Figure

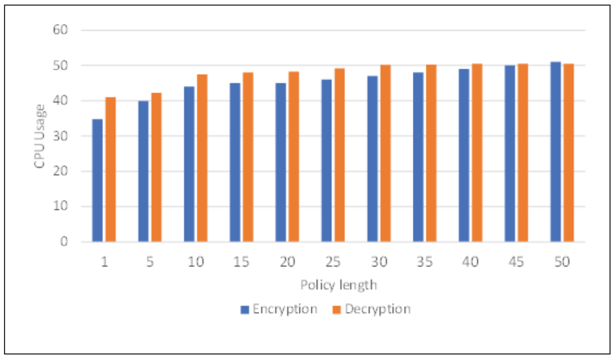


Figure 5.7 CPU utilization

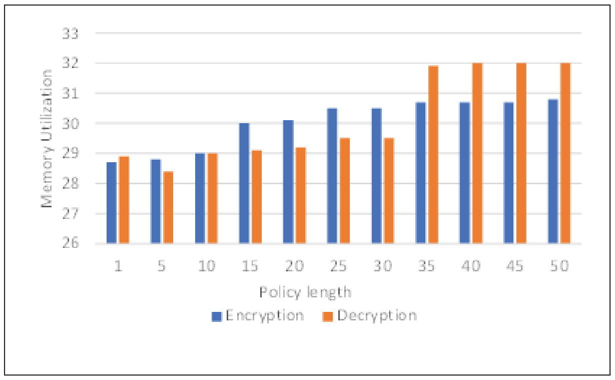


Figure 5.8 Memory utilization

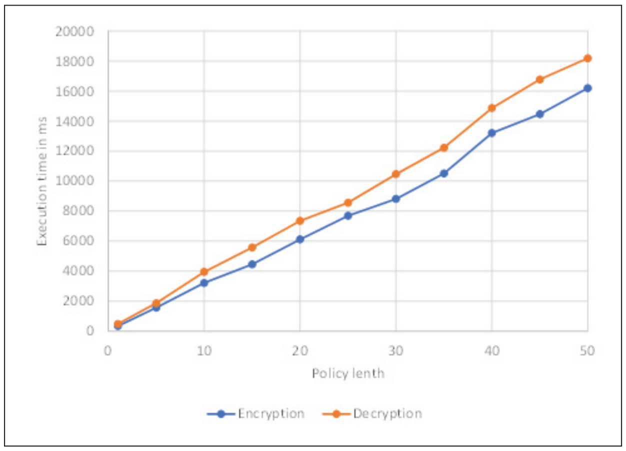


Figure 5.9 Execution time

5.9 is the overall time required for the encryption after data are received by the context broker. The execution time for decrypting is higher as it has to communicate with the context broker to obtain the permission for the data from the data store before it can start the process of decryption. From this figure, we see that, as the policy length increases, the execution time for the whole process increases from 0.5 s up to 16 s for encryption and 18 s for decryption.

In summary, the approach we proposed for developing our context-aware, adaptable, intelligent and lightweight security model consists of two main axes, namely, the dynamic creation of context-aware policies and the context-awareness dynamic encryption model. The conducted in-depth experiments proved the efficiency of both. Our framework ensures that the accuracy of our algorithms is above 90%, and their precision is around 85%, which is considerably high compared to the other approaches studied. However, many industrial aspects are still challenging and could affect our model accuracy rates, such as the complex identity management of multiuser, multiprofile, and multidevice technologies

Furthermore, the implemented lightweight security framework is capable of granting secure interapplication data communication by encrypting all the requested sensitive sensor data and making them available for only the authorized applications according to the generated context-aware policies. Thorough experiments have been performed demonstrating the efficiency of CPU and memory utilization, as well as the execution time for the whole encryption/decryption process. More evaluation criteria might be considered for future improvements.

Finally, we believe that our smart adaptive model will open up promising directions for research and improvements using different ML techniques in heterogeneous IoT environments.

