UNIVERSITÀ DEGLI STUDI FIRENZE

UNIVERSITÀ DEGLI STUDI DI PERUGIA

INdAM
Istituto Nazionale
di Alta Matematica

Università di Firenze, Università di Perugia, INdAM consorziate nel CIAFM

DOTTORATO DI RICERCA
IN MATEMATICA, INFORMATICA, STATISTICA

CURRICULUM IN INFORMATICA
CICLO XXIX

Sede amministrativa Università degli Studi di Firenze
Coordinatore Prof. Graziano Gentili

Smartphones, Drones and IoT: Security and Privacy in Heterogeneous Smart Devices

Settore Scientifico Disciplinare INF/01

Dottorando:
Kanishka Ariyapala

Tutore
Prof. Mauro Conti
Prof. Cristina M. Pinotti

Coordinatore
Prof. Graziano Gentili

Anni 2013/2016

Abstract

Heterogeneous computing devices are surrounding us in our day-to-day life at an unprecedented rate and they are showing promising capabilities. For example, the drone *Loon Copter* is one such device providing an unrestricted mobility in air, surface and underwater. Similarly, smartphones and IoT are also enabling to sense our environment. Many of these devices are embedded with processing, sensing, software and communication capabilities, allowing many services to be built on top. In the near future these connected devices will be everywhere from smart cities, factories to our homes and even on our bodies. In order to reach the full potential of these emerging devices, a prominent requirement, “security by design", must be fulfilled to make the technology ready for mass adaptation. Following this direction, we focus on security and privacy issues of three heterogeneous devices: (i) Smartphone related security and privacy issues; (ii) Application of drones for secure localization; and (iii) IoT related security issues.

In the first part of this dissertation we look into security and privacy challenges in smartphones. Smartphones are taking a leading role in binding most of the heterogeneous computing devices and also getting cluttered with lot of personal data. We investigated three security related issues in smartphones: i) malware detection, ii) preserving anonymity in mobile cloud communications, and iii) analyzing the energy consumption of cryptographic protocols to improve user experiences. As for point i), in most attack scenarios an adversary takes local or remote control of a mobile device (by leveraging system vulnerabilities via malicious apps), and sends the collected information from the smartphone to a remote web server. This undermines the users security and privacy, and we propose a new approach for detecting malware by focusing on network communications. As for point ii), Smartphone applications are increasingly relying on cloud services such as online banking, instant messaging and file exchange. For an external observer, this communication side channel may reveal a lot of information. Strong adversaries like government agencies are also proposing these channels as a

means to monitor their surveillance targets. Similarly, if the mobile network providers and cloud service providers collude together, they can violate the privacy of the users. We propose an end-to-end anonymous communications protocol for delay-tolerant applications (similar to Whatsapp or Email), to protect user privacy and prove the security properties of the protocol under this strong attack model. Finally, as for point iii), we analyze the energy consumption of cryptographic protocols running on smartphones. The number of web services accessed over encrypted traffic is rapidly growing, especially via SSL/TLS. In our investigation, we focused on TLS and show how TLS session resume can greatly save energy by avoiding asymmetric cryptographic operations. We further propose Cloud aided TLS (CaT).

In the second part of this dissertation, we explore the possibility of using the emerging drone technology to solve the secure location verification problem. Many innovations are emerging using drones such as last mile delivery and emergency response. Many dependable distributed systems are vulnerable to node displacement attacks. For example, a hostile actor physically moving few sensors in a pollution monitoring system can easily disrupt the monitoring. This displacement attack is simple, but difficult to detect. Current solutions require several fixed anchor nodes with trusted positions. We propose VerifierBee, which replace all the fixed anchors with a single drone that flies through a sequence of waypoints. VerifierBee, finds a good approximation of the shortest path, and at the same time it respects a set of requirements about drone controllability, localization precision and communication range.

The third part of this dissertation focuses on IoT related security issues. In many scenarios, IoT systems comprise of widely deployed sensors and actuators with connectivity. Many of these sensors are battery operated, with low processing power and left unattended after deployment. Therefore, lightweight low-power security protocols are needed. In this part of the dissertation we propose a framework to detect IoT sensor node actions by observing the encrypted communication traffic. In particular, IETF standardized DTLS encrypted traffic. There are many recent incidents about DDoS attacks using compromised IoT devices and our work steps in this direction to detect any compromised nodes.

Acknowledgments

First I would like to thank my supervisors Prof. Mauro Conti and Prof. Cristina M. Pinotti for their supervision of my PhD during the last three years. Also, I would like to thank Prof. N. Asokan of Alto Univeristy and University of Helsink, for the valuable insights given from time to time during the course of this research. Further, I would like to thank Prof. Wee Keo Ng and Prof. Newton Fernando of Nanyan Technological University, Singapore, for the guidance and support given to me during my visiting period at NTU, Singapore.

I am grateful for the tripartite consortium: University of Florence, University of Perugia and INDAM who supported me with the scholarship during my PhD studies. Also, I would like to thank Prof. Graziano Gentili the PhD program coordinator.

I would also like to thank the University of Padova, for hosting me, where I spent most of my time during my studies. Also, I thank the lab mates for the discussions we had, who's suggestions greatly helped me to improve this work. Also, I thank my friends who were always by my side to share happiness and adversity. Finally, I am immensely gratitude to my parents and my family who have always been a great strength to me in all moments of my life.

Contents

List of Figures

List of Tables

Acronyms

6LoWPAN IPv6 over Low power Wireless Personal Area Networks

APK Android application Package

BYOD Bring Your Own Device

CaT Cloud aided TLS

CCTV Closed Circuit Television

CoAP Constrained Application Protocol

CPU Central Processing Unit

DDoS Distributed Denial of Service

DNS Domain Name Service

DTLS Datagram Transport Layer Security

DVR Digital Video Recorder

GPS Global Positioning System

HMM Hidden Markov Model

HTTPS Hyper Text Transfer Protocol Secure

IETF Internet Engineering Task Force

IoT Internet of Things

IPv6 Internet Protocol version 6

IR-UWB Impulse Radio Ultra Wide Band

LAN Local Area Network

MAC Message Authentication Code

OS Operating System

PC Personal Computer

PDA Personal Digital Assist

PHY Physical Layer

RAM Random Access Memory

REST Representational State Transfer

SSH Secure Shell

SSL Secure Socket Layer

SVM Support Vector Machine

TCP Transmission Control Protocol

TLS Transport Layer Security

TLVP Traveller Location Verifier Problem

TSP Travelling Salesman Problem

UAV Unmanned Aerial Vehicle

UDGM Unit Disk Graph Model

UDP User Datagram Protocol

VoIP Voice over Internet Protocol

VPN Virtual Private Network

WSN Wireless Sensor Network

Chapter 1

Introduction

Heterogeneous computing devices are emerging in our surrounding at an unprecedented rate and are showing promising capabilities in the next generation of computing and sensing [113]. For example, the drone ‘Loon Copter’ is providing an unprecedented mobility in air, surface and underwater [57]. Furthermore, new connected devices like Kinect [33], Google glass [91] are also enabling new opportunities for innovation and knowledge creation. Many of these devices are mainly enabling technologies and potential apps are built around these devices. Smartphone is another such device which revolutionized our way of communication. At present it is more than a mere communication device, and it also acts as a computing device as well. Smartphone is embedded with rich sensing, computing and many connecting interfaces in one single hardware and software platform. Given the ubiquitous nature of the smartphone, along with the support for users day-to-day digital activities, it has become an epicenter for personal data and provides access to privacy sensitive information such as user locations and activities, through the rich inbuilt sensors. Internet of Things (IoT), is another emerging paradigm taking the sensing and actuation to the next level by deploying billions and billions of embedded devices around us. Most of the time these devices are embedded with processing, sensing, software and a means of connectivity, enabling to communicate with other devices and enabling remote access from anywhere. For example, the Nest thermostat enables to control the house temperature while you are away (through the smartphone app) and enables to save energy and be more efficient [15]. However, the availability of these devices in huge numbers is opening new attack surfaces for the cyber criminals. Following, we present the research motivation and contributions.

1.1 Research Motivation and Contribution

In the era of IoT we are constantly being surrounded by a plethora of heterogeneous devices. The falling prices and shrinking size of computer processors and sensors, coupled with connectivity, is enabling to create a global interconnection of devices addressable from anywhere in the world. Advent of technologies like IPv6 is just fueling the process. According to Cisco, IoT alone is foreseen to connect over 50 billion low-power devices to the Internet by end of 2020 [134]. Furthermore, it is expected by 2020 to have around 7 millions of drones [69] and around 6.1 billion smartphones [68] connected to our technological echo-system. Smartphones play an essential role in connecting a variety of heterogeneous devices including IoT and drones. Recent progress in IoT technologies, enable remotely controllable lights, thermostats and cameras, while smartphone apps are providing the binding interface between the device and the user [126, 122, 190]. Similarly, many drones on the market use smartphone as a cost-effective solution to provide the drone controlling interface [38, 167, 60, 117, 94]. Smartphones offer a wide range of connecting interfaces, such as NFC, Bluetooth, WiFi Direct and cellular, making it ideal to interact with other devices and sensors. This capability enables the smartphone to be the perfect qualifier to be the eyes and ears for a given heterogeneous device ecosystem [66]. Furthermore, these emerging smart devices are recognized by the industry leaders and media as the next wave of innovation, disrupting the way we organize our daily life's [115]. These connected smart devices will provide different services to help us be more productive with our daily tasks.

However, given the great pace at which connected smart devices are growing, we are starting to experience also the dark side of these connected devices. These heterogeneous devices are deployed in homes, factories, hospitals, entire cities and accompanied by billions of people in their daily lives. This is providing a widened attack surface for attackers offering a large number of entry points to build botnets of tens of thousands of compromised devices. These hacked devices can be used as weapons in cyber attacks to bring down web services or intimidate the oppositions [27, 64]. Some recent attacks using compromised connected cameras and DVRs, were able to direct extraordinary high volumes of traffic (e.g., 700 Gbps) to neutralize their target [67]. Similarly these attacks can be directed towards smartphone users too for monitory gains. Current smartphone malware are capable of performing data theft, surveillance and financial fraud compromising the security and privacy rights of the mobile users, while showing new threat tactics such as ransomware [16]. Therefore, the security aspect of these smart devices is a growing concern as the devices scale into billions while also being connected to the Internet.

The research work presented in this dissertation will focus on the security aspects of three heterogeneous smart devices. In particular, we analyzed specific emerging security requirements in smartphones and IoT, along with the application of drones for securing IoT sensor network deployments. We present our research organized according to the three heterogeneous devices:

- Smartphones: to harden against *harmful malware* and overcome *specific privacy issues* users are facing while making use of mobile cloud services.
- Drones: to solve the *secure localization problem* of distributed sensor network deployments in dependable systems against displacement attacks, using a mobile drone.
- IoT sensor nodes: to *detect the IoT node activities over encrypted traffic* in IoT sensor networks.

In the following section we briefly introduce each of the above issues and summarize our contributions.

1.1.1 Smartphone Security and Privacy Issues

Smartphones have changed the way we do our communications and interact with technology. Internet, cloud access, online banking, instant messaging, third-party device controlling (e.g., Nest thermostat) and file exchange through cloud are just a handful of the myriad of smartphone services that we make use of every day. Smartphone is an advanced communication device with considerable computing and sensing capabilities. Furthermore, the different communication interfaces (e.g., Cellular, WiFi, Bluetooth, WiFi direct, NFC) are providing connectivity to further enhance the capabilities of the smartphone as a platform. Generally, smartphones come with a pre-installed operating system (OS) and applications by the vendor. Also, the device manufactures encourage third-party application developers by providing APIs for their OS platforms and managing application markets for hosting these applications (e.g., Google Play, Apple App Store). However, due to restrictions in different countries (e.g., China, where Google Play store is banned) third-party app stores are still attracting many users. These smartphone apps, coupled with emerging paradigms like mobile cloud computing is offering new services to the end-user and the smartphone app market is expected to grow from current $50 billion to $101 billion by 2020 [71].

With the rising popularity of smartphones we are also witnessing a rising threat in malware targeting smartphone platforms [150, 23, 12]. At present, majority of the users are inseparable from their smartphones and they perform a lot of day-to-day operations on these devices, turning the smartphone to a vault containing personal and privacy sensitive data. Figure 1.1, presents some of the usecases smartphones are used.

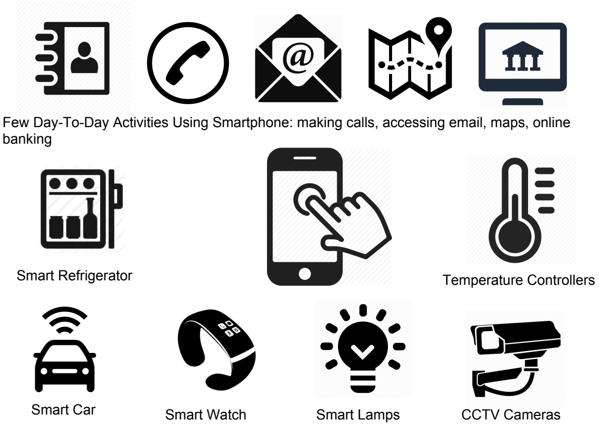

Few Day-To-Day Activities Using Smartphone: making calls, accessing email, maps, online banking
Smart Refrigerator
Temperature Controllers
Smart Car
Smart Watch
Smart Lamps
CCTV Cameras

Figure 1.1: Some smartphone usecase connecting devices and providing a controlling interface.

Smartphones store contacts, emails, messages, photos, credit card and banking information. Also, privacy sensitive information such as user locations and activities. Cybercriminals are always interested in these user personal information [132, 61]. They are looking at ways to compromise these devices by leveraging system vulnerabilities via malicious apps to gain remote or local access to the device and sends the collected user data to some remote web service. According to the anti-virus vendors like Sophos, smartphone platforms are a target of many emerging malware families, such as: surveillance, data theft, botnet activity, impersonation, financial fraud and intrusive advertising [92]. Apart from the malware threat, the application traffic can compromise the user privacy as well. For example, government agencies were known, to be able to retrieve, sensitive information from smartphones, such as: location, app preference, unique device identifiers, frequency of app usage, version of the OS by looking into the advertising and analytic traffic generated by the ad libraries embedded in most of the free apps [70, 53]. Therefore, preserving end user privacy in such mobile-cloud communications is another growing concern related to smartphones. In this dissertation our investigations were motivated towards malware detection and some specific privacy related issues.

1.1.2 Application of Drones for Secure Localization

Another interesting technology, which is emerging rapidly is drones. The commercialization of cheap unmanned aerial vehicles (UAVs) or commonly known as drones is starting to change the way sensor network system designers, think of data collection. For example, decades ago military relied on ground sensor nodes for information gathering about the enemy movement in a battle field, however they are now being replaced by mobile drones. Mobile drones can be seen as small embedded computers that move almost unconstrained while carrying rich sensory payloads, such as cameras and microphones, bring sensing and actuation to where no other technology can reach. They are effectively replacing the connected sensors at rest with one device being able to be: i) Deployed to different locations, ii) Capable of carrying flexible payloads, iii) Re-programmable in missions and being able to measure just about anything, anywhere. Figure 1.2 presents some application scenarios.

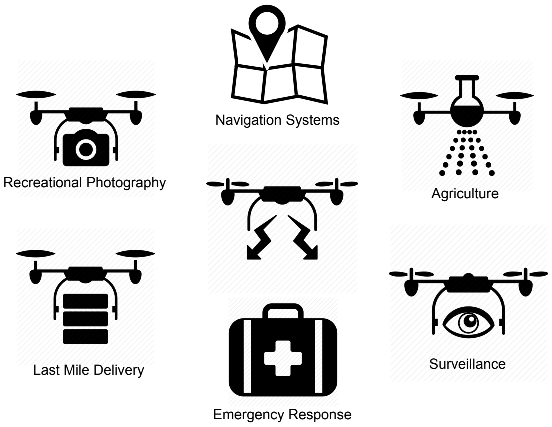


Figure 1.2: Some drone application usecase scenarios.

Autonomous drones are emerging as a powerful new breed of mobile sensing system [130] and many of them can be controlled by setting waypoints or by manually steering using a graphical interface through a tablet or a smartphone. As new designs emerge, drones progressively achieve higher speeds, carry larger payloads, and cover larger distances on batteries [129]. They are increasingly being used to monitor a farmers crops [181], perform surveillance in disaster relief [157], or monitoring underwater telecommunication systems [158] more practically and/or more cost-effectively than stationary sensors. Compared to other kinds of mobile sensing, such as opportunistic sensing that piggybacks on the mobility of smartphones or vehicles, drones offer direct control over where to sample in the environment and the application can explicitly instruct them where to move. At the moment there are many innovations around drones to help us in our day-to-day life apart from sensing and data collection. For example, there is a mail delivery system introduced using drones in Singapore [7] and the defibrillator drone introduced by TU Delft in Netherlands [65]. Also, Amazon and Google are testing drone based delivery systems [131].

In this dissertation we apply drones to another novel application: secure location verification of distributed dependable systems. Dependability of many distributed systems relies on the integrity of the component devices. As an example, let us consider a sensor network deployed for pollution monitoring. The sensors could measure the density of dioxin in the air at different positions and report it to a centralized gateway, which eventually decides whether it should raise an alarm. An adversary willing to mask a pollution event could simply move some sensors in different positions, in a way such that it will avoid the detection. This attack is simple to carry out, yet difficult to detect. To mitigate such attacks and to enhance the reliability of distributed sensor networks against node displacement attack, we propose a drone based verifiable multilateration method. Drones setup a platform for creating innovative services which are flexible, cost-effective, reusable, environmental friendly (being emission free) and is foreseen to be an effective enabler in the future cyber physical society [178]. However, flying time is a major limitation of the current drone technology. Therefore, the energy cost of the drone must be considered in order to increase the drone operation time and a key optimization relies on determining the best drone route.

1.1.3 IoT Security Issues

IoT is becoming quite commonplace in the environments where we live, work and play. Cisco and Ericsson in their individual assessments have estimated around 50 billion IoT connected devices in the world by 2020 [134, 90]. These devices help us be more productive and provide for our safety. For example they are used in the environment to monitor earthquakes, to detect changes in plains, forests and measure CO_2 emission levels in our atmosphere. They monitor and facilitate vehicle traffic updates on highways and in cities to provide better commuter experience. They provide passive security in airports, shopping malls, garages and other facilities. They help us monitor, to move and promote merchandise in supermarkets and in warehouses. They monitor status, conditions and materials in production processes in our factories. Also, they are used to turn on heating and cooling in our houses at the right moment and enable us to be more energy efficient. These interconnected sensor nodes collect information regarding our physcial environment such as temperature, sound, acceleration, vibration, pressure, motion, video or particles in the air and help build useful services. IoT is enabling a new era of cyber physical society. Key characteristic of IoT applications are typically composed of: i) A sensor at rest, i.e., on a smart city a sensor on a highway or a bridge gathers input like weather conditions, seismic activity, ii) A connection is established between the sensor and a back-end data collection infrastructure, iii) The back-end data collection infrastructure is commonly deployed in the cloud and infer knowledge from the data. Figure 1.3 presents some use case scenarios where IoT sensor networks are used to sense the environment around us.

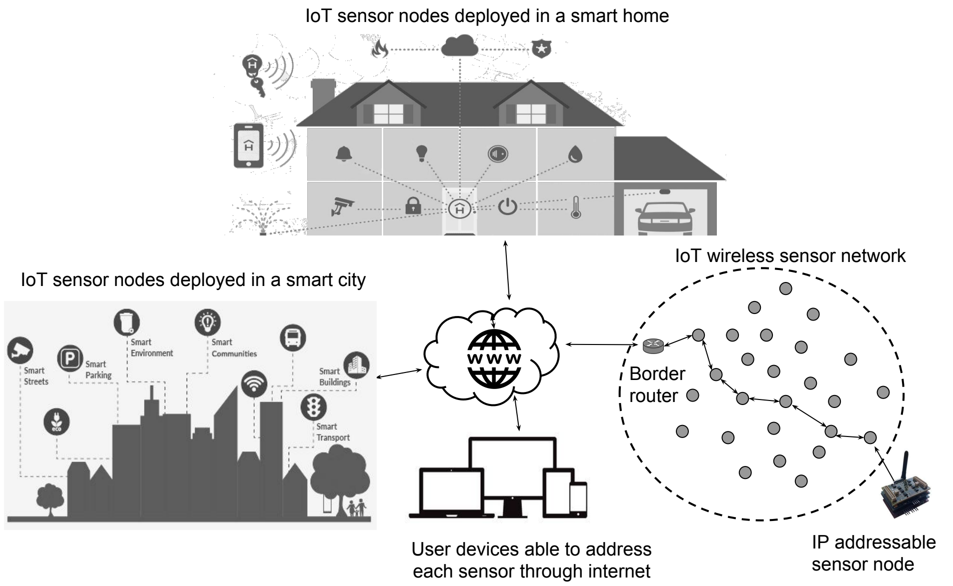

IoT sensor nodes deployed in a smart home
IoT sensor nodes deployed in a smart city
Smart Streets
Smart Parking
Smart Environment
Smart Communities
Smart Buildings
Smart Transport
eco
WWW
IoT wireless sensor network
Border router
User devices able to address each sensor through internet
IP addressable sensor node

Figure 1.3: Some IoT wireless sensor network application usecase scenarios.

However, these sensors are constrained in nature and most of the time these devices are equipped with non-rechargeable batteries and are left unattended after deployment. Therefore, IoT is providing a wider attack surface for the attackers [113]. Since, IoT is going to play a major role in our lives in the near future, securing these networks is important.

1.1.4 Contributions

Finally, the issues addressed in this dissertation can be summarized under three heterogeneous devices considered. The inter-dependency of these three devices can also be summarized as presented in Figure 1.4. Especially, smartphones play an essential role in connecting a variety of heterogeneous devices including IoT and drones. Among the considered three heterogeneous devices, IoT is expected to contribute most number of connected devices, around 50 billion by 2020 [134]. Similarly, it is expected around 7 million drones [69] and around 6.1 billion smartphones [68] in use by 2020. The connected nature of these devices are opening the network infrastructure for wider attack entry points. We discuss possible vulnerabilities and present counter measures to harden this technological ecosystem in the following chapters of this dissertation. In the following we summarize the main contributions:

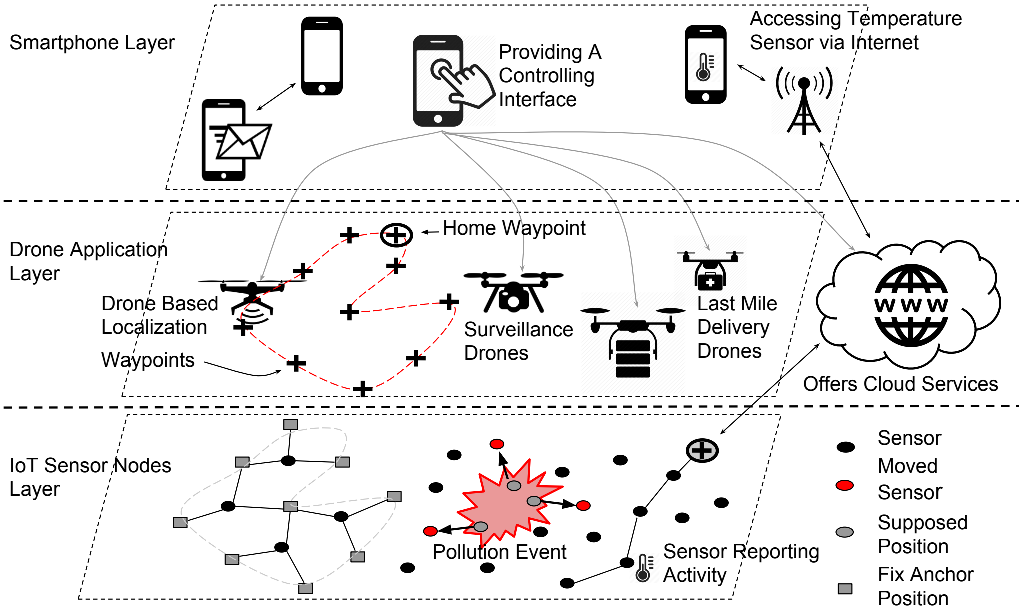


Figure 1.4: The system architecture, where IoT, drones and smartphones interact with each other.

Smartphone Security and Privacy Issues: In this part of the dissertation we present our work addressing three pressing issues in smartphones: i) Hardening smartphones against malware, ii) Preserving anonymity in adversarial mobile cloud communications, and iii) Analyzing the energy consumption of cryptographic protocols running on smartphones. Over the recent years with the popularity of smartphone usage, we are witnessing a growth in malware targeting smartphones and in particular the Android platform.

In most attack scenarios the adversary takes local or remote control of the mobile (by leveraging system vulnerabilities via malicious apps) and send the collected information to some remote web server. Leveraging the above network behavior, we propose a new approach for detecting malware solely by focusing on Android network communications. In particular, we used advanced machine learning techniques for identifying any anomalous malicious traffic. Our approach will be explained in detail in Chapter 2. Next we investigates a specific privacy issue around mobile cloud computing. Many services are appearing around the processing capability of clouds and the ubiquitous accessibility for these services with smartphones. However, the very enablers of these services, mobile Internet providers and cloud platforms that host them pose several threats to the anonymity of the users communications. Therefore, we considered the problem of providing end-to-end anonymous communications and file exchange under the cooperative privacy threat of involved parties including network operators and cloud providers, which could actively tamper with the communications. We present our protocol and the experimental analysis of it in Chapter 3. Many mobile apps rely on security protocols like Transport Layer Security (TLS) for securing the web communications of smartphones. However, cryptographic operations such as public key crypto is expensive on the CPU and hence on the energy consumption. In particular we designed a possible solution (named CaT-Cloud aided TLS) for offloading the TLS asymmetric key exchange. We conducted an extensive experimental analysis to evaluate whether offloading the asymmetric key exchange of the TLS protocol to a cloud instance could bring some benefit from the mobile device point of view. We present our protocol and the experimental analysis in Chapter 4.

Application of drones for sensor network security: In the second part, we explore the possibility of using the emerging drone technology to solve the secure location verification problem. Drones, or Unmanned Aerial Vehicles (UAV), are air-crafts without human pilots. They can enjoy different levels of autonomy [77]: ranging from being remotely piloted to being completely autonomous in movements and decisions. In practice, our idea is to replace many fixed anchors with a single mobile drone. The drone follows a path that passes though a series of waypoints. At each waypoint, the drone “acts like” an anchor by executing a distance bounding protocol with a node. At the end of the path, each node has been measured from three different waypoints, and the position can be securely computed by means of verifiable multilateration. We thus completely eliminate the need for many expensive fixed anchors. The problem is how to determine a convenient path for the drone. We cannot use existing path planning algorithms, because a valid path for verifiable multilateration must respect additional geometric constraints. In particular, the triangle formed by the waypoints must contain the node, otherwise the computed position will not be secure. Furthermore, other specific issues must be addressed, like the imprecision on the control of the drone movements. We explain in detail our approach in Chapter 5.

IoT Security Issues: In the first part of the dissertation we investigated the ability to detect IoT node activities over encrypted channels. Our work steps in the direction of identifying any compromised nodes in a network, where a lot of critical systems depend on. In particular, the industry and research community is aware about the lack of security in these low power devices and they are proposing new security protocols for sensor nodes [138]. Many initiatives are on the way and IETF accepted CoAP over DTLS is emerging as the *de-facto* standard for securing the communications for the low power devices [155, 142, 112, 80]. As a result the future IoT devices will communicate with each other over encrypted channels. In our work we have focused on IoT activity detection over encrypted traffic, using state of the art machine learning techniques and we will elaborate further in the Chapter 6.

1.2 List of Publications

The research presented in this dissertation was developed during the PhD program. The complete list of accepted peer-reviewed workshop, conference and journal publications are presented in the chronological order.

Journal Papers

J1 Claudio Ardagna, Kanishka Ariyapala, Mauro Conti, Cristina M. Pinotti, Julinda Stefa. Anonymous End-to-End Communications in Adversarial Mobile Clouds Pervasive and Mobile Computing. In (Elsevier) Pervasive and Mobile Computing, in press, 2016.

Conference Papers

C1 Kanishka Ariyapala, Mauro Conti, Chamath Keppitiyagama. ContextOS: a Context Aware Operating System for Mobile Devices. In Proceedings of the IEEE International Conference on Cyber, Physical and Social Computing (IEEE CPSCom 2013), pages. 976-984, Beijing, China, August 20-23, 2013.

C2 Pericle Perazzo, Kanishka Ariyapala, Mauro Conti, Gianluca Dini. The Verifier Bee: a Path Planner for Drone-Based Secure Location Verification. In Proceedings of the IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (IEEE WoWMoM 2015), Boston, MA, USA, June 14-17, 2015.

C3 Kanishka Ariyapala, Mauro Conti, Cristina M. Pinotti. CaT: Evaluating Cloud-aided TLS for Smartphone Energy Efficiency. In Proceedings of the 1st IEEE Workshop on Security and Privacy in Cybermatics (IEEE CNS 2015 workshop: SPiCy 2015), Florence, Italy, September 30, 2015.

C4 Kanishka Ariyapala, Hoang Giang Do, Huynh Ngoc Anh, Wee Keong Ng, Mauro Conti. Host and Network Based Intrusion Detection For Android Smartphones. In Proceedings of the 2016 Workshop on Security, Trust, Privacy and Analytics (IEEE AINA 2016 workshop: STPA 2016), Crans-Montana, Switzerland, March 23-25, 2016.

Part I

Security and Privacy Issues in Smartphones

Chapter 2

Smartphone Malware Detection Through Dynamic Analysis

Recently, we are witnessing a growing adaptation of mobile devices over traditional computing devices like desktop or laptop computers. The presence of increasing computing power granted by hardware, networking, sensing, and user-friendliness is empowering the mobile device usage. Out of the popular mobile device platforms, (e.g., iOS, Android and Windows) Android based devices are dominating the smartphone market share with around 80% of the market presence by the end of 2015[1]. The ease of program development, platform openness, and the user-friendliness have made the Android platform pleasant and ingenious. Millions of Android applications (apps) are distributed through the Google play store and other third-party application stores.

The social expectations of the 21st century to be always connected is exposing the smartphone users to possible new attack vectors. For example, the roaming nature of the mobile device is enabling the device to connect with different networks, which can make them vulnerable to additional attacks [146]. Smartphones store a lot of personal data (e.g., contacts, emails, web history, photos, and credit card information) on the devices and these private information have a lot of monetary value. The user popularity and the new hardware capabilities, along with the user private information stored on the smartphones are drawing the attention of cybercriminals. Around 99% of mobile malware is targeted towards Android platform and the malware

[1] http://www.forbes.com/sites/dougolenick/2015/05/27/apple-ios-and-google-android-smartphone-market-share-flattening-idc/2/#4e0ea8b7194f

authors are coming up with new sophisticated methods to avoid the malware detection. Therefore, there is an urgent need to develop methods to mitigate these threats.

According to anti-virus vendors like Sophos [92], the malware can be categorized as follows: surveillance, data theft, botnet activity, impersonation, financial fraud, and intrusive advertising. Though malware can be categorized into different groups, the aim of most malware authors is to have some degree of control over a victim's smartphone. Therefore, many malware apps request network connectivity permission [159], [37] to connect with the Internet. This is justifiable as many malware apps need to connect with their Command and Control (C&C) center, either to receive a malicious payload or to send user data from a device. According to the study in [163], it is possible to model the apps network behavior patterns and to identify any noticeable anomalous deviations for the majority of the cases.

Malware detection has attracted much attention from the research community and a considerable number of methods have been proposed to protect smartphone users from malware. Signature-based methods, which rely on the previously identified malware cannot detect well the new malware, meanwhile, anomaly based methods show promising performance. Anomaly based methods take their advantages from the use of machine learning techniques and the dynamic analysis. They can be divided into host-based feature and network-based feature methods. Host-based feature methods use OS related information to extract features. In [162], besides using different classifiers, Shabtai et al. employ Chi-square, Fisher score for feature selection, while methods described in [51], [29], [82] exploit system calls to build detection models. Those methods have the same drawbacks of using few malware sources and self-written malware. Unlike host-based methods, network-based methods focus on network related features like IP addresses, service ports, protocol usage, etc. Methods in [163], [9], [133] utilize network behaviours. Method described in [163] shows good result, but the benchmarking contains only ten self-written malware and five real malware. In [133], they extract 11 features, meanwhile, filter out DNS that can be important to identify malware. Another limitation of the previous work is performing the malware detection on the device or using third party cloud service. Performing the computation on the device effects the usability of the device (i.e., battery and responsiveness), while using a third party cloud service for overcoming the resource limitations may cause privacy issues, which shuns away the users. We identify these limitations and propose a use case for securely deploying our solution.

According to [192], [104] network behaviours of applications over time can be characterized as a discrete stochastic processes. Therefore, Markov chain-based methods are appropriate choice to construct malware detection models. Moreover, the sequence of states in a stochastic process can be considered as a state string. Thus, string kernel methods are potential since they take advantage of kernel-based methods that have been widely applied. In this dissertation, we propose two novel methods for malware detection. In our methods, we collect and analyze netflows that summarize statistics about network connections of each application. Besides, we employ a high number of recent malware belonging to diverse malware families to train and test the constructed models. For classifiers, we adopt Markov chain-based methods and string kernel-based methods. Therefore, the proposed methods can overcome the limitation of many existing methodologies.

Our approach can be used in several practical ways to enhance the security and privacy of the smartphone users. In the following we report some possible scenarios:

- Frequently, companies employ WiFi in the work premises and a VPN server to access their resources from outside. Our system can be easily deployed as a second line of defense to monitor the network traffic centrally. A company can use network based anomaly detection to secure the smartphones/tablets given to their employees, while being relaxed on the apps installed without being worried about enforcing policies on the application usage [43], [191]. Our method uses netflows which extracts summarized statistics of the netflows between apps and web services. Hence, our method is non-intrusive unlike a middle-box [165], which intercept the data packets and inspect the contents.

- Educational institutes, which have embraced BYOD paradigm [95] is encouraging personal devices to connect into secure networks which normally is behind the firewalls. However, this can compromise these networks due to malware apps running on these devices (e.g., Not-Compatible malware [119]). In this case, our proposed method can be applied for monitoring any anomalies in the network traffic in a non-intrusive manner and help secure the network.

- Some smartphone users are aware of the security threats prevailing for the smartphones. Our proposed method can be adopted by simply setting up a personal server and routing the traffic through it. This enables the user not only to detect any anomalies in the network but also to gain insights into the network patterns of the installed apps on her mobile device.

The contributions of our dissertation for Android malware detection are three folds:

- *Novel detection methods.* We introduce two novel methods, using dynamic analysis and efficient machine learning techniques for Android malware detection. To the best of our knowledge, this is the first attempt to apply Markov chain and string kernel methods for Android malware detection over network features.
- *Generalized network models.* We formally model our proposed methods, giving sound mathematical explanations and run a thorough set of experiments. The proposed models use network traffic samples. In order to build a generalized learning model, we collected the network traces of a large number of malware and benign application samples, belonging to different families on a real Android device.
- *Automation Framework.* We implement an automation framework using python and shell scripting to automate many of the tasks, such as installing, executing, uninstalling apps and collecting network traces and app related information from the experimental setup.

Organization The rest of the chapter is organized as follows: Section 2.1 explores the important related work in the field, especially focusing network-based anomaly detection for mobile devices. Section 2.2 explains the terminology and the background knowledge used in our work. Section 2.3 introduces our proposed approach for detecting Android malware using Markov chain model and string kernel model. Section 2.4 explains the experimental setup and the evaluation criteria, followed up by the results and discussion in Section 2.5. Finally, Section 6.6 concludes the findings and the highlights for future work.

2.1 Related Work

Recently, there has been a dramatic rise in malware targeting smartphones. This poses severe threats to the smartphones security systems. Therefore, many malware detection methods have been proposed to detect malware. Malware detection models can be categorized into two groups: signature and anomaly-based models. Signature based methods rely on previously identified malware and analyzing the malware files to create a signature. Many research work have been proposed in this area, for example, DERBIN [17] gathers features such as permissions, API calls, network addresses in the code and other information from the manifest file. DroidAPIMinor [6] is a similar work using package and API level information from the apk files. DroidMOSS [196] uses fuzzy hashing of the apk files for identifying repackaged apps mainly using author information retrieved from the apk files. Furthermore, RiskRanker [77], Dendroid [170] and Apposcopy [62] are deploying methods for analyzing the apk files and generating signatures from the features extracted from the apk files. The main drawback of the above static analysis based signatures is the constant need for updating the signature databases. Furthermore, dynamic code downloads, obfuscation methods and repackaging make it difficult to detect unseen variants of malware [170] and detection is getting tougher due to the rapidly changing nature of malware. Subsequently, signature based methods often fail to detect new malware [197].

Anomaly based detection systems are gaining more popularity from the research community and showing promising results. These systems deploy machine learning techniques to learn the behaviours of the malware to build detection models. Thus, they are able to detect new malware which may use obfuscated techniques or where they deliver the malware payload separately. Most anomaly-based methods use dynamic analysis, where the data collected by monitoring the behavior of applications while running. This is because most malware applications need to connect to a central C&C server to receive commands or to exfiltrate private data [146]. Anomalies can be detected based on either host-based or network-based features. In the coming paragraphs, we give an overview of the popular anomaly based detection methods for both host-based and network-based approaches.

Host-based feature methods exploit OS related information such as system calls, process IDs, CPU usage, memory consumption and battery usage. Shabtai et al. propose the Andromaly method in [162]. In this method, six classifiers for malware classification are employed. They are Decision Tree (DTJ48), Naive Biased (NB), Bayesian Networks (BN), K-Means, Histogram and Logistic regression based methods. Besides, they also use Chi-square and Fisher score for feature selection. The classifications are locally done on the device. Even though the accuracy rates are high especially for DTJ48, to evaluate, they have used self-written malware. A multilevel anomaly detector for Android mobile is introduced in [51] by Dini et al. This method works by monitoring system calls of the applications and using K-Nearest Neighbor (KNN) algorithm to classify software. The anomalies are detected locally on the devices and the authors test the system with 50 goodware and 10 malware samples. Crowdroid [29] is another method, that monitors the system calls by using a host-based monitoring application. After preprocessing, the data are sent to a central server where they are analyzed using the K-Means algorithm. For the evaluation, they have used self-written malware too, which makes the usability of the system questionable. Enack et al. presents PREC [82], an on device, malware detection system, which monitors the system calls and uses self organizing maps to detect Root exploits. They have tested their prototype with 140 benign apps and with only 10 malware apps. RobotDroid [195], introduced by Zhao et al. uses SVM classifier to detect unknown malware. They mainly focused on privacy information leakage and hidden payment systems. Their system was assessed with only three types of malware, namely Gemini, DroidDream and Plankton. Common drawbacks of the above systems are either the use of too few real malware samples or self-written malware, which may compromise the accuracy of these systems. Furthermore, the above-discussed methods use different detection techniques compared to our work, which only uses network-based features.