5.5 Conclusion

In this work, we have extended our previous work on LCA-ABE to provide a smart, adaptive, context-aware security model by adopting a novel intelligent dynamic creation of context-aware policies, which was achieved through smart-learning techniques. By leveraging attribute-based encryption (ABE), we also provided a context-awareness dynamic encryption model. Furthermore, our implemented security framework is capable of granting secure inter-application data communication by encrypting all of the requested sensor's sensitive data and making it available only for the authorized applications according to the predefined context-aware policies. Thorough experiments and evaluations, and by leveraging the full power of modern IoT smart environments and the newly imposed privacy regulations, we demonstrate the efficiency of the proposed solution.

CONCLUSION AND RECOMMENDATIONS

This section concludes this doctoral research work. It summarizes the main addressed problems and the presented key contributions, while opens the door for different future research directions.

6.1 Conclusion

Security and privacy of Android and IoT smart applications are essential requirements for the industry to cope with the current technological trends and also with the future vision of 6G networks which will impact every interconnected device by including intelligent connections. Based on our research, Context-awareness and encryption can be used to tackle these challenges coming from the industry. However, existing approaches pose great challenges when building Lightweight Context-Aware Encryption models for Android and IoT Applications. Therefore, addressing such problem has drawn the fundamental objective of this dissertation. Throughout this work, we answered many research questions and we set different objectives to reach the main goal.

In Chapter 2, we focused on android applications that are malicious when only running in particular context. We proposed a rewriting framework that provides an instrumentation technique for security policies enforcement in android applications. The framework allows users to define high-level and context-aware inter-app policies which are then automatically refined to fine-grained enforceable policies. The results showed that the resulting instrumented applications have only 1.2% additional code compared to their original version, which is relatively small. The results also showed that the rewriting time takes no more than 51s and most importantly the success rate was 99.78%, which is significantly high compared to other existing approaches.

In Chapter 3, we addressed problems related to context aware policies for Android applications as its one of the main targets of attackers. We, therefore developed CAPEF, which is a formal policy specification framework that enforces context-aware inter-app security policies to effectively describe users defined consents. Thorough experiments we have performed a study on the efficiency of CAPEF with respect to the size and execution time of the enforced applications. The evaluation results demonstrated the feasibility of our framework and the effectiveness of our policy specification language in enforcing complex context-aware policies on different Android applications.

In Chapters 4 and 5, we have extended our work LCA-ABE to provide a smart adaptive context-aware security model, by adopting a novel intelligent dynamic creation of context-aware policies, which has been achieved through smart learning techniques, and also, providing a context-awareness dynamic encryption model, by leveraged Attribute Based Encryption (ABE). Also, our implemented security framework is capable of granting a secure inter-application data communication by encrypting all requested sensor's sensitive data and making it available only for the authorized applications according to the pre-defined context-aware policies. Thorough experiments and evaluations show the efficiency of the proposed solutions by leveraging full power of modern IoT smart environments and the new imposed privacy regulations.

6.2 Future Research Directions

The contributions presented in this dissertation open the door for interesting future research directions. Although many research efforts have addressed different context-aware scenarios, application and deployment aspects, more efforts are still needed for further improvements. For instance, improving our model by using different ML techniques for varying Android applications in IoT smart environments.

Furthermore, we believe that our Secure Adaptive model can be adapted to many potential researches and applications for both academia and industry professionals where context-awareness, security, privacy and access control is required. Very recently, we have started working in a new domain of Tactical Networks (TN), to solve problems in several military systems such as command, control, communications, intelligence and surveillance in different challenging and heterogeneous networking environments. The presented TN approaches opens the door for interesting future tracks to our smart adaptive context-aware model. Future work could deal with varying network conditions generated by a realistic simulation of scenarios for tactical networks. Furthermore, we could extend our model to support different content and context-aware clustering-based algorithms for the distribution of network state information that can significantly reduce the overhead associated with network state information sharing.