Network-based anomaly detection methods utilize network-based features such as IP addresses, service ports, protocol usage, traffic volume etc. STREAM presented by Amos et al. in [9], uses various features and classifiers. They have used different features such as network, memory, CPU usage, permissions, and binder interactions. However, network features are not dominant in their framework, when creating detection models. Besides that, not using real devices for creating malware profiles is another limitation. Shabati et al. in their work [163], introduce a network behaviour based malware detection framework. They create the network profiles using average values of data sent and received over the network interfaces. Moreover, they utilize cross-feature analysis approach for building their detection models which run on the mobile device, for which they report a resource overhead. For benchmarking the framework, ten self-written malware, and five real malware were used. It is a too small data set and only focuses on self-updating malware. Narudin et al. [133] presents a network behaviour based detection method for Android malware detection. In order to create the network behaviour profiles, they let the malware run once for either 10, 20 or 30 mins. They extract 11 features (i.e., src-ip, dst-ip, etc.) from pcap data and store them in a csv file. Further, they filter out Domain Name System (DNS) packets from their network capture. However, DNS traffic may give important indications of any malicious behavior [37].

Network behaviour of an application over time, such as establishing a connection, listening, exchanging data, idling or terminating connections, can be modelled as a discrete stochastic process where the behaviour can be considered as states. Therefore, using Markov Chains for constructing network-based methods is a logical choice. Markov chains have been applied in host-based methods in Android [36], [189] and network-based malware detection methods in local area networks (LANs) [97], [32], [104]. To the best of our knowledge, there is no network-based detection methods for Android using Markov chains. Besides, the sequence of network states can also be presented as a string [74]. Therefore, string kernel-based methods are potential to use as well.

In this dissertation, we propose two novel network-based methods for malware detection that use Markov chain-based and String kernel-based method. Our proposed methods are able to overcome the limitations of all the previous methods that have been mentioned before. Namely, our methods use network communication data from real smartphone devices. It is useful since many malware will not perform malicious tasks if they detect virtual environment [133]. We take into account all the network packets of applications as they reveal important information, and which we believe should be retained in the models. The use of different malware families allows our models to learn a wide variety of malware behaviours. Furthermore, our methods exploit two efficient machine learning techniques that have not been applied to network-based malware detection methods to build classifiers.

2.2 Background

In this section we will introduce some background terminology used in our approach. Netflow data are temporal data whose features can be naturally represented as strings. In order to construct a learning system using netflow data, there are two approaches. The first approach is based on Markov chain theorem. This approach shows promising results in many applications. The second one employs string kernel based methods. Following, we briefly describe Markov chain and string kernel based methods, while we point the reader to appropriate references for a complete introduction on those topics.

Markov chain

A Markov chain is a random process that undergoes transitions from one state to another on a state space [100]. It needs to satisfy the Markov property, that is, the probability distribution of the next state depends only on the current state and not on the sequence of events that preceded it. Therefore, its property is referred as “memorylessness". Mathematically, a Markov chain is a sequence of random variables $X1, X2, X3, ...$ such that:

$$P(X_{n+1} = x | X_n = x_n, X_{n-1} = x_{n-1}, ..., X_1 = x_1) = P(X_{n+1} = x | X_n = x_n),$$

where $SS = \{x_1, x_2, ..., x_n\}$ is state space. It contains all possible values of X_i. A Markov chain is determined by an initial vector and a transition matrix. An initial probability distribution, defined on S, is a vector that specifies the starting state of the system. A transition matrix (T) is a square matrix whose transition probability T_{ij} characterizes that its current state is

i given that its previous state was j. Markov chains have many applications as statistical models of real-world processes.

Kernel methods

In machine learning, kernel methods [176], [177] are considered as state of the art that have shown impressive performance in many applications of different domains. Kernel methods owe their name to the use of kernel functions. A kernel function is a positive semi-definite function. It allows measuring similarities over pairs of data examples in a Hilbert space ($\mathcal{H}$) by computing inner product without an explicit transformation from raw data representation into $\mathcal{H}$. Mathematically, a kernel function of an input space $\mathcal{X}$ is defined as:

$$\begin{gathered} k : \mathcal{X} \times \mathcal{X} \longrightarrow \mathcal{R} \\ k(x_1, x_2) = <\varphi(x_1), \varphi(x_2)>, \end{gathered}$$

where $x_1, x_2 \in \mathcal{X}$, φ is a mapping: $\varphi : \mathcal{X} \longrightarrow \mathcal{H}$. When considering a finite domain, a kernel function, k, can be presented by a matrix called kernel matrix (K):

$$K_{ij} = k(x_i, x_j).$$

A kernel-based method can be divided into two modules: a general problem solver (a well-known method is support vector machine - SVM [45]) and a specific kernel function. Thus, its performance depends on the quality of both modules. Traditionally, classical machine learning techniques are defined over data in vectorial form. However, there exist many potential application domains where a vectorial representation of the data could lose important information or is even infeasible. Kernel methods can be employed to overcome the restriction of classical methods since they only operate on very general types of data and detect very general types of relations. In netflow traffic, we can use strings to present data features. Therefore, using string kernel-based methods to solve such kind of problem is a logical choice. In the next paragraph, we introduce some of the most used string kernels.

String kernels are functions that allow us to measure the pairwise similarity of any finite string couples on feature space. String kernels are widely used for biological data [114], netflow data [123], text analysis [118], etc. There exists a number of strings that have been proposed. A famous string kernel named p-spectrum string kernel is introduced in [114]. This kernel uses a feature space indexed by substrings of length p. The p-spectrum kernel $K_p(s, t)$ counts all the substrings of length p in both strings and kernel value is computed by taking the sum of the products. This algorithm's time complexity is quadratic in terms of $|s|$ and $|t|$. Another popular string kernel is String Subsequence Kernel [118]. It is a generalization version of p-spectrum kernel. The kernel is an inner product in the feature space consisting of all sub-sequences of length p. A sub-sequence is any ordered sequence of k characters occurring in the text though not necessarily contiguously. The sub-sequences are weighted by an exponentially decaying factor of their full length in the string, hence emphasising those occurrences which are close to contiguous. A sequence kernel called the Locality-Improved kernel is described in [198] relates recognition of so called translation initial sites in the sequence. This kernel puts more emphasis on local correlations while the dependencies between distant positions are of the minor importance or even do not exist.

2.3 Proposed Methods

In this section, we describe in detail the proposed method for malware detection. The overall schema of our method is presented in Figure 2.1. Our method captures network traffic of smartphones while apps are running on the device. This is elaborated in the data collection section. The collected data are processed, in order to convert the raw network data into state strings, based on netflows and a set of features. Finally, these state strings are used for training the Markov chain and String kernel based models. In the testing phase, the coming strings are compared against the models and outputs a labeled list with predicting if the coming application is benign or malicious. The following sections will explain in detail each step of the proposed method.

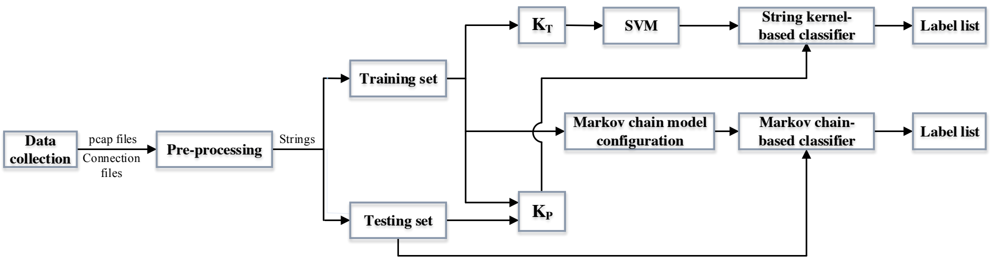


Figure 2.1: Schema of the proposed methods

Data Collection

Having a proper data set is crucial for any machine learning based method. Therefore, we collected the dynamic behavior of many app samples on real devices in order to train our models accurately. We used over 600 malware samples from [63], belonging to different malware families and over 1000 top rated applications from the Google play store, which were verified with the virustotal website [182] to be benign. The malware repository in [63] consisted of Genome Project [197], Derbin [17], M0Droid [120], virustotal [182] and some recent malware samples from contagio[2]. We used apps which required network connectivity. During application execution we captured the network traffic and the host-based features like which apps connected over network. After each experiment we collected a pcap file[3] containing the network traffic and a network connections log file. The exact steps will be explained in Section 2.4.

Pre-Processing

During the pre-processing phase we converted the outputs from the data collection into suitable format for our models. First, we converted the raw network data into netflows, which gives summarized statistics of the network connections of the smartphone. Using the connection details file, we labeled the netflows with the corresponding applications in the smartphone. The steps of generating the labeled netflow files are presented in Figure 2.2. We used Argus open source software for converting the pcap data into netflows and the Ra open source software for labeling the netflows. To extract meaningful features from the labeled netflows we used the method described by Sebastian et al. [73]. Since one netflow cannot represent all the network behavior, we used the aggregated traffic for an application using a unique four-tuple structure, which consists of Src-IP, Dest-IP, Dest-Port and Protocol. Some applications have many unique netflows, for example when a service is behind a domain which may constantly change the IP address or uses a pool of IP addresses. In order to aggregate these unique but different netflows, we used a special character (#) as a separator.

To analyze the behavior of an application three features were used and each netflow was assigned a state based on the three features. The three features comprised of size, duration and the periodicity of the netflow. The size and duration were directly extracted from the netflow while periodicity was computed from the time-stamp of the netflow. For example, if the time

[2] http://contagiodump.blogspot.it/

[3] http://www.tcpdump.org/manpages/pcap.3pcap.html

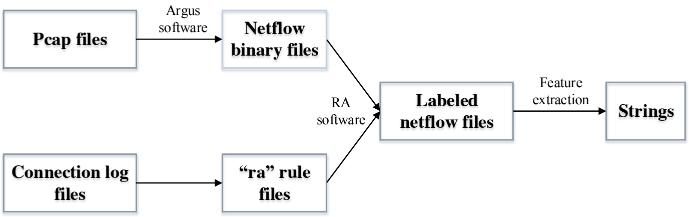


Figure 2.2: Steps in data preprocessing

difference between the first and the second flow is represented as T1 and the time difference between the second and third flows presented as T2, then the periodicity is computed by $T2 - T1$. When the app communications are not periodic, the periodicity is greater than zero. In order to represent the network behavior as states, we used thresholds for each feature. Using two thresholds, the features size and duration were broken down into three states each, while periodicity was broken down into four states. Table 2.1 presents the symbol table (ST) used for assigning a state to the netflow. After transforming the network behavior to state strings we use them for creating our models. The exact details of building our models using these strings and testing a coming application will be explained in the next section.

Model Configuration and Prediction

Consider a set of applications $A = \{A_1, A_2, \ldots, A_n\}$ and a set of labels $L = \{L_1, L_2, \ldots, L_n\}$ in which each application $A_i \in A$ is associated to a label $L_i \in L$. L_i equals to 1 if A_i is malware, and equals to 0 if A_i is non-malware. By collecting and processing data from A as presented in previous steps, we achieve a set $S = \{S_1, S_2, \ldots, S_n\}$ such that $S_i \in S$ is a set of strings $S_i = \{S_{i1}, S_{i2}, \ldots, S_{in_i}\}$ in which $S_{ij} \in S_i$ is a sequence of characters in Alphabet. We aim at constructing a model that can learn from S, L and detect malware applications. Once the models are built, for each coming application A_c that has corresponding set of strings $S_c = \{S_{c1}, S_{c2}, \ldots, S_{cm}\}$, our models need to classify the application as malware or not. Following, we describe how we construct our two proposed methods for malware detection based on Markov chain and String kernel.

Markov chain-based method

In this section, we characterize how we can build a Markov chain-based model for malware detection. For each application $A_i \in A$, we first form a transition

Table 2.1: Symbol table for netflow conversion [74]

Periodicity	Small Size			Medium Size			Big Size		
	Short	Medium	Long	Short	Medium	Long	Short	Medium	Long
Strongly periodic	a	b	c	d	e	f	g	h	i
Weakly periodic	A	B	C	D	E	F	G	H	I
Weak Non periodic	r	s	t	u	v	w	x	y	z
Strong Non periodic	R	S	T	U	V	W	X	Y	Z
Not enough data	1	2	3	4	5	6	7	8	9

matrix T_i that represents for its Markov chain M_i. As we mentioned before, A_i is corresponding to a set of strings S_i in which the string $S_{ij} = s_1 s_2 \dots s_{nij}$, such that $s_d \in ST,\ d = \overline{1, nij}$. The transition matrix T_i is built as follow:

$$T_i(x_u x_v) = \frac{q_{x_u x_v}}{p_{x_u}}, \tag{2.1}$$

where $p_{x_u} = \sum_{x_v \in ST} q_{x_u x_v}$ and $q_{x_u x_v} = \frac{f_{x_u x_v}}{\sum_{x_p, x_k \in ST} f_{x_p x_k}}$ with $f_{x_u x_v} = \sum_{S_{ij} \in S_i} |\{(x_u, x_v) | x_u = s_w, x_v = s_{w+1};\ s_w, s_{w+1} \in S_{ij}\}|$ is the frequency of x_u and x_v appear in consecutive order in state sequences S_i. We then take the longest string in $S_i = s_1 s_2 \dots s_n$ to represent for A_i and compute the probability p_i of its string generated by M_i by adopting (1) in the following formula:

$$p = \sum_{i=1}^{n-1} log(T_{s_i s_j}). \tag{2.2}$$

As a consequence, our model contains a set of n Markov chains $M = \{M_1, M_2, \dots, M_n\}$ and a set of probabilities $p = \{p_1, p_2, \dots, p_n\}$. To predict the coming application A_c, for each $S_{ci} \in S_c$, we first use the formula (2.2), to compute the probability of S_{ci} generated by each Markov chain $M_i \in M$. As a result, we have a set of probabilities $p_{ci} = \{p_{ci1}, p_{ci2}, \dots, p_{cin}\}$. We then use p and p_c to measure the difference d_{cij} of each S_{ci} compared to each representative string of each application $A_j \in A$ by using formula $d_{cij} = |p_j - p_{ci}|$. Subsequently, we obtain a set of differences $d_{ci} = \{d_{ci1}, d_{ci2}, \dots, d_{cin}\}$. The representative string of application in A associated with the minimum value of d_{ci} is the closest to S_{ci}. Therefore, the label of its application is assigned for S_{ci}. Finally, we obtain the set of labels for strings in S_c. To decide the label for A_c, the majority vote is used. Namely, if the number of predicted labels assigned as malware is greater than the number non-malware for the strings of the coming application A_c, it will be predicted as malware or else as non-malware.

String kernel-based method

In order to configure the string kernel-based model for malware detection, we first combine all the strings of the training applications to have an unique set, $S_T = S_1 \cup S_2, \ldots, S_n$. All S_i strings corresponding to an application A_i will have the same label as application A_i. We next apply a string kernel to the training set S_T. As a result, we get a gram matrix, K_T. This gram matrix [47] contains similarity measures of all string tuples and information needed for kernel-based methods. We then feed this matrix into Support Vector Machine (SVM) classifier. SVM works by finding the optimal hyperplane in Hilbert space that maximizes the minimal margin distribution of positive and negative hulls. Once the model is built, it is then used to predict the labels for new applications. For predicting a new application A_c, we compute the gram matrix K_P containing the proximities between any strings in S_c with any strings in S_T. The classifier takes K_P to calculate and returns a list of scores showing the likelihood of each string in S_c to be related to malware. We then use a threshold r to decide the label for each string in S_c. Similar to Markov chain based method, we use the majority vote to assign a label for A_c. In practice, we use tuning to find the proper value for r.

2.4 Experimental Setting and Evaluation

In this section, we first describe in detail the steps used to carry out our experiments. Then we will explain the procedure and metrics adopted to evaluate the performance of our new methods for malware detection.

Experimental Setting

In order to capture and analyze Android application network traffic, we designed an experimental setup to generate and capture the traffic. This allowed us to execute a large number of apps (malware and benign) and capture the network traffic. We used Nexus 6 smartphone (Quad-core 2.7 GHz with 3 GB RAM), running Android 5.1.1 as our Android device for executing malware and a Macbook Pro (Intel Core i5 @ 2.4 GHz with 8 GB RAM) running OS X server for creating a VPN connection with the smartphone for capturing the network traffic. Figure 2.3 presents the experimental set up.

Our automation algorithm, which was written in python and shell script performs: installing, executing, uninstalling of apps and collecting the generated data. The application dataset we used in our experiments is explained in Section 2.3. We executed five malware samples in one execution for a total of 2.5 hours. Our automation script begins with extracting the package name and activity name from the APK files and store them along with the path to APK in a Python list. Later, we installed the APKs and used the MonkeyRunner library to send various user inputs to the device. Further, our automation algorithm started the tcpdump to capture the network traffic on the laptop and ran AndroidLogger app [16] for collecting host data from the device (i.e., connection log files). At the end of our experiments, the tcpdump process was stopped and the log files from the AndroidLogger was retrieved from the smartphone and stored for analysis.

Proposed System Setup

An important design goal we would like to propose is the trade-off between user privacy and the computational power available for running our detection algorithms. Many existing solutions either try to run the detection algorithms on the devices under memory and computational constrains or send user private data from the mobile device to the cloud for processing. Many users are not willing to take part in a security solution where they have to send their sensitive and private information to a third party server for processing [16]. Our design choice, using a VPN connection through a dedicated personal computer, enables the user to securely browse the Internet and perform anomaly detection of her Android device. This setup can be especially applicable for security enthusiastic Android users or corporations where they provide VPN connections for their employees to access corporations resources [95]. Our proposed system design could enable these cooperations to run traffic anomaly detection on the employees smartphones as a second line of defense in addition to the security provided by the VPN connection. This may eliminate the need for applying policy enforcement mechanisms [43], [191] for the applications to be installed on the phone.

Figure 2.3, presents the proposed system using a personal computer. However, the personal computer can be replaced with an enterprise server and our detection algorithm can be run in any third-party cloud service as well.

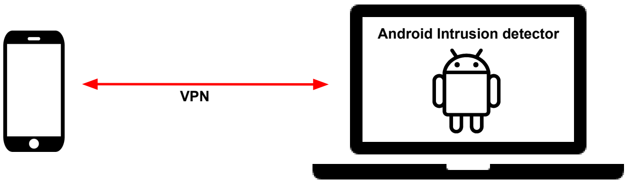


Figure 2.3: Proposed network monitoring and analysis setup for trade-off between user privacy and computational power.

Evaluation Method

In order to evaluate the performance of our methods, we employ k-fold cross-validation that is widely used for evaluating classification algorithms. To use this evaluation, the dataset is divided into k folds (k subsets), and the evaluation process is involved into k rounds. Each round uses $(k-1)$ folds for the training set to build the classifiers while the remaining one is used as the test set. We measure the performance of the method in each round and take the average of all k trials as the final performance of the algorithm. The advantage of this method is that every data point gets to be in a test set only once, and gets to be in a training set $(k-1)$ times so that the variance of the resulting estimate is reduced as k is increased. In our experiment, we choose k equal to 10. For string kernel-based model construction, we utilize subsequence [118] and p-spectrum string [114] kernels that is described in Section 2.2 to compute the gram matrix. We consider the subsequence of length two for both adopted kernel methods.

Evaluation Metrics

The following evaluation metrics are used to measure the detection performance of our methods:

$$Precision = \frac{TP}{TP+FP} \times 100\%; \tag{2.3}$$

$$Recall = \frac{TP}{TP+FN} \times 100\%; \tag{2.4}$$

$$F-measure = \frac{2 \times Precision \times Recall}{Precision + Recall} \times 100\%; \tag{2.5}$$

where TP (true positive) and FP (false positive) are the numbers of correctly and incorrectly identified malware, respectively. FN (False negative) is the number of incorrectly identified applications as non-malware. Precision is the fraction of retrieved instances that are relevant while recall (also called detection rate) is the fraction of relevant instances that are retrieved. F-measure considers both the precision and the recall of the test to compute the score. It can be considered as a weighted average of the precision and recall. The value of F-measure is large when both precision and recall are high, and small when either of them is poor.