BIBLIOGRAPHY

[Accessed: 18-07-2019]. CloudCrypto. Retrieved from: [online]https://github.com/liuweiran900217/CloudCrypto.

[Accessed: 18-07-2019]. Gartner Says Worldwide Information Security Spending Will Grow 7 Percent to Reach 86 Billion USD in 2017. Retrieved from: [online]https://www.gartner.com/newsroom/id/3784965.

[Accessed: 18-07-2019]. Advantages and disadvantages of certificate authentication. Retrieved from: [online]https://www.ssh.com/manuals/server-zos-product/55/ch06s03s05.html.

Android Developers. Retrieved from: https://developer.android.com/guide/topics/manifest/manifest-intro.

(2010). Retrieved from: Online[Access05/10/2016]url{http://igm.univ-mlv.fr/~chilowi/teaching/subjects/java/android/ipc/index.html}.

(2011). Retrieved from: Online[Access05/10/2016]url{http://developer.android.com/guide/components/fundamentals.html}.

(2011). Retrieved from: http://www.cse.wustl.edu/~jain/cse571-09/ftp/soa/index.html.

(2012). Ftc privacy and data security. Retrieved from: Online[Access18/10/2022]url{https://www.ftc.gov/policy/reports}.

(2022). Cybersecurity and Infrastructure Security Agency CISA. Retrieved from: Online[Access18/10/2022]url{https://www.cisa.gov/federal-information-security-modernization-act}.

Abowd, G. D., Dey, A. K., Brown, P. J., Davies, N., Smith, M. & Steggles, P. (1999). Towards a better understanding of context and context-awareness. *International symposium on handheld and ubiquitous computing*, pp. 304–307.

Akinyele, J. A., Pagano, M. W., Green, M. D., Lehmann, C. U., Peterson, Z. N. & Rubin, A. D. (2011). Securing electronic medical records using attribute-based encryption on mobile devices. *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, pp. 75–86.

Alhanahnah, M., Yan, Q., Bagheri, H., Zhou, H., Tsutano, Y., Srisa-An, W. & Luo, X. (2020). Dina: Detecting hidden android inter-app communication in dynamic loaded code. *IEEE Transactions on Information Forensics and Security*, 15, 2782–2797.

Alkhresheh, A., Elgazzar, K. & Hassanein, H. S. (2018). Context-aware automatic access policy specification for iot environments. *2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC)*, pp. 793–799.

Alkurdi, M. Z., Samra, A. A. & Qunoo, H. (2014). Malware Detection for Android Applications Using SimHash Algorithm.

Ambrosin, M., Conti, M. & Dargahi, T. (2015). On the feasibility of attribute-based encryption on smartphone devices. *Proceedings of the 2015 Workshop on IoT challenges in Mobile and Industrial Systems*, pp. 49–54.

Android. (2011). Android Developers. Retrieved from: https://developer.android.com/about/versions/marshmallow/android-6.0-changes.html.

Annane, B., Alti, A. & Lakehal, A. (2022a). Blockchain based context-aware CP-ABE schema for Internet of Medical Things security. *Array*, 14, 100150. doi: 10.1016/j.array.2022.100150.

Annane, B., Alti, A., Laouamer, L. & Reffad, H. (2022b). Cx-CP-ABE: Context-aware attribute-based access control schema and blockchain technology to ensure scalable and efficient health data privacy. *Security and Privacy*, 5(5), e249. doi: 10.1002/spy2.249.

Arena, V., Catania, V., La Torre, G., Monteleone, S. & Ricciato, F. (2013). SecureDroid: An Android security framework extension for context-aware policy enforcement. *Privacy and Security in Mobile Systems (PRISMS), 2013 International Conference on*, pp. 1–8.

Bany Taha, M., Talhi, C., Ould-Slimane, H. & Alrabaee, S. (2020). TD-PSO: task distribution approach based on particle swarm optimization for vehicular ad hoc network. *Transactions on Emerging Telecommunications Technologies*, e3860.