Table 2.2: The performance of the proposed methods in experiment

Measures	Methods	
	Markov chain-based	String kernel-based
Recall	84.0%	82.8%
Precision	88.3%	75.0%
F-measure	86.1%	78.7%

2.5 Results and Discussion

Table 2.2 presents the performance of our two methods in the experimental setting.

According to the results described in Table 2.2, our methods show promising results. Overall, Markov chain-based method outperforms string kernel-based method. Related to recall measure, although Markov chain-based method has a higher value (84%) compared to string kernel-based method (82.8%), this difference is very small. That means the malware retrieving ability of both methods are close. However, the Markov chain-based method demonstrates high precision value (88.3%), meanwhile, it is only 75% in string kernel-based method. This leads to the significant difference in F-measure between two methods with 86.1% for Markov chain-based method and 78.7% for string kernel-based method.

Existing Anti-Virus software is in a constant battle to detect new Android malware families emerging with new threat tactics. According to the experiments in [197], the existing mobile Anti-Virus Softwares could detect, in the best case, 79.6%. In our experiments, we tested the performance of our methods with an updated malware repository belonging to different malware families. Our Markov chain-based method shows better result (see Table 2.2), showing potential to be applied for malware detection.

2.6 Summary

With the fast growth of smartphone market, we are witnessing more and more malware targeting smartphones, especially for the Android platform. In most attack scenarios, the adversary takes local or remote control of the mobile device (by leveraging system vulnerabilities via malicious apps) and send the collected information to some remote web service. Many research work have been conducted for detecting Android malware. However, very few of them investigate the possibility of detecting malware by solely focusing on Android network communications. A network-based anomaly detection method for Android enables to detect most of the anomalies while being relaxed on the applications installed on the smartphone. A typical use case is a company wanting to protect their digital assets such as smartphones/tablets given to their employees, while not controlling the applications used. The existing network-based anomaly detection research work shows promising results, but under restricted conditions. For example, using few malware samples, self-written malware or using a particular family of malware. In this dissertation, we investigate to what extent Android malware that can be detected by only observing the Android network traffic. We design a system, that achieves this goal using advanced machine learning techniques. We built a complete implementation of this system and run a thorough set of experiments using over 600 malware samples belonging to various families and over 1000 top rated benign app samples from the Google play store. Our system shows promising results achieving 86.1

Chapter 3

Preserving Anonymous End-to-End Communications in Adversarial Mobile Clouds for Smartphone Users

Mobile cloud computing is the paradigm that was built with the goal to save a resource very precious to mobile devices—their battery. The idea is simple: Pushing the execution of (parts of) mobile apps to remote servers residing on the cloud in order to avoid the energetic cost coming from the local execution on the device. The paradigm works best with computation-intensive applications with very limited access to device local resources like sensors, data. In fact, the more computation-intensive a given task, the more the device will benefit from executing it remotely. The less a given task needs to access local resources, the smaller is the device-cloud communication overhead to execute it remotely. Through the years, researchers have proposed offloading frameworks that take smart decisions on what to execute remotely [39, 46, 106], and solutions that boost the security of our devices [20, 22, 147] or enable efficient data/application backup [21]. Also, solutions that create virtual peer-to-peer networks of smartphone software clones in the cloud enable unprecedented and efficient, complex distributed protocols on mobiles [107, 108].

The nature of the mobile apps, but most importantly, their typical complexity, makes it very hard, if not impossible, to use privacy-preserving execution mechanisms like homomorphic schemes. In fact, these mechanisms, designed to operate in hostile environments (e.g., the untrusted cloud) over encrypted user data, are not suitable for application scenarios considering remote execution of mobile apps [175]. While offloading to the cloud, a mobile user has then to fully trust the cloud-side of the process. Not only is the cloud aware of which data and jobs the user is running, but it also knows exactly who is communicating to whom and what information is being shared.

In this chapter, we advocate that communication and file-exchange privacy among mobile cloud computing users is achievable, even under very powerful attacks. To this aim, we propose a protocol that, while supporting storage and computation offloading, implements anonymous end-to-end communications for mobile devices in adversarial mobile clouds. Specifically, we consider a strong attack model in which smartphones, cloud clones, the network operator, and the cloud provider are all adversarial entities and can collude to de-anonymize a communication. The cloud provider can both monitor the traffic from/to the user's cloud clones, and have access to the memory within the machines hosting them. Under this powerful and multifaceted attack model, all previous solutions for anonymous end-to-end communication in a mobile cloud computing setting, including ours [13], are unable to provide the requested privacy. In this chapter we challenge the common belief and come up with a solution that provides anonymity and unlinkability to the users.

Organization The rest of the chapter is organized as follows: Section 3.1 presents the literature review. Section 3.2 presents the system and attack models used as the basis of our anonymity protocol described in Section 3.3. Section 3.4 provides an analysis of the security and anonymity provided by our protocol. Section 3.5 presents experimental results on performance and efficiency. Finally, in Section 5.7 gives our concluding remarks.

3.1 Related Work

The mobile cloud computing paradigm, though initially designed with the offloading of heavy computations in mind [39, 46, 106], brings multifaceted benefits in a large number of application scenarios. It can enable more complex security mechanisms for smartphones [147], or help exploit the cloud to optimize incoming data-traffic, minimize the device connections to remote servers, and ensure efficient data backup in the cloud [20, 21, 19]. It opens the way to complex peer-to-peer services on mobile devices [22, 107, 108], otherwise impossible to run on our battery-limited smartphones. All these solutions assume full trust on both the cloud and the network operators providing the device-cloud communication channel. Also, encryption can come to hand for the protection of the user data stored in the cloud. Unfortunately, if the data/application code is encrypted with a key known only to the user, the cloud cannot be exploited for offloading anymore. In addition, encryption does not guarantee full user privacy. Both the cloud and the network operator in fact know how often a user is: *i)* Offloading computation to her cloud server (a.k.a. clone of the device [39, 107, 108]); *ii)* storing data on her cloud server; *iii)* exploiting the clone as a bridge to communicate/send the data previously stored on it to other users [107]. If the first two issues are unavoidable to achieve all the benefits of cloud computation offloading and backup, the user is increasingly concerned about her privacy when communicating with other users through the cloud.

Wired, wireless, and hybrid networked systems, have always brought the need of anonymous communication protocols [14, 125, 127, 128, 151, 152, 161, 194]. Most applicable solutions exploit chains of proxy nodes [34, 171], accumulating and forwarding source-encrypted messages in batches. Among them, TOR [171] is probably the most popular one. However, TOR is not applicable in the scenario in this chapter because devices and clones on the cloud are uniquely coupled. Also, the communication among two devices directly involves the corresponding clones. If the latter are compromised, they will identify the sender (receiver) even if TOR is employed when communicating with the corresponding clone.

With the increasing popularity of social networks, several works put the trust among friends as a means to achieve anonymity of communications [125, 127, 128, 151, 152, 161, 194]. However, these solutions either not fit at all for mobile-cloud computing scenarios, or are computationally heavy for battery-limited devices. To the best of our knowledge, our previous work [13] was the first attempt to address the issue of anonymous communications through the mobile cloud. It provided a user-tunable level of anonymity to sender (indistinguishable among α users) and receiver (indistinguishable among β users), the (α, β)-anonymity, as defined Section 3.2, in presence of colluding adversaries, including both cloud providers and network operators. The protocol worked under the assumption that the cloud clones of friend users could trust each other, and rely on each other to thwart anonymity breaches of communicating users. Differently from [13], in this work we consider a much stronger attack model: The cloud provider is able to look into a hosted clone's memory and read encryption keys stored therein; other clones, even friend ones, are malicious and can collude with both the cloud provider and the network operator to de-anonymize other user's communication. Our solution also supports computation offloading, in addition to storage offloading, balancing it with data confidentiality.

Other works have addressed a variety of issues in research areas similar to the ones considered in this chapter. Senftleben et al. [160] propose a decentralized privacy-preserving microblogging infrastructure based on a distributed peer-to-peer network of mobile users. The infrastructure, using device-to-device communications, is robust against censorship and provides high availability. Daubert et al. [49] present a solution to privacy-preserving sharing of smartphone sensor data and user-generated content via Twitter. The proposed solution ensures both confidentiality and anonymity of users and their messages. Finally, authentication, a milestone in sensitive-data handling platforms like the mobile cloud computing, is exahustely reviewed in the survey in [8].

3.2 System and Attack Models

The goal of our proposal is to achieve (α, β)–anonymity, that is, given a sender s and a receiver r, an adversary Adv should not be able to associate s to less than α users, and r to less than β users. In this section, we present the system and attack models at the basis of our proposal.

System Model. Our system involves different entities, namely *mobile devices* and *standalone apps* belonging to *users*, *cloud providers* hosting *clones* of the devices, *telco operators*, and a supporting *proxy* acting as a middleware between devices and clones. Mobile devices communicate through both the cellular network infrastructure and short-range ad-hoc wireless communication links (we will consider WiFi from now on as a representative technology for this layer). Each device d_k of user k is mapped with a clone c_k (i.e., a virtual machine) in the cloud, as well as with a standalone application std_k in the Internet. Having clones in the cloud is an emergent practice for offloading computations and communications, and for backup purposes. Hence, clones, connected through P2P links in the cloud, are likely to be entities already present and not necessarily introduced for the sake of our protocol. Also, we assume that information on friendship relations involving the system users are freely available (e.g., the public friendship information available in Facebook).

Key Distribution and User Registration. In our setting, all entities in the system (device, clone, cloud provider, standalone application, and proxy) have a private/public key pair and can securely verify the authenticity of others public keys. The clone keys are distributed by the hosting cloud provider, while the device and standalone public/private key pairs are locally generated and then certified by the trusted proxy. To enter the system a user needs to first register its device with the proxy, and certify device and standalone app keys. Then, the user registers its device with a cloud provider of her choice and have a clone assigned to her. During the registration phase the device exchanges the public keys with her clone. Finally, each device d_k shares a secret key SK_k with the corresponding standalone application std_k in the Internet, generated locally on the corresponding device and distributed when necessary through appropriate encryption mechanisms.

The standalone application, the user device, and the proxy are not controlled by the cloud provider. So, their private and secret keys are unknown to it.

Attack Model. We assume a strong adversarial model, where all communication channels in our protocol can be the target of an attack. We consider attacks on wireless communications among devices, communications with the telco operator and proxy, and communications between the clones in the cloud. We also assume different types of adversaries that are either malicious (i.e., possibly diverging by the protocol flow) or just honest but curious (i.e., aiming to violate the privacy, but without tampering with the exchanged messages). In particular, we consider malicious devices, malicious clones, and malicious standalone applications, while we assume honest but curious cellular network operator and cloud provider. Adversaries might collude among them and share their knowledge, such as for instance the device position within the cellular network and keys stored within clones.

Adversaries aim to identify sender s and receiver r of the communication or, in other words, to reduce the anonymity to (1,1)–anonymity. We note that the proxy is trusted and does not collude with any of the adversaries, although our solution is resilient to the scenario in which it is compromised by malicious adversaries [13]. In addition, a device or clone can attack or collude with an adversary to compromise the anonymity of a friend device or clone.

We underline that, when compared to the attack model considered in [13], our work considers a significantly stronger adversary model. In particular, *i)* we consider the ability of the cloud provider to look into the memory of the clones and search for encryption keys; *ii)* we depart from the assumption of having trusted friend clones (including c_s and c_r); *iii)* files can be stored by the clones in the clear.

3.3 Anonymity Protocol

Our solution provides an end-to-end anonymity communication protocol between mobile devices accessing the Internet. We assume a user carrying a mobile device associated with a clone on the cloud, and installing a standalone application supporting anonymity activities on its personal computer. Smartphone data are stored in the clear in the clone and synchronized with it through an encrypted channel. This approach allows to support *full computation offloading* in addition to storage offloading, while reducing as much as possible the parties able to access private data of the users. We note that, although our focus is on (α,β) end-to-end anonymity with support for storage/remote offloading, the offloading can be balanced with data confidentiality as discussed in Section 3.4. Clearly, the opposite scenario of full computation offloading is that of *full data confidentiality*, which can be provided by encrypting all data stored in the clones at a price of a reduced/nullified computational capability of the clones. An approach *balancing* full computation offloading (all data in the clear to the cloud provider) and full data confidentiality (all data encrypted) can selectively encrypt sensitive data, while storing the remaining data in the clear.

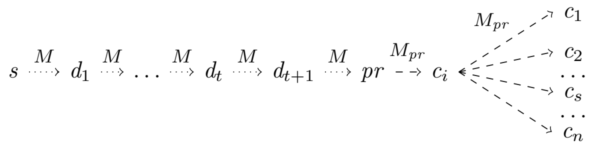
3.3.1 High-Level Overview of the Protocol

Each communication between sender s and receiver r is composed of three phases as follows [13]: *i) Sender communication*; *ii) clone communication*; *iii) receiver communication.*

Sender communication implements an anonymous communication between the sender s and corresponding clone c_s, through the proxy and a set of clones. The sender s initially sends its message through a multi-hop WiFi communication on an ad hoc WiFi network of devices. Randomly, a receiving device forwards the message to the proxy using the cellular network. The proxy receiving the message forwards it to a friend clone of c_s, which in turn broadcasts the message to all its friends including c_s. The last communications are carried out on the cloud.

Clone communication implements the part of the communication responsible for anonymously distributing a message between c_s and clone c_r of receiver r. Each friend clone of c_s involved in the sender communication forwards the received message to its standalone app through the Internet. Standalone apps then forward the message to a friend clone, say c_j, of clone c_r via the proxy. Clone c_j finally uses cloud-based communications to broadcast the message to all its friends in the cloud including c_r.

Receiver communication implements the communication between c_r and corresponding receiver r. It is the inverse of the sender communication and involves a proxy, the cellular operator, and a device in the proximity of



M=$\{[id_{cm}, \alpha, \beta, c_i, c_j, nonce_1, nonce_2]_{K^p_{pr}}, id_f, [id_{cm}]_{K^p_{pr}}, [c_s, nonce_1]_{SK_s},$
$[c_r, \beta, nonce_2]_{SK_s}\}$
M_{pr}=$\{id_f, [id_{cm}]_{K^p_{pr}}, [c_s, nonce_1]_{SK_s}, [c_r, \beta, nonce_2]_{SK_s}\}$

Figure 3.1: Protocol flow for Sender Communication

r. Each friend clone of c_r involved in the clone communication sends the received message to its standalone app through the Internet, which is then forwarded to the proxy. The received messages are filtered by the proxy, which forwards only the real message of s to r via a supporting device (WiFi peer of the destination). The last step uses a mix of cellular and wireless communications.

The following subsections formalize each of the aforementioned high-level phases by presenting, in details, the activities carried out by all the parties involved. Figures 3.1, 3.2, and 3.3 summarize the distribution of packets among parties illustrating also the content of each message in all three communication phases. Edges with a dotted line refer to wireless communications carried out on either ad hoc WiFi network (between peers) or cellular network (between peers and the proxy); edges with a dashed line refer to communications over the cloud (between clones and proxy); edges with a solid line refer to communications over the Internet (between clones, standalone apps, and proxy). The edge labels denote the messages exchanged on the corresponding links while the description at the bottom of each figure presents the messages in their entirety.

3.3.2 Sender Communication

Sender communication (Figure 3.1) determines the activities carried out in order to anonymously send a message from s to c_s.

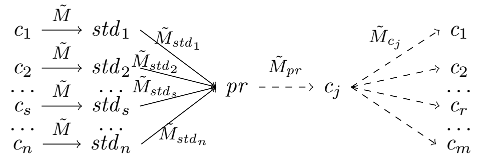
User. Similarly to [13], for each communication, user s defines preferences α and β at the basis of the anonymous communication and selects: *i)* One friend clone c_i whose social network (S_{c_i}) has at least α members, that is, $|S_{c_i}| \geq \alpha$; *ii)* one friend clone c_j of c_r whose social network (S_{c_j}) has at least β members, that is, $|S_{c_j}| \geq \beta$. This selection is done using the friendship database. Then, user s prepares a message M to be sent to c_s that includes: *(a)* The id id_{cm} of the communication, preferences α and β, the identity of c_i and c_j, and two nonces $nonce_1$ and $nonce_2$ encrypted with K^p_{pr} (the public key of pr); *(b)* the id id_f of the file to be sent, a number carrying no information neither on the user nor on the device; *(c)* the identifier id_{cm} of the communication encrypted with K^p_{pr} (the public key of pr); *(d)* the identity of c_s and $nonce_1$ encrypted with SK_s (the secret key shared between s and its standalone app std_s); *(e)* the identity of c_r, parameter β, and nonce $nonce_2$ encrypted with SK_s (the secret key shared between s and its standalone app std_s). We note that, to counteract an attack by the cloud provider that aims to uncover the sender s by identifying all clones with less than id_f files, s exploits a concealed file identifier. The same operation is performed by all involved clones to blindly identify the file to be sent according to the protocol. In this way, all selected files will be valid (including the correct file by clone c_s), and the attacker cannot gain any information on the sender. We also note that $nonce_1$ is used to let *i)* standalone app std_s know that it is the standalone app of sender s of the communication and *ii)* pr distinguish the correct messages among the received ones. Nonce $nonce_2$ has the same role as $nonce_1$ when std_r and r are involved. In addition, it is used to allow replies from r to s over the same anonymized channel (see Section 3.3.5).

The message M, prepared by the user as described above and depicted in Figure 3.1, is then sent to proxy pr using a probabilistic multi–hop WiFi forward to devices in its physical proximity. To guarantee α anonymity, s sends M *only when* it is surrounded by at least α devices. This process prevents the re-identification from nearby devices [13]. To this aim, all devices periodically broadcast a *probe request* with their identity to surrounding devices.

Device. Upon receiving M, device d_t forwards the message to either another device d_{t+1} in its proximity, or to proxy pr, through the cellular network, using a probabilistic function. The same process is repeated by all involved peers.

Proxy. Upon receiving message M, pr decrypts $[id_{cm}, \alpha, \beta, c_i, c_j, nonce_1, nonce_2]_{K^p_{pr}}$ using its private key K^s_{pr} and stores them for future computations. It then forwards $M_{pr} = \{id_f, [id_{cm}]_{K^p_{pr}}, [c_s, nonce_1]_{SK_s}, [c_r, \beta, nonce_2]_{SK_s}\}$ to c_i.

Clone c_i. It forwards the received message M_{pr} to **all** clones in its social network including c_s. We note that, by sending M_{pr} to all clones, α and β become lower bounds to anonymity, and c_i and c_j behavior is then independent from their value.



$\tilde{M}$=$\{f, [id_{cm}]_{K^p_{pr}}, [c_s, nonce_1]_{SK_s}, [c_r, \beta, nonce_2]_{SK_s}\}$

$\tilde{M}_{std_s}$=$\{[f]_{\overline{K}^p_{std_r}}, [c_r, \beta, nonce_2]_{\overline{K}^p_{std_r}}, [nonce_2]_{K^p_{c_j}}, [id_{cm}]_{K^p_{pr}}, [nonce_1]_{K^p_{pr}}\}$

$\tilde{M}_{std_k}$=$\{[f]_{\overline{K}^p_{std_r}}, [c_r, \beta, nonce_2]_{\overline{K}^p_{std_r}}, [nonce_2]_{K^p_{c_j}}, [id_{cm}]_{K^p_{pr}}, [rnd]_{K^p_{pr}}\}$

$\tilde{M}_{pr}$=$\{[id_{cm}]_{K^p_{pr}}, [f]_{\overline{K}^p_{std_r}}, [c_r, \beta, nonce_2]_{\overline{K}^p_{std_r}}, [nonce_2]_{K^p_{c_j}}\}_{K^p_{c_j}}$

$\tilde{M}_{c_j}$=$\{[id_{cm}]_{K^p_{pr}}, [f]_{\overline{K}^p_{std_r}}, [c_r, \beta, nonce_2]_{\overline{K}^p_{std_r}}\}$

Figure 3.2: Protocol flow for Clone-to-Clone Communication

3.3.3 Clone-to-Clone Communication

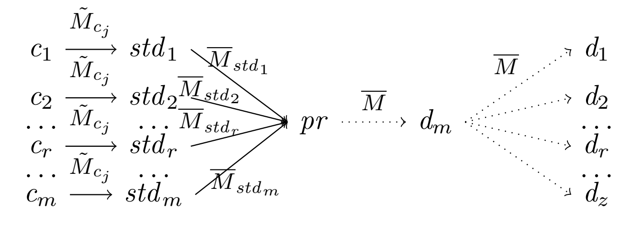
Clone-to-Clone communication (Figure 3.2) includes all activities aimed to anonymously send a message from c_s to c_r.