Bethencourt, J., Sahai, A. & Waters, B. (2007). Ciphertext-policy attribute-based encryption. *2007 IEEE symposium on security and privacy (SP'07)*, pp. 321–334.

Ching, W.-K., Huang, X., Ng, M. K. & Siu, T.-K. (2013). Higher-order markov chains. In *Markov Chains* (pp. 141–176). Springer.

Chowdhury, R., Ould-Slimane, H., Talhi, C. & Cheriet, M. (2017). Attribute-based encryption for preserving smart home data privacy. *International conference on smart homes and health telematics*, pp. 185–197.

Cooper, V. N., Shahriar, H. & Haddad, H. M. (2014). A Survey of Android Malware Characterisitics and Mitigation Techniques. *Information Technology: New Generations (ITNG), 2014 11th International Conference on*, pp. 327–332.

Crussell, J., Gibler, C. & Chen, H. (2012). Attack of the clones: Detecting cloned applications on android markets. *European Symposium on Research in Computer Security*, pp. 37–54.

Dalianis, H. (2018). Evaluation Metrics and Evaluation. In *Clinical Text Mining: Secondary Use of Electronic Patient Records* (pp. 45–53). Cham: Springer International Publishing. doi: 10.1007/978-3-319-78503-5_6.

Davis, B. & Chen, H. (2013). RetroSkeleton: retrofitting android apps. *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, pp. 181–192.

Davis, B., Sanders, B., Khodaverdian, A. & Chen, H. (2012). I-arm-droid: A rewriting framework for in-app reference monitors for android applications. *Mobile Security Technologies*, 2012(2), 17.

Developers, A. (2015). UI/Application Exerciser Monkey. Retrieved from: Online[Access10/05/2017]url{https://developer.android.com/studio/test/monkey.html}.

Developers, A. (2022). Preparing for the Android Privacy Sandbox Beta. Retrieved from: Online[Access15/12/2022]url{https://android-developers.googleblog.com/2022/11/preparing-for-android-privacy-sandbox-beta.html}.

El-Harake, K., Falcone, Y., Jerad, W., Langet, M. & Mamlouk, M. (2014). Blocking Advertisements on Android Devices using Monitoring Techniques. *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, pp. 239–253.

El-Serngawy, M. & Talhi, C. (2015). CaptureMe: Attacking the User Credential in Mobile Banking Applications. *Trustcom/BigDataSE/ISPA, 2015 IEEE*, 1, 924–933.

Elarbi, M. (2018). *APSL : Langage de spécification des politiques de sécurité basées sur le contexte pour le contrôle des applications Android.* (Master's thesis, École de technologie supérieure).

Enck, W., Gilbert, P., Han, S., Tendulkar, V., Chun, B.-G., Cox, L. P., Jung, J., McDaniel, P. & Sheth, A. N. (2014). TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS)*, 32(2), 5.

Falcone, Y. & Currea, S. (2012). Weave droid: aspect-oriented programming on android devices: fully embedded or in the cloud. *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, pp. 350–353.

Feizollah, A., Anuar, N. B., Salleh, R., Suarez-Tangil, G. & Furnell, S. (2017). Androdialysis: Analysis of android intent effectiveness in malware detection. *computers & security*, 65, 121–134.

Feth, D. & Pretschner, A. (2012). Flexible data-driven security for android. *Software Security and Reliability (SERE), 2012 IEEE Sixth International Conference on*, pp. 41–50.

Georgakopoulos, D., Zaslavsky, A. & Perera, C. (2012). Sensing as a service and big data. *Proceedings of the International Conference on Advances in Cloud Computing (ACC'12)*.

Gibler, C., Crussell, J., Erickson, J. & Chen, H. (2012). AndroidLeaks: automatically detecting potential privacy leaks in android applications on a large scale. *International Conference on Trust and Trustworthy Computing*, pp. 291–307.

Goyal, V., Pandey, O., Sahai, A. & Waters, B. (2006). Attribute-based encryption for fine-grained access control of encrypted data. *Proceedings of the 13th ACM conference on Computer and communications security*, pp. 89–98.