Clone. Each clone c_k receiving M_{pr} blindly identifies the file to be sent by applying a function (e.g., a modulo operation) on the received id_f. It then replaces id_f with f generating a new message $\tilde{M}$=$\{f$, $[id_{cm}]_{K^p_{pr}}$, $[c_s$, $nonce_1]_{SK_s}$, $[c_r$, β, $nonce_2]_{SK_s}\}$, and forwards it to the corresponding std_k on the Internet.

Each clone c_k then sends a file in the clear with the same identifier to the corresponding standalone application, showing the same behavior to all observing parties. We note that this approach based on blind file selection is robust to a scenario where the clone c_s is compromised and malicious (see Section 3.4 for more details). In this case in fact c_s behaves as any other clone in the system and is not able to understand what is going on in the communication, unless it also owns the corresponding standalone app std_s.

Standalone app. Upon receiving $\tilde{M}$, a standalone app first decrypts $[c_s, nonce_1]_{SK_s}$ using its secret key SK_k. If SK_k=SK_s, the decrypted chipertext contains c_k=c_s, and std_k identifies itself as std_s, that is, the application of the sender of a communication. std_s decrypts $[c_r$, β, $nonce_2]_{SK_s}$ using its secret key SK_s, encrypts $[c_r$, β, $nonce_2]$ using $\overline{K}^p_{std_r}$ (the public key of std_r), and encrypts $nonce_2$ using $K^p_{c_j}$ (the public key of c_j). It also encrypts f using $\overline{K}^p_{std_r}$ (the public key of std_r) and adds $[id_{cm}]_{K^p_{pr}}$ to the message. Nonce $nonce_1$ is finally added to the new message and encrypted with K^p_{pr} (the public key of pr).

After these activities have been completed, message $\tilde{M}_{std_s}$=$\{[f]_{\overline{K}^p_{std_r}}$, $[c_r$, β, $nonce_2]_{\overline{K}^p_{std_r}}$, $[nonce_2]_{K^p_{c_j}}$, $[id_{cm}]_{K^p_{pr}}$, $[nonce_1]_{K^p_{pr}}\}$ is generated and sent by std_s to pr. The message sent by std_k≠std_s involved in the communication is



$\overline{M}_{std_r}$=$\{[f, nonce_2]_{SK_r}, [id_{cm}]_{K^p_{pr}}, [nonce_2, d_m]_{K^p_{pr}}\}$
$\overline{M}_{std_k}$=$\{[f, nonce_2]_{SK_r}, [id_{cm}]_{K^p_{pr}}, [rnd]_{K^p_{pr}}\}$
$\overline{M}$=$\{[f, nonce_2]_{SK_r}\}_{K^p_{d_r}}$

Figure 3.3: Protocol flow for Receiver Communication

the same as $\tilde{M}_{std_s}$ with the only difference that $[nonce_1]_{K^p_{pr}}$ contains a random number *rnd*.

Proxy. Upon receiving a message $\tilde{M}_{std_k}$ sent by std_k, proxy pr decrypts the last two fields of $\tilde{M}_{std_k}$ using K^s_{pr}. The first field contains the identifier id_{cm} of the communication to which $\tilde{M}_{std_k}$ belongs, while the second field either $nonce_1$ in case the decrypted message is the correct one ($\tilde{M}_{std_s}$) or a random number *rnd* otherwise. Upon identifying $\tilde{M}_{std_s}$, the proxy waits until at least α messages belonging to the same communication id id_{cm} are collected. It then prepares message $\tilde{M}_{pr}$=$\{[id_{cm}]_{K^p_{pr}}, [f]_{\overline{K}^p_{std_r}}, [c_r, \beta, nonce_2]_{\overline{K}^p_{std_r}}, [nonce_2]_{K^p_{c_j}}\}_{K^p_{c_j}}$ and forwards it to c_j. We note that waiting for at least α messages and encrypting the whole message with the public key $K^p_{c_j}$ of c_j forbids re-identification by attackers able to observe the cloud and the standalone apps as discussed in Section 3.4. We also note that α and c_j are identified using id_{cm} previously stored by the proxy with α, β, c_i, c_j, $nonce_1$, and $nonce_2$.

Clone c_j. Upon receiving message $\tilde{M}_{pr}$, c_j first decrypts it using its private key $K^s_{c_j}$. It then decrypts $nonce_2$ again with its private key $K^s_{c_j}$. We note that $nonce_2$ is used to support bidirectional communications as discussed in Section 3.3.5. Clone c_j then forwards message $\tilde{M}_{c_j}$=$\{[id_{cm}]_{K^p_{pr}}, [f]_{\overline{K}^p_{std_r}}, [c_r, \beta, nonce_2]_{\overline{K}^p_{std_r}}\}$ to **all** c_k in its social network.

3.3.4 Receiver Communication

Receiver communication (Figure 3.3) includes all activities aimed to anonymously send a message from c_r to r.

Clone. Each clone c_k receiving $\tilde{M}_{c_j}$ forwards the message to its corresponding std_k on the Internet.

Standalone app. Upon receiving $\tilde{M}_{c_j}$, a standalone app first decrypts $[c_r, \beta, nonce_2]_{\overline{K}^p_{std_r}}$ using its private key $\overline{K}^s_{std_k}$. If $\overline{K}^s_{std_k}=\overline{K}^s_{std_r}$, the decrypted chipertext contains $c_k=c_r$, and std_k identifies itself as std_r, that is, the application of the receiver of a communication. std_r decrypts $[f]_{\overline{K}^p_{std_r}}$ using $\overline{K}^s_{std_r}$ (the private key of std_r), and encrypts f and $nonce_2$ using SK_r (the secret key of r). $[id_{cm}]_{K^p_{pr}}$ is then added to the message. Nonce $nonce_2$ and d_m are finally added to the new message and encrypted with K^p_{pr} (the public key of pr).

After these activities have been completed, message $\overline{M}_{std_r}=\{[f, nonce_2]_{SK_r}, [id_{cm}]_{K^p_{pr}}, [nonce_2, d_m]_{K^p_{pr}}\}$ is generated and sent by std_r to pr. The message sent by $std_k \neq std_s$ involved in the communication is the same as $\overline{M}_{std_r}$ with the only difference that $[nonce_s, d_m]_{K^p_{pr}}$ contains a random number *rnd*. We assume the standalone application to know devices in the proximity of r. Supporting device d_m is then selected based on β by extending the probe request-based mechanism used by user s to start the communication [13]. In particular, the probe request in sender communication phase is extended with the information about the number of devices surrounding the sender of the probe request. Then, r periodically collects and notifies std_r of neighboring devices around it, that is, the ones from which it received a *Probe request* including the number of their neighboring devices. In fact, neighboring devices with less than β devices in their proximity would expose the anonymity of r, whether selected as destination d_m. If this privacy condition is not met, then std_r would simply ask pr to stop the procedure (as for the scenario where c_r is the final destination).

Proxy. Similarly to the previous phase, upon receiving a message $\overline{M}_{std_k}$ sent by std_k, pr decrypts the last two fields of $\overline{M}_{std_k}$ using K^s_{pr} (the private key of pr). The first field contains the identifier id_{cm} of the communication to which $\overline{M}_{std_k}$ belongs, while the second field either d_m and $nonce_2$ in case the decrypted message is the correct one ($\overline{M}_{std_s}$) or a random number *rnd* otherwise. Upon identifying $\overline{M}_{std_s}$, the proxy waits until at least β messages belonging to the same communication id id_{cm} are received. It then prepares message $\overline{M}=\{[f, nonce_2]_{SK_r}\}_{K^p_{d_r}}$ and forwards it to d_m, via the cellular operator. Again, waiting for at least β messages and encrypting the whole message with the public key $K^p_{d_r}$ of d_r forbid re-identification by attackers able to observe the cloud and the standalone apps as discussed in Section 3.4.

Device. Upon receiving message $\overline{M}=\{[f, nonce_2]_{SK_r}\}_{K^p_{d_r}}$, d_m broadcasts the received message to the nearby devices. Among other devices, r receives the broadcasted message, decrypts it with $K^s_{d_r}$ and SK_r, and reads the file.

3.3.5 Discussion

The proposed protocol provides an end-to-end anonymity approach for a mobile cloud environment, which supports storage and computation offloading. It allows for a tunable tradeoff between the amount of computation that can be offloaded to the clones in the cloud and the amount of data that are potentially disclosed to the cloud operator. In our protocol, for easy of exposition, we considered one of the extreme scenarios where data in the clone's memory are all stored in the clear (high computation offloading, no confidentiality).

Our protocol employs encryption facilities to hide the two endpoints of a communication according to α and β anonymity preferences. Its behavior can slightly differ from the working discussed in this section depending on α and β. For preference α=1, sender s does not involve WiFi neighbors in its proximity during the sender communication phase, while it directly sends M to c_s via pr. For preference β=1, clone c_r directly receives message $\widetilde{M}_{pr}$ from pr in the clone-to-clone communication phase, and sends message $\overline{M}$=$\{[f, nonce_2]_{SK_r}\}_{K^p_{d_r}}$ to r via pr, bypassing d_m, in the receiver communication phase.

Bi-directional communications between s and r can be supported by adding a *response communication* phase to the protocol. This phase can be implemented either as a one-way communication switching s with r or by re-using the anonymous channel created for the communication from s to r. In the latter case, as discussed in [13], involved clones c_i and c_j must be the same for both directions, r must keep track of the identity of c_j, and in turn c_j of the identity of c_i. This can be done by using $nonce_2$ and the knowledge at the proxy.

Finally, there is a subtlety to consider when our anonymous protocol is executed. The file received by r using our protocol is not synchronized with the corresponding clone c_r to avoid sender-receiver re-identification by the cloud provider. If synchronized, in fact, the cloud provider could be able to observe a file stored in c_s that is then stored in c_r. A file received by r can be synchronized with c_r, if and only if the file has been previously modified by r.

3.4 Security Analysis

We assess the security of our protocol against possible adversarial entities aiming to reduce preserved anonymity to (1,1)-anonymity. In particular, we focus on the novel security features introduced by our proposal and evaluate: *i)* The security of our solution against a malicious cloud operator that tampers with the memory of clones (Section 3.4.1), *ii)* the security against malicious clones and standalone apps (Section 3.4.2), *iii)* the security against colluding cloud provider, clones, and standalone apps (Section 3.4.3). We note that, as far as malicious devices, malicious cellular network operator, and adversary tampering with the proxy are concerned, the security of the scheme proposed in this chapter is the same as the one discussed in [13].

3.4.1 Cloud operator tampering with clones' memory

Adversary and capabilities We consider an adversarial cloud operator that, beyond eavesdropping and analyzing all the traffic going through his domain, can also tamper with the memory of the clones it hosts.

Execution of the attack Since clones (e.g., Android virtual machines) are deployed in the physical architecture of the cloud operator, a malicious cloud can indeed inspect the memory of the clones, retrieve cryptographic keys, and decrypt all the communications involving the clone.

Defense Our proposal is resilient against this attack, for a simple but effective reason: All clones involved in the protocol (i.e., c_s, c_r, as well as the supporting clones) will "blindly" execute a set of operations according to the received messages. Since these operation are, for all the clones involved, "meaningful" operations (e.g., selecting and sending one of the files they store), the cloud operator cannot discern the actual c_s and c_r from the supporting nodes. More specifically, let us consider the *Sender Communication* phase of our protocol, as discussed in Section 3.3.2. Message M_{pr} received by each clone involved in this step does not require any computation. The clone just needs to select the file f corresponding to id_f and send it (in the *Clone-to-Clone Communication* phase) to the corresponding standalone application. Therefore, in the last step of *Clone-to-Clone Communication* and the first step of *Receiver Communication*, each of the supporting clones acts simply as a forwarder of message $\tilde{M}_{c_j}$, while c_j only decrypts a random number $nonce_2$ in $\tilde{M}_{pr}$.

Result of the attack Our protocol provides at least (α,β)-anonymity in the worst case.

3.4.2 Malicious clones and standalone apps

Adversary and capabilities In this scenario the clones can be honest-but-curios or act in a malicious way by tampering with the protocol. At the same time, the standalone apps can support the corresponding malicious clone, or co-operate with either the sender s (receiver r) to identify the other party involved.

Execution of the attack Honest-but-curious clones obeys to the protocol, while trying to understand whether they are c_s and c_r. Malicious clones also tamper with the protocol by dropping messages. Malicious standalone app of supporting clones can only retrieve the information about the fact that it is the app of neither the sender nor the receiver of the communication. On the other side, if the standalone app of the of the sender s (receiver r) is compromised, the standalone app knows it belongs to s (r).

Defense Similar to the previous scenario, nor the cloud provider neither honest-but-curios clones involved in a communication channel are able to infer the identity of c_s or c_r, and thus the one of the sender or the receiver. This simply follows by the fact that the knowledge of the clones cannot be bigger than the one of the cloud provider that hosts them. Malicious clones tampering with the protocol by dropping messages do not endanger the anonymity of senders or receivers either—note that, our protocol is general enough to forbid an adversarial clone from understanding its role in the protocol. All this attack could achieve is at most a denial of service—the messages are dropped by either c_s or c_r. However, in this case the corresponding users will eventually detect this behavior, and possibly change cloud provider to mitigate it.

If the standalone app of a malicious clone is also malicious, the user privacy is still preserved. In fact, even in the worst case scenario, when this happens for the sender (receiver) standalone app, the identity of the sender (receiver) is protected by the fact that the malicious *std* does not know the real identity of the corresponding clone. We achieve this by storing random numbers in c_s and c_r of $\widetilde{M}$, which are pre-installed in the standalone app without any link to the real identities of the involved parties.

Result of the attack Our protocol provides at least (α,β)-anonymity in the worst case.

3.4.3 Colluding cloud provider, clones, and standalone apps

Adversary and capabilities We consider the possibility of collusion among cloud provider, clones, and standalone apps.

Execution of the attack The attacker controls the network on the cloud, and either the couple (c_s, std_s), the couple (c_r, std_r), or both.

Defense The defense against this attack is given by the complexity of the attack itself. The attack might be very costly to be implemented, while it might provide limited results in terms of retrieved information. In fact, it requires to compromise clones and standalone apps of both sender and receiver, and have control of the cloud network (e.g., support by the cloud provider), to access communications involving a single pair of sender and receiver.

Result of the attack When only one among the couples (c_s, std_s) and (c_r, std_r) is compromised by an attacker also controlling the network in the cloud, our protocol can still guarantee (1, β)–anonymity when (c_s, std_s) is compromised, and (α,1)–anonymity when (c_r, std_r) is compromised. But, if the attacker compromises c_s, c_r, std_s, and std_r at the same time, and have the support of the cloud, it can violate the privacy of both sender and receiver. This is the only case in which the attacker fully identifies both parties in a communication.

We note that the proposed attack is very expensive since standalone apps and cloud clones reside on different platforms—the clones on the cloud, whereas the standalone apps on decoupled machines on the Internet—and requires a supporting cloud provider. Also, a single occurrence of this attack would uncover communications only involving a single pair s and d. Thus, though possible, it is almost impossible for an attacker to simultaneously have a full control of both clones and standalone apps for all possible sender–receiver couples in the system.

All remaining combinations including an attacker observing the cloud and the standalone apps are not able to achieve (1,1)–anonymity.

3.5 Experiments

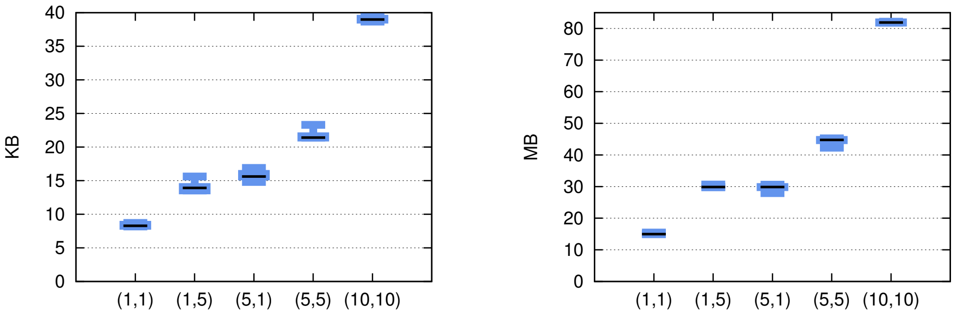
In this section we investigate on the possible overheads induced by our anonymity protocol. The evaluation focuses on the entities that suffer from hardware-related limits (the battery-limited smartphones), and on the proxy, which could introduce bottlenecks that harm the usability of the system. The protocol is tested for messages with two types of content: A regular text message of 160 Bytes (SMS) and a mp3 file of 3.87 MBytes (MP3). Each experiment is repeated 30 times and the results are aggregated. To measure the energy-related costs on the phone side we used the Power Monitor[1] meter. It samples the smartphone battery with high frequency (i.e., 5,000 Hz) so to yield accurate results on the battery power, current, and voltage. The mobile devices in our testbed were Samsung Galaxy S+ devices, 1.4 GHz Scorpion CPU, and 512 MB of RAM running Android 2.3. The proxy and the clones were running on a commodity laptop with the following characteristics: Ubuntu 14.04, Intel Core i7–4500U CPU, 1.80GHzX4, 8GB RAM. Algorithm AES with 192 bit key length was used for symmetric encryption and RSA with 1024 bit key length for asymmetric encryption.[2]

3.5.1 Evaluation on the proxy-side

The proxy plays a crucial role in the system and its anonymity: It is responsible of "coupling" device traffic towards clones and standard applications and vice versa. As such, it is important to study the amount of traffic per message the proxy needs to handle during a single one-to-one communication among devices. Recall that the proxy is involved in all the three steps of the protocol, while the traffic overhead is determined by the number of clones (standalone apps) involved in a single communication. Indeed, the proxy needs to receive as many messages as clones (standalone apps) both in the clone-to-clone and receiver communication steps, from which it discriminates the correct message to push forward in the protocol (see Figures 3.2 and 3.3). This number is strictly related to the (α, β) anonymity preferences of the communication: There are at least α clones (standalone apps) involved in the clone-to-clone step, and at least β clones (standalone apps) involved in the receiver communication step. For this reason, we have studied the traffic handled by the proxy varying α and β in the set $\{1, 5, 10\}$. The corresponding results are shown in Figure 3.4. As one might expect, the traffic handled by the proxy is higher for higher values of α and β, for both types of content exchanged among devices. What is surprising, however, is that the amount of

[1] https://www.msoon.com/LabEquipment/PowerMonitor/

[2] We note that, for convenience, we used 1024 bit length for asymmetric encryption though the latest NIST recommendations suggest using a 2048 bit long RSA key (http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800–57Pt3r1.pdf). This choice does not affect our experimental results as the RSA key is mostly used to encrypt the 192 bit long symmetric key in secret session communications.



(a) SMS Overhead.

(b) MP3 Overhead.

Figure 3.4: Traffic overhead per message varying (α, β) anonymity preferences. The graphics include the max, min, and quartiles values.

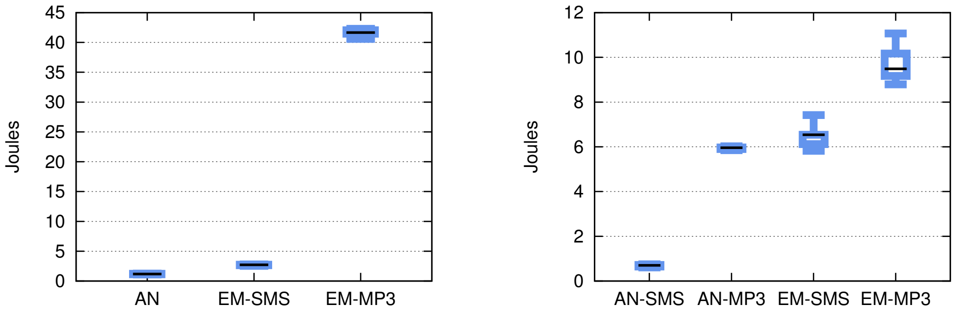
traffic does not grow in a proportional way w.r.t. the anonymity parameters. This observation indicates that higher anonymity guarantees can be met by our protocol without inducing severe traffic overheads to the proxy. Recall that in our testbed the proxy runs on a commodity laptop. Nonetheless, we believe that in real deployments the proxy could be efficiently implemented and deployed on a distributed set of high-performing servers, which will boost its performance and that of the overall protocol.

3.5.2 Evaluation on the device-side

The anonymity protocol involves costly encryption/decryption operations as well as sending message bundles that include the file index to be transmitted and other data necessary to guarantee the anonymity of the communication. In this section we discuss these costs from the perspective of the devices and compared them with the ones of a plain email protocol. Although the email protocol does not involve the cloud and does not guarantee any anonymity properties to users, it served as a benchmark in our evaluation.

Overhead on sender device

We start with the energetic costs on the sender device. They include the costs of *i)* the generation of the bundle M to be forwarded to the next hop by short ad hoc links (sender communication step) and *ii)* the communication through WiFi direct. We note that these costs are content-independent. Indeed, according to our protocol, the content is already on the cloud, and only the *id* of the corresponding file is sent within the message bundle to identify the corresponding file within the cloud and forward it anonymously



(a) Energetic Overhead on the source device: Anonymity protocol (AN) vs E-mail (EM).

(b) Energetic Overhead on the receiving device: Anonymity protocol (AN) vs E-mail (EM).

Figure 3.5: Energetic overhead on the source and destination devices. The graphics include the max, min, and quartiles values.

towards destination. The results are presented in Figure 3.5(a). It is clear how, despite the several cryptographic operations involved, the energetic overhead on the source-side is less than 1.25 J.[3] When compared to the plain email protocol the sender spends 2 times less for short messages (comparable to SMS) and up to around 20 times less for larger content (mp3 file).

Overhead on receiver device

Now let us consider the costs on the receiver side. Again, they include the energy spent for receiving the message bundle by d_m (receiver communication step), and for decrypting the bundle to finally read the content. We note that, in this case, the file is included in the bundle. This makes the costs dependent on the type of content that is being sent. The results are presented in Figure 3.5(b) and show that the consumption of our anonymity protocol is again considerably lower than that of the plain email protocol. In particular, it results 0.78 J for the short text case and around 6 J for the mp3 audio file. Considering the capacity of almost 22KJ of the battery of the involved devices, these values are particularly low. Most importantly, when compared to the plain email protocol the consumption is 3.5 times lower for the short text case, and around 2 times lower for the mp3 file. Our investigation showed that this difference is mostly due to the considerably longer download time of the email content, which is certainly dependent on the mailing server. This forces the destination device to keep its communica-

[3] When fully charged, the capacity of the battery of the devices involved in the testbed contains around 22KJ of energy.

tion interface up for a longer time, which induces considerably more energy consumption. Differently, in our protocol, the communication is ad hoc between the receiving device and d_m. The communication link exploits the WiFi direct protocol for device-to-device communication, which results more efficient from the receiving device's perspective.

Overhead on relay devices

The anonymity protocol involves also other devices—those that behave as relays through ad hoc links on both the sender and the receiver communication steps of the protocol. The devices involved in the sender communication step, however, have a much easier job than those involved in the latter. Indeed, they only need to forward the bundle M generated by the source a step further. According to our experiments, the energetic cost is less than 1 Joule. Differently, in the receiver communication step, we distinguish two types of devices: The d_m, in charge of broadcasting the message bundle to all β devices in its proximity, and a given device d_x which is not the destination of the message, but does not know it yet. It is clear that the cost induced to d_x is similar to that of the receiver. However, the cost of d_m is dependent on the parameter β of the protocol, which determines the number of WiFi-direct transmissions d_m needs to perform. According to our experiments, d_m will spend 1.8 J, 8.9 J, and 17.8 J for β=1, β=5, and β=10. Again, these values are very low w.r.t. the 22 KJ battery capacity of the devices involved in the testbed.

3.6 Summary

We presented a protocol for anonymous end-to-end communications among users in a mobile cloud environment, where the cloud clones handle part of the communication towards destination. The attack model considered is unprecedented. It includes devices, network operators, and the cloud provider behaving as malicious entities, and the possibility of all of them to collude. In this scenario, we built a delay-tolerant solution that provably guarantees (α, β)-anonymity, and evaluated its performance on a real-life testbed. Our future work will extend our approach to scenarios where exchanged files may contain information on sender/receiver, will depart from the assumption of having a standalone app available for each user in the Internet, and will provide a formal security analysis of our protocol using automatic cryptographic protocol verifiers, such as ProVerif.

Chapter 4

CaT: Cloud aided TLS

Smartphone is a very convenient tool providing easy access to information anywhere, anytime. With the development of mobile Internet services, more and more users are adopting smartphones as their primary communication device. According to Gartner statistics [52], the number of mobile devices shipped out in 2014 surpassed the number of traditional PCs (desk-based and notebooks). Furthermore, TechCrunch [144] reports that most of the digital media consumption now takes place via mobile apps (e.g., TechCrunch reports that during the period of June 2013 to June 2014, the digital time spent by mobile apps have grown by 52%, compared to 1% growth in desktop apps). At present, more mobile apps have a legitimate need for secure communication (e.g., e-commerce and e-banking). Transport Layer Security (TLS) and its predecessor, Secure Socket Layer (SSL) are the de facto standards for secure communications over the Internet [145]. TLS[1] is a protocol designed to provide end-to-end security that guarantees data confidentiality, integrity and authenticity to the communicating entities. However, TLS connections are computationally expensive to setup because they require public key cryptography.

Many of the applications available on Google Play store (674 out of 1000 most downloaded free applications) use TLS [59]. In order to establish secure TLS connections, these applications perform CPU intensive cryptographic operations, hence high energy consumption. At present, power management in mobile devices is a complex task because of the many hardware components (including various power hungry sensors), applications and several network access interfaces. Furthermore, the user behavior and how the operating

[1] For rest of the chapter we refer to SSL and TLS as TLS

system manages the applications, contribute to battery drain as well. Recent studies suggest the battery life of the smartphones has become a critical factor in user satisfaction [172].

We mainly focus on increasing the energy efficiency of the smartphone by proposing a possible solution for handling TLS connection establishment. Our methodology leverages the emerging mobile cloud computing paradigm [20, 148, 164, 18, 153] to offload computationally expensive (e.g., asymmetric cryptography) operations of the TLS handshake to the cloud.

Energy consumption of TLS on mobile devices was studied previously [145], [172], [24], [124]. Among the findings, an important observation was that asymmetric cryptography having the highest energy impact [145, 124]. TLS supports session resumption to reuse previously negotiated connection parameters to short-circuit the TLS handshake.

We conducted a survey to understand the frequency of full TLS handshakes on a smartphone. Furthermore, we measured energy consumption on the smartphone for full and abbreviated handshakes to better understand the problem. Motivated by the preliminary studies we designed Cloud-aided TLS (CaT) to use session-resume on the smartphone all times to connect with the web servers. We ran a thorough set of experiments to identify the feasibility of CaT. According to experimental results as well as statistical analysis we noticed CaT is not improving the energy efficiency of the smartphone on a significance level of 0.05. Finally we discuss the motivations of this results, as well as possible directions for improvements.

Organization The rest of the chapter is organized as follows. Related work is discussed in Section 4.1. Section 4.2 introduces the TLS protocol and discusses full TLS handshake and session resumption. Section 4.3 presents the problem statement. Section 4.4 presents the preliminary study we conducted to better understand the problem. Section 4.5 introduces the protocol design and Section 4.6 presents the protocol's implementation. In Section 4.7 we present the performance evaluation, and finally, Section 4.8 concludes the chapter.

4.1 Related work

Security protocols and cryptographic algorithms are known to have high computational requirements especially in resource constrained devices like smartphones [149, 10, 186]. Apostolopoulosb et al. [11] and Coarfa et al. [40] have analyzed the performance of SSL using a SSL Web server. Their performance analysis mainly focused on the timing of an entire client/server handshake. However, they considered desktop computers as clients, while in our work we are interested in smartphones. Wong et al. [187] and Hager et al. [79] measured the timing and the energy overhead of “raw" cryptographic operations on PDAs. Even though, such cryptographic algorithms are used in security protocols like TLS, they did not measure the performance of these algorithms when working together (e.g., TLS). Potlapally et al., [149] studied the timing performance of SSL on a PDA. However, the authors did not analyze the energy overhead whereas we analyzed both time and energy cost for TLS protocol. Diana [24], has considered the timing costs of full SSL handshake and session resumption. Here the author points out the session resumption uses less time for connection establishment. However, the work does not analyze the energy overhead. Furthermore, the work does not assess the two forms of session resumption: session ID and session-ticket.

While researchers have quantified and addressed the performance overhead of TLS, the energy implications are relatively less understood, in particular, for the session resumption. Our work fills this gap by performing a thorough set of experiments with full and abbreviated (session resume) TLS handshake using two session resumption mechanisms. Finally, we propose a novel security protocol, CaT, based on cloud computing and session resumption paradigms.

4.2 Background

TLS (and its predecessor SSL) is one of the most widely used security protocols on the Internet [149]. TLS uses a set of cryptographic algorithms to provide authenticity, integrity and encryption for user data. Authenticity provides a mechanism to verify the validity of the connecting servers and is achieved by using a chain of trust (through Certificate Authorities). Integrity provides a mechanism to detect message tampering and is achieved using Message Authentication Codes (MAC). Encryption provides a mechanism to obfuscate the user data and is achieved using ciphersuites and secret keys. TLS operates on top of the transport layer, (e.g., TCP/ IP).

It consists of two main layers: handshake and record layer. These two layers help establish a cryptographically secure data channel over insecure public networks (e.g., Internet). Handshake protocol authenticates the connecting peers and help exchange session keys. Record layer, using the negotiated session keys, protects the data over the Internet.

Negotiating session keys of the TLS handshake protocol is the computationally expensive, hence in terms of energy consumption as well [24]. We will start with a brief overview of the TLS full handshake and session resume.

4.2.1 Full TLS handshake

In-order to communicate securely, the client and the server must establish an encrypted channel. To establish this secure channel both parties have to negotiate a TLS version, verify the certificates and a cypher suit to use. The process involves exchanging four messages between the client and the server (without the TCP handshake).

Step 1 The client will initiate TLS handshake by sending a “ClientHello" message, which contains: TLS protocol version, supported cipher suites, client’s random value and other TLS options it may wish to use (e.g., session resume).

Step 2 The server responds with a “ServerHello" message, which contains: protocol version, a ciphersuite from the list provided by the client and the server certificate. Optionally it may request clients certificate and additional parameters for TLS extensions.

Step 3 If both sides negotiate a common version of cipher and the client accepts the certificate provided by the server, client initiates either RSA or Diffe-Hellman key exchange, which is used to establish the symmetric key for the session. Finally, the client sends the server an encrypted “Finished" message, to indicate the finish of the client handshake.

Step 4 The server will process the key exchange parameters sent by the client. After checking the message integrity by verifying the MAC, it will return an encrypted “Finish" message back to the client.

Step 5 Upon receiving the server Finish message, the client will decrypt it (with the negotiated symmetric key) and will verify the MAC. If the client is satisfied, the TLS tunnel will be established and the application will start exchanging data securely.

Figure 4.1 presents the messages passed during the full TLS Handshake.

4.2.2 TLS session resume

In the previous section, we noticed the full TLS handshake is computationally expensive (asymmetric crypto) and add an extra latency (two round trips) too. To overcome this, TLS protocol supports session-resume and two mechanisms to accomplish this: 1) Session-Identifier and 2) Session-Ticket mechanisms

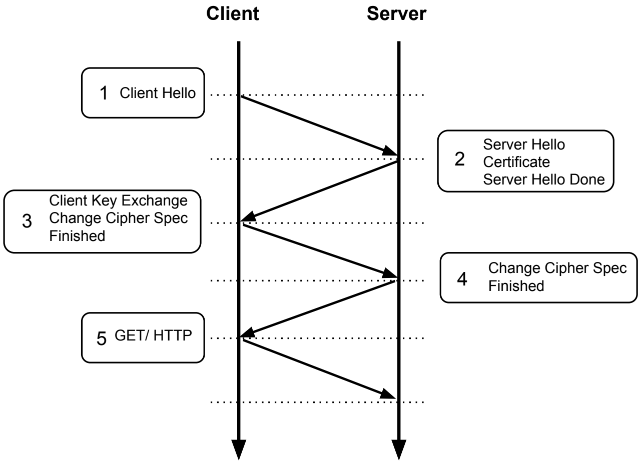


Figure 4.1: Full SSL handshake

4.2.3 Session-Identifier

This is described in detail in RFC 5246 [4]. In this mechanism, the server creates and sends a 32 byte session-identifier as part of the “ServerHello” message. The server, maintains a cache of session IDs and the negotiated session parameters for each peer. Clients, too, store the session ID information and include the ID in the “ClientHello” message when trying to establish a new session. If both server and client manage to find the correct session-identifiers in their caches, an abbreviated handshake can take place. This will allow the pre-negotiated cipher suite and keys. However, if the server notices a mismatch in the session IDs, the protocol will fall back to a full handshake to re-negotiate new keys.

4.2.4 Session-Ticket

The session-ticket mechanism is explained in RFC 5077 [3]. Session-ticket mechanism was introduced to overcome the limitations of the server-side deployment of TLS session caches. This eliminate the need for the server to keep per-client session state. A client willing to use session-ticket will indicate the session-ticket support in the Client Hello message. The server in response will send an empty session-ticket extension in the Server Hello message, to indicate it will send a new session-ticket using the “NewSessionTicket" message. The server sends the NewSessionTicket message before the server side ChangeChiperSpec message. If the server notices any anomalies with the client ticket, it will force a full TLS handshake.

Figure 4.2 presents the abbreviated handshake, showing the messages exchanged during a session resumption. The abbreviated handshake avoids one round trip and the costliest asymmetric operation of the TLS handshake.

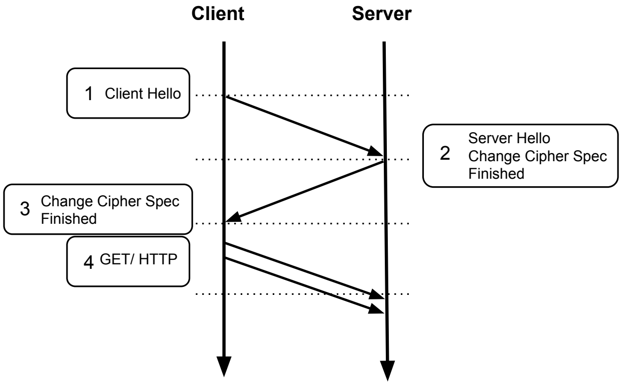


Figure 4.2: Abbreviated SSL handshake

4.3 Problem statement

Battery life of smartphones is a limiting factor in the usability of the smartphones [172]. Our problem statement was: *how to increase the energy efficiency of smartphones, with respect to TLS connection establishment?* We started with two preliminary experiments to investigate the problem better: 1) Conducting a user study to identify the frequency of full TLS connections on smartphones, 2) Measuring the energy and time consumption for full and resume TLS handshakes. Section 4.4 gives more information.

4.4 Preliminary Study

We started with a preliminary experiment to identify the frequency of full TLS handshakes on smartphone. For this we captured the smartphone network traffic for three university students, over a period of three days. For packet capturing we used `tpacketcapture` application [5] and used `wireshark` application to filter (`ssl.handshake.session_id` and `ssl.handshake.session_ticket`) only the TLS full handshake messages from the entire network traffic. Table 4.1 presents our findings.

According to our study, we noticed the number of full TLS handshakes are related to personal usage habits (e.g., application usage patterns, applications

User	No of SSL/TLS Conn.
User 1	825
User 2	1873
User 3	3269

Table 4.1: Number of full SSL/TLS connections for a three day user study

installed) and it varies among users (e.g., between 275 to 1090 connections per day).

Intuitively session-resume may enhance the energy efficiency of the smartphone, especially for the users with high number of full TLS connections.

To identify the significance of session-resume in energy saving, we measured the time and energy consumption for Full TLS connection and for TLS resume on the smartphone. In Figures 4.3 and 4.4 we report the average findings with the standard deviation per full TLS connection and per TLS session resume. We used the Facebook (FB) server for our experiments and used session ID and session-ticket methods for Full TLS connections and TLS resumes correspondingly. Also, we used the Wilcoxon ranked-sum test (Appendix A) to further analyze the results.

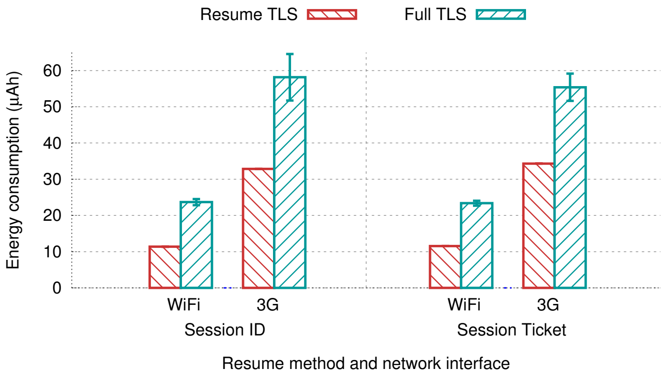


Figure 4.3: Energy consumption (in µA h) for performing a full TLS handshake and a session-resume over Session ID and Session-Ticket methods

According to the findings, session-resume saves significant amount of time and energy for both WiFi and the 3G network access interfaces at 0.05 significance level. From Figure 4.3, we can see WiFi interface using around half of the energy, while the 3G interface using little more, with respect to their Full TLS connection establishment energy. Table 4.2 reports the Wilcoxon significance levels for each energy experiments in Figure 4.3. Figure 4.4, presents the time consumption for full TLS connection and for TLS session

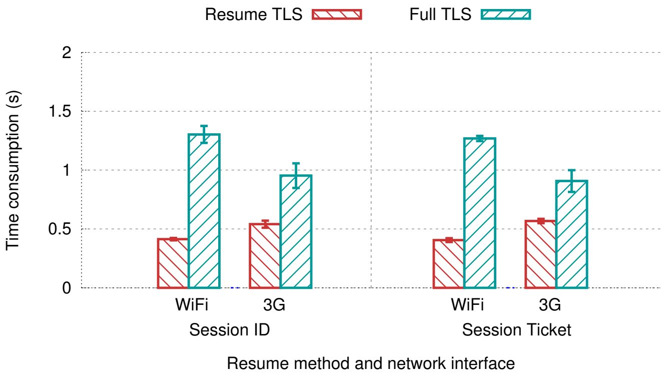


Figure 4.4: Time consumption (in µA h) for performing a full TLS handshake and a session-resume over Session ID and Session-Ticket

resume, over WiFi and 3G network access interfaces. Session-resume over WiFi consumes half the time for a corresponding full TLS connection and over 3G session-resume consumes little more than half the energy compared to the corresponding full TLS connection time.