Grace, M. & Sughasiny, M. (2022). Behaviour analysis of inter-app communication using a lightweight monitoring app for malware detection. *Expert Systems with Applications*, 210, 118404.

He, Y., Yang, X., Hu, B. & Wang, W. (2019). Dynamic privacy leakage analysis of Android third-party libraries. *Journal of Information Security and Applications*, 46, 259–270.

Hornyack, P., Han, S., Jung, J., Schechter, S. & Wetherall, D. (2011). These aren't the droids you're looking for: retrofitting android to protect data from imperious applications. *Proceedings of the 18th ACM conference on Computer and communications security*, pp. 639–652.

IDC. (2020). Smartphone OS Market Share. Retrieved from: http://www.idc.com/promo/smartphone-market-share/os.

Inshi, S., Chowdhury, R., Elarbi, M., Ould-Slimane, H. & Talhi, C. (2020). LCA-ABE: Lightweight context-aware encryption for android applications. *2020 International Symposium on Networks, Computers and Communications (ISNCC)*, pp. 1–6.

Inshi, S., Chowdhury, R., Ould-Slimane, H. & Talhi, C. (2022). Dynamic Context-Aware Security in a Tactical Network Using Attribute-Based Encryption. *MILCOM 2022 - 2022 IEEE Military Communications Conference (MILCOM)*, pp. 49-54. doi: 10.1109/MILCOM55135.2022.10017647.

Inshi, S., Chowdhury, R., Ould-Slimane, H. & Talhi, C. (2023a). Secure Adaptive Context-Aware ABE for Smart Environments. *IoT*, 4(2), 112–130. doi: 10.3390/iot4020007.

Inshi, S., Elarbi, M., Chowdhury, R., Ould-Slimane, H. & Talhi, C. (2023b). CAPEF: Context-Aware Policy Enforcement Framework for Android Applications. *Journal of Engineering Research and Sciences*. doi: 10.55708/js0201002.

Jancic, A. & Warren, M. J. (2004). PKI-Advantages and Obstacles. *AISM*, pp. 104–114.

Jeon, J., Micinski, K. K., Vaughan, J. A., Fogel, A., Reddy, N., Foster, J. S. & Millstein, T. (2012). Dr. Android and Mr. Hide: fine-grained permissions in android applications. *Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices*, pp. 3–14.

JesusFreke. (2013). Smali. Retrieved from: https://github.com/JesusFreke/smali.

Kavitha, D. & Ravikumar, S. (2021). IOT and context-aware learning-based optimal neural network model for real-time health monitoring. *Transactions on Emerging Telecommunications Technologies*, 32(1), e4132. doi: 10.1002/ett.4132.

Kim, J., Yoon, Y., Yi, K., Shin, J. & Center, S. (2012). ScanDal: Static analyzer for detecting privacy leaks in android applications. *MoST*, 12.

Kim, J.-C. & Chung, K. (2020). Neural-network based adaptive context prediction model for ambient intelligence. *Journal of Ambient Intelligence and Humanized Computing*, 11, 1451–1458. doi: 10.1007/s12652-018-0972-3.

Lab, S. (2014). http://siis.cse.psu.edu/ded/. Retrieved from: Online[Access02/05/2018]url{http://siis.cse.psu.edu/ded/}.

Lin, J., Liu, B., Sadeh, N. & Hong, J. I. (2014). Modeling users? mobile app privacy preferences: Restoring usability in a sea of permission settings. *Symposium On Usable Privacy and Security (SOUPS 2014)*, pp. 199–212.

Liu, X., Liu, J., Zhu, S., Wang, W. & Zhang, X. (2019). Privacy risk analysis and mitigation of analytics libraries in the android ecosystem. *IEEE Transactions on Mobile Computing*.