Therefore, according to our findings session-resume saves energy and time compared to full TLS handshake. This is because of session resume, re-using previous session keys (avoiding asymmetric crypto) and using one round trip time less.

Based on the above observation, we considered assessing whether using session resumption on the smartphone can improve the energy efficiency of the smartphone.

In particular, we wanted to offload the asymmetric computation of the TLS protocol to a resourceful instance in the cloud.

One possible design solution was to make the cloud instance acts like a transparent proxy [2] where all the smartphone network traffic goes through a single point in the cloud. However, there are many existing solutions using a proxy to mediate the traffic for security and access control reasons (e.g., VPNs, F-Secure Freedome[2], CDroid [148]). We identified the following problems that might occur by sending all the traffic through a single point in the cloud. They are:

- Sending all the traffic through a single point, requires greater trust level assumption and hence more security for the cloud instance (e.g., an adversary who compromises the cloud instance will have access to all the traffic between the smartphone and the server). In a solution that

[2] https://www.f-secure.com/

avoids sending all the traffic through the cloud instance, the adversary will have to compromise both the cloud instance and the encrypted TLS channel between the smartphone and the web server.

- Sending all the traffic through a single point, is non-optimal in terms of bandwidth and latency. (e.g., the cloud instance may be far away, whereas the smartphone and web server may be close by: non-optimal routing). Further, the cloud instance could be a home PC [180], which may sit behind a relatively slow ADSL connection: non-optimal bandwidth usage.

Considering the high number of TLS connections, the energy efficiency of TLS session-resume and after analyzing the best way to use a cloud instance, we propose a new architecture to offload the secret key exchange of the TLS protocol to a cloud instance, while maintaining the smartphone/web server communications. Section 4.5 gives further details of our design.

4.5 Protocol Design

In this section we introduce: *Cloud-aided TLS (CaT),* which offloads the asymmetric crypto of the TLS handshake to a cloud instance.

We perform this asymmetric crypto operation in a cloud instance, to increase the energy efficiency of the smartphone. The cloud instance establishes the initial connection with the web server (as specified by the smartphone) where, it performs the initial full TLS handshake. During the handshake operation, the cloud instance saves the negotiated session parameters.

Our protocol passes these session parameters to the smartphone via a secure channel, where the smartphone resumes the connection with the web server using the session keys.

For session-resume our protocol supports session-identifier and session-ticket methods.

When designing the protocol, we assumed the data stored on the cloud instance is secure, especially the session related details. We protected all the communications between the smartphone/cloud instance (Section 4.6).

Further, all the communications between the smartphone/web server (TLS session resume) and web server/cloud instance (TLS session) are implicitly secure through TLS connections.

If the web server notices any anomalies in the session parameters, as a safety precaution, it falls back to a full handshake. Thus mitigating the security vulnerabilities of tampered session details on the cloud.

Protocol Description Figure 4.5 shows the overall components of our protocol. The arrow numbers indicate the sequence of the messages. In step I the device will indicate to the clone the web service it wants to connect with. The clone will connect with the web service and perform a full handshake in step II. As the step III of the protocol the clone will send the Session Context to the mobile device. Finally, in step IV the device will resume the session with the web service using the Session Context from step III.

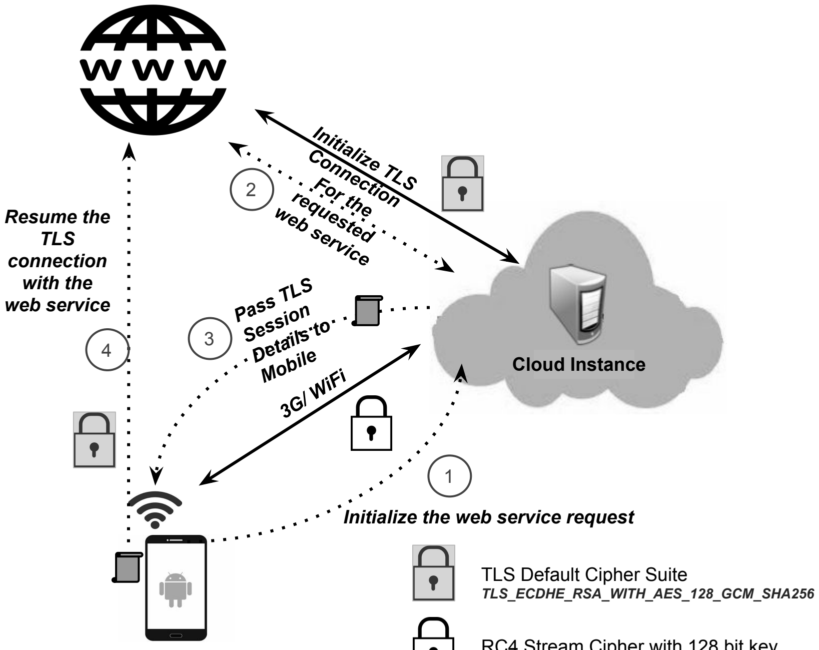

Initialize TLS Connection For the requested web service
2
Resume the TLS connection with the web service
4
3
Pass TLS Session Details to Mobile
3G/ WiFi
Cloud Instance
1
Initialize the web service request
TLS Default Cipher Suite
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
RC4 Stream Cipher with 128 bit key

Figure 4.5: Protocol design

Energy Model Now we introduce the energy model which we use to explain the results in Section 4.7.

Offloading a mobile computing task is a tradeoff between energy for offloading (uploading the data and downloading the result) vs. energy for local processing.

If we define E_{Device} as the energy required for local computation and E_{Cloud} as the energy for offloading, then the difference in the energy can be given as E_{Saved}:

$$E_{Saved} = E_{Device} - E_{Cloud} > 0. \tag{4.1}$$

A positive E_{Saved} means a saving in energy by offloading and vice versa.

Energy consumption (offload or local) can be represented as a function of power (P_x) and time (T_x). Therefore, energy E_x:

$$E_x = P_x \times T_x. \tag{4.2}$$

Energy for offloading can be further broken down as follows: energy to send the request to the cloud E_{s1}, the idle energy while waiting for the C to complete the full TLS handshake E_i, the energy for receiving the session parameters E_{sp} and energy to resume the session with the web server E_{s2}. This can be represented as follows:

$$E_{Clone} = E_{s1} + E_i + E_{sp} + E_{s2}. \tag{4.3}$$

By substituting Equation 4.2 into Equation 4.3, we get the following Equation:

$$E_{Clone} = P_{s1} \times T_{s1} + P_i \times T_i + P_{sp} \times T_{sp} + P_{s2} \times T_{s2}. \tag{4.4}$$

According to Equation 4.4, the execution time plays a crucial role in saving energy when offloading. If we define the local power consumption as P_l and local execution time as T_l, we can summarize the condition for energy saving by offloading as follows:

$$P_l \times T_l > P_{s1} \times T_{s1} + P_i \times T_i + P_r \times T_r + P_{s2} \times T_{s2}. \tag{4.5}$$

From Equation 4.5, we can identify a higher local execution time and a large gap between the local execution power and offloading power, would increase the benefits of computation offloading.

4.6 Implementation

We implemented the CaT protocol using C programming language following a client/server architecture. All the communications between the smartphone and the cloud instance was protected. To get indicative measurements we used RC4 stream cipher, with a 128 bit key. However, in a real deployment we plan to replace RC4, with a secure ciphersuite (e.g., a block cipher in a counter mode like Galois/Counter Mode-GCM).

For our experiments we used OpenSSL version 1.0.1g on Android and on the cloud instance. We cross compiled an OpenSSL client for Android, as an OpenSSL client is not available by default for Android. To complete the

OpenSSL setup on Android we installed certificates of trusted Root Certifying Authorities (CA's) too. We collected them from the Mozilla Firefox certificate bundle [1] and converted them to the PEM format.

We used the following two commands to create new sessions (session ID and with ticket options respectively):

- `openssl s_client -connect servername:443 -no_ticket -state -msg -sess_out session.out`
- `openssl s_client -connect servername:443 -state -msg -sess_out session.out`

and the following command to resume the sessions:

- `openssl s_client -connect servername:443 -no_ticket -sess_in session.out`
- `openssl s_client -connect servername:443 -sess_in session.out`

We wrapped these commands in our program implementation. C handled the full TLS in new session creation in CaT, new session creation was done by C, while D resumed the session.

4.7 Experiments

Under normal circumstances the mobile device cannot perform a session resume with a web server on the first attempt. Therefore we introduced CaT, which integrates Full handshake and session resume under one protocol. CaT allows the mobile device to establish a secure connection with the web service using resume on the first attempt. We tested our protocol CaT vs. the Full TLS handshake with the following servers: Facebook (FB), Google (G) and a sever that we control (which we refer to as X in our work to remain anonymous). We tested our protocol under two different network access interfaces WiFi and 3G. We used a Galaxy Nexus smartphone (Dual-Core 1.2 GHz with 1 GB of RAM) running Android 4.3. The Lithium-Ion battery capacity of the smartphone was 1.75 Ah with an operating voltage of 3.7 V. For the cloud service we used an Amazon T2.small instance (Intel Xeon processor operating at 2.5 GHz with 2 GB of RAM). We measured the power consumption using the Monsoon power monitor[3] connected to our phone.

[3] https://www.msoon.com/LabEquipment/PowerMonitor/

This device sampled the smartphone battery with high frequency (i.e., 5,000 Hz) to yield accurate results on the battery power, current and voltage. During the experiments, we switched off the smartphone screen to get accurate measurements. For each scenario we ran ten sets of experiments, each consisting 100 TLS connections to the web server. The histograms report per connection values. First, we analyzed the results further using the Wilcoxon rank-sum test [185] at 0.05 significance level.

Wilcoxon rank sum (WRS) test is a non-parametric statistical hypothesis test used to compare the two related samples or repeated measurements on a single sample to check the population mean rank difference. The test can be considered as an alternative to the t-test. However, WRS can be applied to any distribution contrary to t-test, which can be applied only to a normally distributed data samples. In our experiments, p-value less than 0.05 significance level implies a statistical difference in the two data samples. Table 4.2 presents the summary of the statistical analysis.

Test ID	**Sub-Category**	**P-value**	**W**	**z**	**r**
Full & Resume TLS on Phone (Fig. 4.3)	Session ID (WiFi)	p <0.001	55.0	-3.780	-0.845
	Session Tkt (WiFi)	p <0.001	55.0	-3.780	-0.845
	Session ID (3G)	p <0.001	55.0	-3.780	-0.845
	Session Tkt (3G)	p <0.001	55.0	-3.780	-0.845
Cat & Full TLS for FB (Fig. 4.8)	Session ID (WiFi)	p = 0.005	68.0	-2.797	-0.625
	Session Tkt (WiFi)	p = 0.023	75.0	-2.268	-0.507
	Session ID (3G)	p <0.001	56.0	-3.704	-0.828
	Session Tkt (3G)	p <0.001	55.0	-3.780	-0.845
CaT & Full TLS for X (Fig. 4.10)	Session ID (WiFi)	p <0.001	56.0	-3.704	-0.828
	Session Tkt (WiFi)	p <0.001	55.0	-3.780	-0.845
	Session ID (3G)	p <0.001	55.0	-3.780	-0.845
	Session Tkt (3G)	p <0.001	55.0	-3.780	-0.845
CaT & Full TLS for G (Fig. 4.11)	Session ID (WiFi)	p <0.001	55.0	-3.780	-0.845
	Session Tkt (WiFi)	p = 0.002	65.0	-3.024	-0.676
	Session ID (3G)	p <0.001	55.0	-3.780	-0.845
	Session Tkt (3G)	p = 0.005	68.0	-2.797	-0.625

Table 4.2: Summary of Wilcoxon rank-sum test. At a significance level of 0.05, there is a statistically significant difference between all the data samples.

We used two session-resume methods in our experiments. They are session ID and session-ticket methods. Figure 4.6 presents the average power consumption of CaT and full TLS connections. Using Figure 4.6 it is noticeable that the CaT protocol consumes less power than the full TLS connection with the web server. Figure 4.7 presents the time consumed for TLS connection establishment with both protocols. We can observe CaT consuming a longer time for establishing a TLS connection. As a result, according to Equation 4.2 the energy consumption of the CaT protocol increases. Figure 4.8 presents the energy consumption for both operations. With the WiFi interface, both protocols consumed almost the same amount of energy, while with 3G, CaT consumed more energy. According to our statistical analysis there is a significant difference at 0.05 level, in the energy consumption for the two methods. Table 4.2 gives detailed results of the statistical analysis.

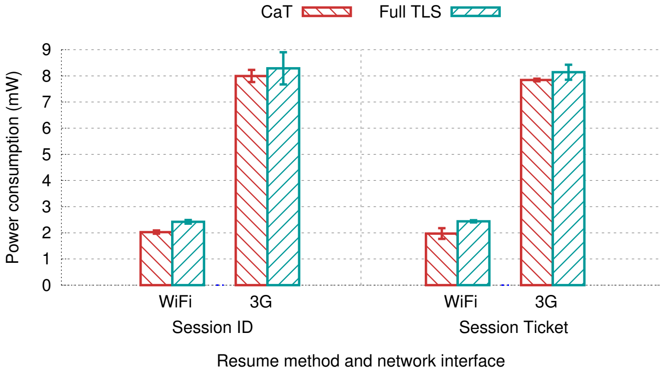


Figure 4.6: FB - Power

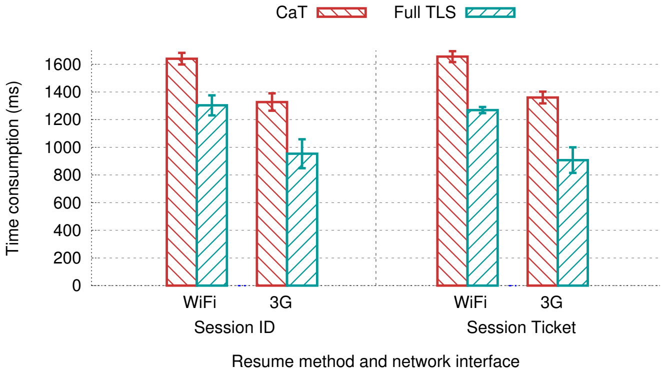


Figure 4.7: FB - Time

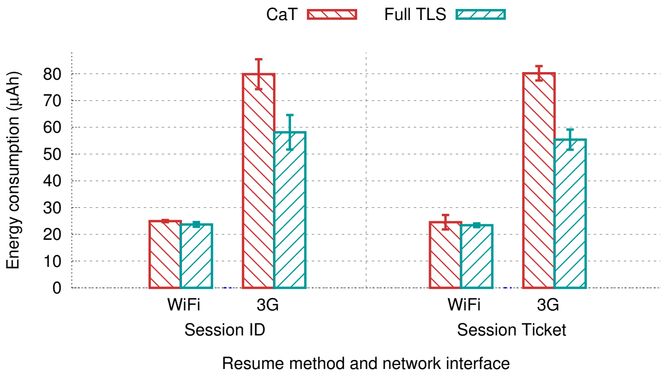


Figure 4.8: FB - Energy

Since energy consumption (as given by Equation 4.2), is a function of power and time we can conclude, the time increase in the CaT protocol, causes to consume more energy. Figure 4.9 gives a visual representation of the instantaneous power consumption when connecting with the FB server via 3G for one experiment. The full TLS reports high energy peaks in comparison to CaT and the execution time of full TLS is shorter compared to CaT. Therefore, according to Equation 4.5 we can generalize the operation time of CaT is currently the bottleneck in saving energy.

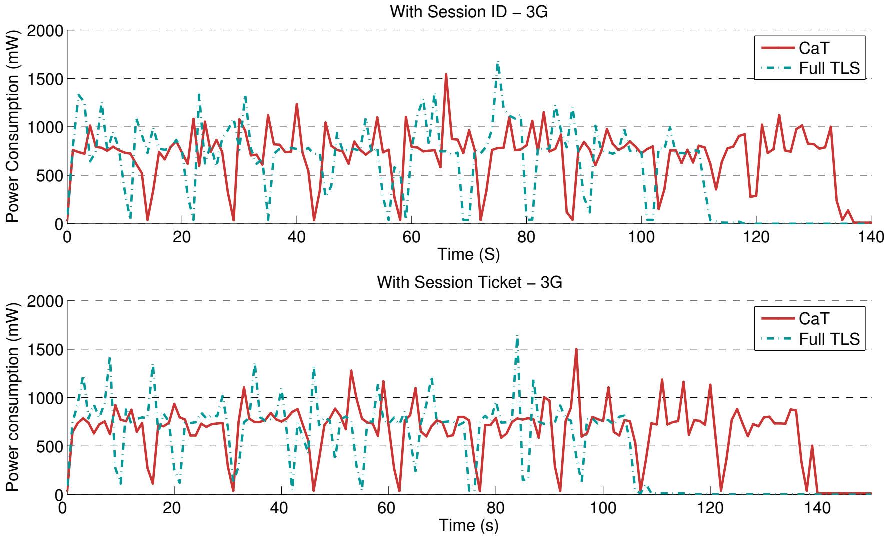


Figure 4.9: FB - Instantaneous power

We experimented with two other servers, where we observed a similar pattern. Hence, we report the total energy consumption for them. Figure 4.10 presents the total energy consumption for the Server X and Table 4.2 the statistical analysis results. Server X shows a similar behavior to FB experiments in terms of energy consumption.

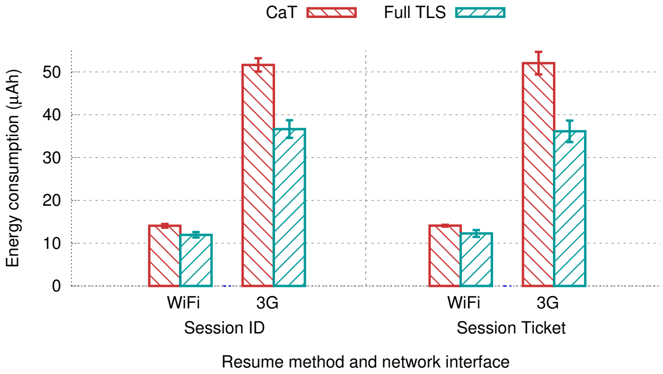


Figure 4.10: X - Energy

Figure 4.11 presents the total energy consumption for creating a session with the Google web server. According to the Figure 4.11, at 0.05 significance level (Table 4.2), CaT protocol consumes less energy, compared to the Direct method, when using WiFi. We will analyze the session ID method with WiFi interface scenario to understand the situation better. CaT protocol consumed 1.08 mA and 0.32 s to establish a TLS connection. While full TLS connection consumed 2.03 mA and 0.23 s to establish a connection. Applying the above values to the Equations 4.1 and 4.2, we can notice CaT saving energy compared to Full TLS.

Taking a closer look, we notice Google server reports the smallest execution times for all scenarios. Since Google servers have high availability, which allows to create connections faster with the server.

As a result, smartphone connects faster with the Google server, causing the crypto operations to be faster. This increases the load on the CPU in a given time compared to other scenarios.

Current smartphones [172, 168] have adjustable CPU frequency in-order to scale to the workload and save energy. According to [168, 111, 172], the energy consumption of the CPU increases exponentially with the increase of the CPU frequency.

In this specific scenario, we noticed the increased load on the CPU, causing the full TLS to use more power for the operation. Hence we notice CaT saving energy.