Lu, L., Li, Z., Wu, Z., Lee, W. & Jiang, G. (2012). Chex: statically vetting android apps for component hijacking vulnerabilities. *Proceedings of the 2012 ACM conference on Computer and communications security*, pp. 229–240.

Maheswari, S. & Gudla, U. (2017). Secure sharing of personal health records in Jelastic cloud by attribute based encryption. *2017 4th International Conference on Advanced Computing and Communication Systems (ICACCS)*, pp. 1–4.

Mann, C. & Starostin, A. (2012). A framework for static detection of privacy leaks in android applications. *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pp. 1457–1462.

Maring, J. (2018). Android Central. Retrieved from: https://www.androidcentral.com/google-removed-over-700000-malicious-apps-play-store-2017.

Martín, I. & Hernández, J. A. (2019). CloneSpot: Fast detection of Android repackages. *Future Generation Computer Systems*, 94, 740–748.

Martinelli, F., Mori, P. & Saracino, A. (2016). Enhancing android permission through usage control: a BYOD use-case. *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, pp. 2049–2056.

Michalakis, K. & Caridakis, G. (2022). Context awareness in cultural heritage applications: a survey. *ACM Journal on Computing and Cultural Heritage (JOCCH)*, 15(2), 1–31. doi: 10.1145/3480953.

Mila. (2011, July). Take a sample, leave a sample. Mobile malware mini-dump. Retrieved from: http://contagiodump.blogspot.ca/2011/03/take-sample-leave-sample-mobile-malware.html.

Mshali, H., Lemlouma, T. & Magoni, D. (2018). Adaptive monitoring system for e-health smart homes. Pervasive Mob. Comput. 43, 1–19.

Nauman, M., Khan, S. & Zhang, X. (2010). Apex: extending android permission model and enforcement with user-defined runtime constraints. *Proceedings of the 5th ACM symposium on information, computer and communications security*, pp. 328–332.

Nawrocki, P., Sniezynski, B., Kolodziej, J. & Szynkiewicz, P. (2020). Adaptive context-aware service optimization in mobile cloud computing accounting for security aspects. *Concurrency and Computation: Practice and Experience*.

OASIS. (2011). OASIS eXtensible Access Control Markup Language (XACML). Retrieved from: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml.

Oberko, P. S. K., Obeng, V.-H. K. S. & Xiong, H. (2021). A survey on multi-authority and decentralized attribute-based encryption. *Journal of Ambient Intelligence and Humanized Computing*, 1–19.

Ongtang, M., McLaughlin, S., Enck, W. & McDaniel, P. (2012). Semantically rich application-centric security in Android. *Security and Communication Networks*, 5(6), 658–673.

Onwuzurike, L. (2019). *Measuring and Mitigating Security and Privacy Issues on Android Applications*. (Ph.D. thesis, UCL (University College London)).

Pearce, P., Felt, A. P., Nunez, G. & Wagner, D. (2012). Addroid: Privilege separation for applications and advertisers in android. *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, pp. 71–72.

Perera, C., Zaslavsky, A., Christen, P. & Georgakopoulos, D. (2013). Context aware computing for the internet of things: A survey. *IEEE communications surveys & tutorials*, 16(1), 414–454.

Phung, P. H., Mohanty, A., Rachapalli, R. & Sridhar, M. (2017). Hybridguard: A principal-based permission and fine-grained policy enforcement framework for web-based mobile applications. *2017 IEEE Security and Privacy Workshops (SPW)*, pp. 147–156.

pxb1988. (2014, 10). SourceForge. Retrieved from: Online[Access03/05/2017]url{https://sourceforge.net/projects/dex2jar/}.

Raftery, A. E. (1985). A model for high-order Markov chains. *Journal of the Royal Statistical Society: Series B (Methodological)*, 47(3), 528–539.

Rasthofer, S., Arzt, S., Lovat, E. & Bodden, E. (2014). Droidforce: enforcing complex, data-centric, system-wide policies in android. *Availability, Reliability and Security (ARES), 2014 Ninth International Conference on*, pp. 40–49.

Rasthofer, S., Asrar, I., Huber, S. & Bodden, E. (2015). *An investigation of the android/badaccents malware which exploits a new android tapjacking attack*.

Rathore, I. (2022). Google gets rid of these 16 apps having millions of downloads. Retrieved from: https://dazeinfo.com/2022/10/25/google-removes-apps-that-have-affected-20-million-android-users-worldwide/.

Riganelli, O., Micucci, D. & Mariani, L. (2019). Controlling interactions with libraries in android apps through runtime enforcement. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 14(2), 1–29.

Saini, G. (2015). *Permission Based Android Malware Detection Using Supervised Learning Techniques*. (Ph.D. thesis, NATIONAL INSTITUTE OF TECHNOLOGY JALANDHAR).

Sandhu, R. S., Coyne, E. J., Feinstein, H. L. & Youman, C. E. (1996). Role-based access control models. *Computer*, 29(2), 38–47.

Sarker, I. H. & Salah, K. (2019). Appspred: predicting context-aware smartphone apps using random forest learning. *Internet of Things*, 8, 100106.

Sarker, I. H., Kayes, A. & Watters, P. (2019). Effectiveness analysis of machine learning classification models for predicting personalized context-aware smartphone usage. *Journal of Big Data*, 6(1), 1–28.

Sarker, I. H., Colman, A., Han, J., Khan, A. I., Abushark, Y. B. & Salah, K. (2020). Behavdt: a behavioral decision tree learning to build user-centric context-aware predictive model. *Mobile Networks and Applications*, 25(3), 1151–1161.

Schilit, B. N. & Theimer, M. M. (1994). Disseminating active map information to mobile hosts. *IEEE network*, 8(5), 22–32.

Selvan, S. & Mahinderjit Singh, M. (2022). Adaptive Contextual Risk-Based Model to Tackle Confidentiality-Based Attacks in Fog-IoT Paradigm. *Computers*, 11(2), 16. doi: 10.3390/computers11020016.

Sen, S., Aysan, A. I. & Clark, J. A. (2017). SAFEDroid: using structural features for detecting android malwares. *International Conference on Security and Privacy in Communication Systems*, pp. 255–270.

Shao, J., Lu, R. & Lin, X. (2015). Fine-grained data sharing in cloud computing for mobile devices. *2015 IEEE Conference on Computer Communications (INFOCOM)*, pp. 2677–2685.

Shekhar, S., Dietz, M. & Wallach, D. S. (2012). Adsplit: Separating smartphone advertising from applications. *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*, pp. 553–567.

Taha, M. B. & Chowdhury, R. (2020). GALB: Load Balancing Algorithm for CP-ABE Encryption Tasks in E-Health Environment. *2020 Fifth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)*, pp. 165–170.

Taha, M. B., Ould-Slimane, H. & Talhi, C. (2020). Smart offloading technique for CP-ABE encryption schemes in constrained devices. *SN Applied Sciences*, 2(2), 1–19.

Tan, Y.-L., Goi, B.-M., Komiya, R. & Tan, S.-Y. (2011). A study of attribute-based encryption for body sensor networks. *International Conference on Informatics Engineering and Information Science*, pp. 238–247.

Text, O. L. (2012). General Data Protection Regulation (GDPR). Retrieved from: Online[Access18/10/2022]url{https://gdpr-info.eu/}.

Thanigaivelan, N. K., Nigussie, E., Hakkala, A., Virtanen, S. & Isoaho, J. (2018). CoDRA: Context-based dynamically reconfigurable access control system for android. *Journal of Network and Computer Applications*, 101, 1–17.

Tumbleson. (2010). Apktool - A tool for reverse engineering 3rd party, closed, binary Android apps. Retrieved from: https://ibotpeaches.github.io/Apktool.

Ullah, A., Azeem, M., Ashraf, H., Alaboudi, A. A., Humayun, M. & Jhanjhi, N. (2021). Secure healthcare data aggregation and transmission in IoT—A survey. *IEEE Access*, 9, 16849–16865.

von Styp-Rekowsky, P., Gerling, S., Backes, M. & Hammer, C. (2013). Idea: callee-site rewriting of sealed system libraries. *International Symposium on Engineering Secure Software and Systems*, pp. 33–41.

Vronsky, J., Stevens, R. & Chen, H. (2017). *SurgeScan: Enforcing security policies on untrusted third-party Android libraries*. IEEE.

Wu, C., Zhou, Y., Patel, K., Liang, Z. & Jiang, X. (2014). AirBag: Boosting Smartphone Resistance to Malware Infection. *NDSS*.

Xie, J., Fu, X., Du, X., Luo, B. & Guizani, M. (2017). Autopatchdroid: A framework for patching inter-app vulnerabilities in android application. *2017 IEEE International Conference on Communications (ICC)*, pp. 1–6.

Xu, R., Saïdi, H. & Anderson, R. (2012). Aurasium: Practical policy enforcement for android applications. *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*, pp. 539–552.

Yahyaoui, H. & Almulla, M. (2010). Context-based specification of web service policies using wspl. *Digital Information Management (ICDIM), 2010 Fifth International Conference on*, pp. 496–501.

Yang, W., Li, J., Zhang, Y., Li, Y., Shu, J. & Gu, D. (2014). APKLancet: tumor payload diagnosis and purification for android applications. *Proceedings of the 9th ACM symposium on Information, computer and communications security*, pp. 483–494.

Yao, X., Chen, Z. & Tian, Y. (2015). A lightweight attribute-based encryption scheme for the Internet of Things. *Future Generation Computer Systems*, 49, 104–112.

Yu, F., Xuming, L., Xiao, G., Lin, L. & Jingzhao, L. (2020). A survey on key technologies of privacy leakage detection for Android platform. *Journal of Physics: Conference Series*, 1437(1), 012006.

Zacarias, I., Gaspary, L. P., Kohl, A., Fernandes, R. Q., Stocchero, J. M. & de Freitas, E. P. (2017). Combining software-defined and delay-tolerant approaches in last-mile tactical edge networking. *IEEE Communications Magazine*, 55(10), 22–29.

Zhang, M. & Yin, H. (2014). Efficient, context-aware privacy leakage confinement for android applications without firmware modding. *Proceedings of the 9th ACM symposium on Information, computer and communications security*, pp. 259–270.

Zhang, X., Ahlawat, A. & Du, W. (2013). AFrame: isolating advertisements from mobile applications in Android. *Proceedings of the 29th Annual Computer Security Applications Conference*, pp. 9–18.

Zhang, Y., Deng, R. H., Xu, S., Sun, J., Li, Q. & Zheng, D. (2020). Attribute-based encryption for cloud computing access control: A survey. *ACM Computing Surveys (CSUR)*, 53(4), 1–41.

Zheng, D. E. & Carter, W. A. (2015). *Leveraging the internet of things for a more efficient and effective military*. Rowman & Littlefield.

Zhou, W., Zhou, Y., Jiang, X. & Ning, P. (2012). Detecting repackaged smartphone applications in third-party android marketplaces. *Proceedings of the second ACM conference on Data and Application Security and Privacy*, pp. 317–326.

Zhou, W., Zhang, X. & Jiang, X. (2013). AppInk: watermarking android apps for repackaging deterrence. *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, pp. 1–12.

Zhou, Y. & Jiang, X. (2012). Dissecting android malware: Characterization and evolution. *2012 IEEE Symposium on Security and Privacy*, pp. 95–109.

Zhou, Y., Zhang, X., Jiang, X. & Freeh, V. W. (2011). Taming information-stealing smartphone applications (on android). *International conference on Trust and trustworthy computing*, pp. 93–107.

Zibetti, G. R., Wickboldt, J. A. & de Freitas, E. P. (2022). Context-aware environment monitoring to support LPWAN-based battlefield applications. *Computer Communications*, 189, 18–27.