According to the observations we see the CaT protocol consuming less

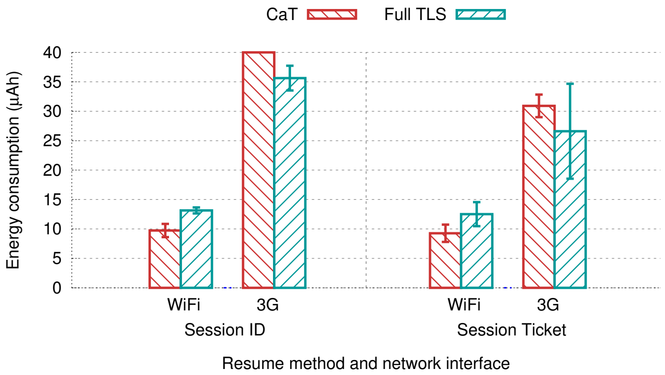


Figure 4.11: G - Energy

power at a given time. This is because of not performing the asymmetric cryptography locally on the smartphone. The energy consumption increases when more CPU intensive operations run on the smartphone. The full TLS connection performs the asymmetric operation in a burst. Since the time duration of the operation is less than CaT, the full TLS connection makes a saving in energy for most cases. However, the energy consumed by the full TLS can vary with the load on the CPU as in the Google server example, where the CaT protocol is efficient.

4.8 Summary

In this dissertation we proposed a new approach for establishing TLS connections for smartphones using computational offloading to handle the asymmetric key exchange. Intuitively offloading computationally expensive asymmetric key exchange from the smartphones is promising in improving energy efficiency of the smartphone. We started with some preliminary studies. According to our study we identified there are two groups of users: light and heavy with full TLS connections varying between 275 and 1090 connections per day. Furthermore, to identify the energy requirements, we performed full TLS handshake and TLS resume on the smartphone using session ID and session-ticket methods. According to the findings session-resume considerably reduced the TLS handshake time and also the energy usage for the operation. Taking into account the positive preliminary results, we implemented our approach, CaT to enhance the energy efficiency of smartphones. We ran a through set of experiments on CaT and compared the performance with the full TLS connection establishment on a smartphone. As it is possible to see in Figure 4.6 the instantaneous energy consumed is lower for CaT than for TLS. However, since the computations lasts more for CaT than TLS, overall Cat looses in comparison with TLS.

Part II

Applications of Drones for Secure Localization

Chapter 5

Application of Drones For Secure Localization Problem

The dependability of many distributed systems relies (often implicitly) on knowing the positions of the component devices. If the system believes that a device is in a position different from the real one, then it could infer wrong information and take wrong decisions. An adversary capable of changing the position of one or more devices (*displacement attack*) can deeply affect the system behavior with little effort. As an example, let us consider a sensor network deployed for pollution monitoring. The sensors could measure the density of dioxin in the air at different positions and report it to a centralized gateway, which eventually decides whether it should raise an alarm. An adversary willing to mask a pollution event could simply move some sensors in different positions, in a way such that it will avoid the detection (as illustrated in Figure 5.1). This attack is simple to carry out and difficult to detect.

Periodically measuring the positions of the devices is not enough to guarantee security. In fact, the majority of the positioning methods are vulnerable to attacks in which an adversary falsifies the position measurement [179, 86]. For example, if the position is inferred from the strength of a received beacon message, the adversary can confuse the measurement by sending fake beacons from wrong positions. Authenticating the beacons does not solve the issue, since the adversary could listen and replay authenticated beacons from different positions (*wormhole attack* [86]). Providing secure measurements of positions has shown to be a non-trivial problem [179, 193, 154]. A promising approach is *verifiable multilateration* [179]. Verifiable multilateration is a secure positioning technique that determines a position by measuring the

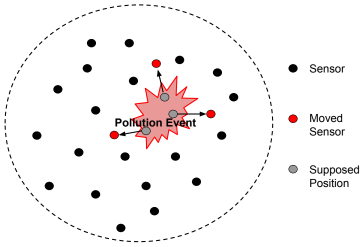


Figure 5.1: Displacement attack against an environment monitoring system. A hostile actor changes the positions of some sensors, and then she can pollute without being detected.

distances from (at least) three anchors by means of *distance bounding protocols* [28]. A distance bounding protocol is a cryptographic protocol able to measure a secure upper bound to the distance between two devices. Verifiable multilateration has the drawback that it requires many fixed anchors. The number of necessary anchors grows roughly linearly with the size of the area in which the nodes are deployed [179]. Another problem is that the fixed anchors must be truly "fixed". Otherwise an adversary could simply move an anchor to jeopardize the security of the system.

In this chapter, we explore the possibility of using the emerging *drone technology* to solve these issues. Drones, or Unmanned Aerial Vehicles (UAV), are aircraft with no human pilot. They can enjoy different levels of autonomy [101]: ranging from being remotely piloted to being completely autonomous in movements and decisions. In practice, our idea is to replace many fixed anchors with a single mobile drone. The drone follows a path that passes though a series of waypoints. At each waypoint, the drone "acts like" an anchor by executing a distance bounding protocol with a node. At the end of the path, each node has been measured from three different waypoints, and the position can be securely computed by means of verifiable multilateration. We thus completely eliminate the need for many expensive fixed anchors.

The problem is how to determine a convenient path for the drone. We cannot use existing path planning algorithms, because a valid path for verifiable multilateration must respect additional geometric constraints. In particular, the triangle formed by the waypoints must contain the node, otherwise the computed position will not be secure. Furthermore, other specific issues must be addressed, like the imprecision on the control of the drone movements.

Contribution

- We explore the approach of using drones to securely verify a set of positions by means of verifiable multilateration.
- We formally state the Traveller Location Verifier Problem (TLVP), that regards finding the shortest path for a drone to securely verify a set of nodes.
- We propose VerifierBee, a path planning algorithm that finds an approximate solution to TLVP.
- We run a thorough set of experimental evaluation of VerifierBee. The results of our experiments show that VerifierBee improves the path length of some 50% with respect to a simple solution.

Organization The rest of the chapter is organized as follows. Section 5.1 compares relevant related work. Section 5.2 introduces the basic concepts. Section 5.3 introduces the idea of drone-based verifiable multilateration. Section 5.4 formalizes the Traveller Location Verifier Problem. Section 5.5 introduces VerifierBee. Section 5.6 reports the results of our experimental evaluation. Finally, Section 5.7 concludes the chapter.

5.1 Related Work

Secure positioning aims at measuring the position of a device in the presence of an adversary that wants to falsify such a measurement. Researchers proposed many methods [98], offering different levels of security (provable or only statistical), and defending against different adversaries (external or internal). Čapkun and Hubaux [179] proposed a secure positioning method called *verifiable multilateration*. In this proposal, the system measures the distances from a set of trusted anchors by means of distance bounding protocols. The position is computed by means of multilateration, and it is considered secure if it lies inside the polygon formed by the anchors. In this chapter, we approach the problem of performing verifiable multilateration not with many fixed anchors, but with a single mobile drone.

Čapkun et al. [154] proposed a drone-based approach for secure location verification. In their system, the adversary is a malicious node that lies about its position. The drone sends a challenge message to the nodes, and then moves to a different random position. After an agreed period of time, the nodes respond with a response message, by which the drone infers their positions. Assuming that the malicious node ignores the drone's new position, it cannot falsify its own position in a coherent manner. Our approach is radically different, because it is based on verifiable multilateration, which is provably secure. As a consequence, we do not need to suppose that the drone's position is unknown by the adversary.

A similar problem to ours is drone-based data gathering from sensors [26, 54, 83, 121]. In this case, a robot (either aerial, terrestrial, or underwater) must collect data from a set of sparse and unconnected sensors. These works propose path planning algorithms that solve generalized forms of the Traveller Salesman Problem (TSP). The objective is usually to minimize the path length while respecting particular constraints. In this chapter, we propose a path planning algorithm to securely verify the positions of a set of nodes. This problem can be viewed as a generalization of the TSP as well. However, our problem is radically different from data gathering, because the path must respect a completely different set of constraints. For example, we must range each node from three distinct waypoints, whereas a single waypoint per node is sufficient for data gathering.

Another problem related to ours is drone-based (insecure) localization of ground devices [30, 44, 48, 143]. All these works do not have security in mind, and their position measurements cannot be considered trusted in a hostile environment. One of the simplest approaches is the one given by Corke et al. [44], in which a robot sweeps the entire area and periodically broadcasts its GPS position. The nodes collect such messages, called *position broadcasts*. They finally infer their own position by averaging all the received position broadcasts. Such a method is not secure, since an adversary could simply send fake position broadcasts, in such a way to confuse the nodes. Authenticating the position broadcasts does not solve the issue, since an adversary could listen to a legitimate broadcast and replay it on different positions. This is commonly known in the literature as the *wormhole attack* [86]. In general, all the localization method based on the strength of received messages like [30, 143] are poorly secure, since an adversary has an easy play on falsifying this information.

Dang et al. [48] uses a set of drones to localize a set of ground nodes, by measuring the relative distances between them. Our system measures the relative distances too, but it assures the security of the computed positions by using verifiable multilateration. Verifiable multilateration poses special requirements on the path that the drone has to follow. In particular, we must range each node from three waypoints, and the triangle formed by them must contain the node. This requirement was not present in classic trilateration.

5.2 Preliminaries

A *distance bounding protocol* [28] is a cryptographic protocol able to measure a distance between two devices, in such a way that an adversary cannot fake the measurement to be shorter than real (*reduction attack*). A distance bounding protocol determines a distance by precisely measuring the round-trip time between a challenge and a response message. The messages convey numeric quantities which are unpredictable by the adversary. The adversary cannot reduce the round-trip time measurement, because she should guess and transmit in advance the messages.

A simple example of distance bounding protocol is the following:

M1: $A \longrightarrow B : a$
M2: $B \longrightarrow A : b$
M3: $B \longrightarrow A : \mathrm{sign}_k(A, B, a, b)$,

where a and b are random numbers unpredictable by the adversary, and k is a shared secret, by which B authenticates the protocol execution. M1 and M2 are the challenge and the response messages. To precisely measure the round-trip time, the challenge and the response are usually transmitted by means of impulse-radio ultra-wideband (IR-UWB) PHY protocols [89], resulting in a precision of centimeters. From now on, we will say "A ranges B" as a shorthand for "A executes a distance bounding protocol with B."

Verifiable multilateration [179] is a method for the secure measurement of positions which leverages distance bounding. In verifiable multilateration, the position of a *node* is determined by measuring the distances between the node and at least three *anchors* whose positions are known (Figure 5.2).

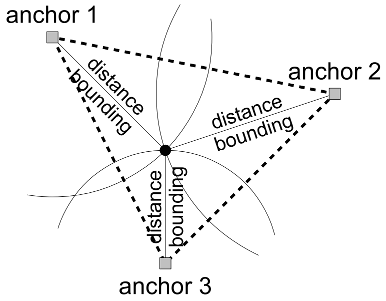


Figure 5.2: Verifiable multilateration. The computed position is accepted only if it lies inside the verifiable triangle (the thick dashed one).

The distance measurements are performed by means of distance bounding protocols. The node's position is computed by trilateration. Such a position is accepted only if it lies inside the triangle formed by the anchors (*verifiable triangle*). Otherwise, it is discarded as untrusted. In fact, if an adversary wants to fake a position inside the verifiable triangle, then she must perform a reduction attack against at least one distance bounding protocol, which is infeasible. Note that the coverage of verifiable multilateration is only the verifiable triangle, because the outside positions are discarded.

5.3 Drone-Based Verifiable Multilateration

Verifiable multilateration is able to securely measure the position of a node, but it needs a large number of anchors. In case of a set of nodes sparsely deployed on a large area and with a short communication range, it is necessary to deploy many anchors to reach them all. In addition, the coverage is restricted to the verifiable triangle, so it is not enough that three anchors are within the communication range: they have to "surround" the node in order to locate it securely. The scalability challenges of verifiable multilateration have been studied in [179]. In particular, the work in [179] proposed to place the anchors following a regular triangles' grid, in order to minimize the anchors necessary to cover a given area. Although a regular anchor placement improves the scalability, the number of anchors cannot scale better than linearly with the size of the area to cover.

With the development of the drone technology [101] and their increased availability on the markets, it became affordable to replace many fixed anchors with a single drone. The drone follows a *path*, touching a sequence of *waypoints*. At each waypoint it acts as an anchor (Figure 5.3), performing one or more distance bounding protocols with the nodes on the ground.

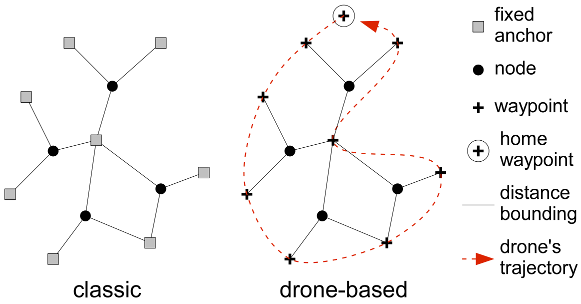


Figure 5.3: Classic vs. drone-based verifiable multilateration.

The drone shares a different secret k with each node, by which the distance bounding protocols are executed. The mechanism just described solves the scalability issue. The problem becomes now how to find a convenient path for the drone in order to securely measure a set of positions in an efficient way. We assume to have an a-priori knowledge of the nodes' positions, from which we compute the drone's path. We call them *supposed positions.* The supposed positions are not trusted, because an adversary could have displaced the nodes. Therefore we want to securely verify them by means of verifiable multilateration. If the positions determined by verifiable multilateration are not consistent with the supposed ones, a node displacement attack is detected.

The drone's path is centrally computed in an off-line fashion, and it must respect the following requirements.

- The path has to start and terminate at a special fixed waypoint, called *home waypoint* (this is where the path is computed and loaded in the drone). The drone takes off from the home waypoint, performs the mission, lands at the home waypoint again, and communicates the outcome of the position verification to some base station.

- Each node has to be ranged from three distinct waypoints, and the verifiable triangle formed by them must contain the node. This is required for the localization to be secure.

- The drone has a limited communication range. Far away nodes could be impossible to reach with a distance bounding protocol.

- The position determined by verifiable multilateration must be sufficiently precise in order to be useful for the verification. Such a precision depends (also) on the relative positions of the drone and the node. For example, if the drone ranges the node from exactly above, the precision will be extremely low. We have to avoid this.

- The path has to be tolerant to sources of imprecision. First, the movements of the drone are not perfectly controllable, because the wind strongly affects them. It is a good practice to provide for some tolerance, since the "true" waypoints actually visited by the drone could be different from the planned ones. The altitude could not be perfectly controllable too. Secondly, the supposed positions of the nodes could be imprecise.

- The mission time should be as short as possible. This is preferable for saving time and drone's battery life.

For the sake of simplicity, we assume that the drone moves at a constant speed. Minimizing the mission time is thus equivalent to minimizing the path length. We assume there are no obstacles to the drone's movements (e.g., walls, buildings, trees). The problem of path planning for secure location verification in presence of obstacles is interesting as well, but it falls outside the scope of the current work.

Many drones are limited in the movements they can do. For example, they cannot perform sharp curves or sudden direction changes (the problem is more stringent for fixed-wing drones, less for quadcopters). In order to respect the dynamic constraints of the specific drone, a path must be successively translated into a *trajectory*. The trajectory generation is a well-studied problem [76], and it falls outside the scope of the current work.

We suppose that all the nodes are on the ground, while the drone flies at a non-negligible *altitude* (h). We imagine the waypoints to be on the ground. The drone "visits" a waypoint when its position is above the waypoint (apart from drone control errors). The verifiable triangle is considered to be on the ground too. We assume that the ground is flat enough to allow the drone to be always in the line-of-sight with the ranged node.

We also suppose that, even if the position and the altitude of the drone are not perfectly controllable, they are actually *measurable*. The drone employs a technology that allows it to always know its own position and altitude with *negligible error*. An example of such a precise technology is differential GPS, which is sometimes installed on drones [81]. When the drone performs a distance bounding protocol with a node, what is measured is the *slant distance* (s), which is the line-of-sight one. However, for the aim of localization, we are interested in the *ground distance* (d), which is the one projected on the ground. The system computes the ground distance by:

$$d = \sqrt{s^2 - h^2}. \tag{5.1}$$

Once the drone has completed its path, three ground distances have been collected for each node. So the system can determine the positions of the nodes by trilateration, and verify if they are consistent with the supposed ones.

5.4 TLVP Formalization

In this section, we formalize the *Traveller Location Verifier Problem* (TLVP). TLVP can be considered as a generalization of the classic Traveller Salesman Problem (TSP), in which the nodes must not be visited, but rather verified for their positions. TLVP regards finding the shortest path to securely verify a set of ground positions by means of drone-based verifiable multilateration.

The *supposed positions* are a set of points on the Cartesian plane $\{N_1, \ldots, N_n\}$ that must be securely verified. A *path* (P) is a couple of sequences: a sequence of *waypoints* $\{W_1, \ldots, W_m\}$, and a sequence of *ranged nodes sets* $\{Rng_1, \ldots, Rng_m\}$. Each waypoint is a point on the Cartesian plane. The drone visits the waypoints in the order specified by the sequence. At each waypoint, the drone performs a distance bounding protocol with every node in the corresponding ranged nodes set. The path is closed, in the sense that the drone goes again to W_1 at the end. The first waypoint coincides with the *home waypoint* (W_{home}). Note that a node could be ranged from three waypoints that are non-consecutive in the path. For example, the drone could range twice a node in two successive waypoints, then move to a completely different zone to range other nodes, and finally return in the neighborhood to perform the third distance bounding. At the end of the path, each node must have been ranged from three distinct waypoints, and the verifiable triangle must contain the node.

We assume that the drone control error is bounded. We call such a bound the *drone control precision* (γ_W):

$$\|W_j - W_j'\| \leq \gamma_W, \tag{5.2}$$

where W_j' is the actual position from which the drone performs the distance bounding. In addition, the altitude is not perfectly controllable, but it is supposed to be bounded above by a *maximum altitude* (h_{max}):

$$h \leq h_{max}. \tag{5.3}$$

Finally, we assume that the error on the supposed positions is bounded. We call such a bound the *supposed positions precision* (ε_N):

$$\|N_i - N_i'\| \leq \varepsilon_N, \tag{5.4}$$

where N_i' is the actual position of the node.

5.4.1 Formalization of the constraints

If the supposed positions are imprecise, then the node could lie outside the verifiable triangle, and the drone will fail in verifying its position. To avoid this, the verifiable triangle must contain the whole circle centered in N_i and with radius ε_N. However, this is not enough, since we have to be tolerant also to the drone control error. If the drone control is imprecise, then the verifiable triangle actually drawn by the drone (*real verifiable triangle*) could be different to the planned one. As a consequence, the node could lie outside

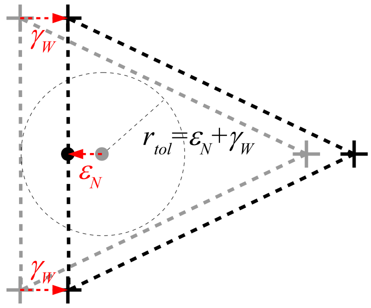


Figure 5.4: Error tolerance in the worst case. The visited waypoints (black crosses) are shifted with the planned ones (gray crosses). The real node's position (black dot) is shifted with the supposed one (gray dot).

the verifiable triangle again. To avoid this, it is sufficient that the verifiable triangle contains the whole circle centered in N_i and with radius $\varepsilon_N + \gamma_W$. We can prove this by considering the worst case, shown in Figure 5.4. The real waypoints are shifted (with respect to the planned ones) of γ_W on the direction orthogonal to the edge of the verifiable triangle. The real node's position is shifted (with respect to the supposed one) of ε_N on the opposite direction. The real verifiable triangle must contain the node even in this case. By geometrical evidence, we obtain this if and only if the planned verifiable triangle contains the circle with center N_i and radius $\varepsilon_N + \gamma_W$. We call such a radius the *tolerance radius* (r_{tol}):

$$r_{tol} \triangleq \varepsilon_N + \gamma_W. \tag{5.5}$$

The error on the slant distance principally depends on the employed IR-UWB technology and the quality of the receivers. We suppose that the error on the slant distance is bounded and we call such a bound the *slant precision* (ε_s). In the IEEE 802.15.4a IR-UWB standard [89], the slant precision is usually of the order of centimeters. For example, the IR-UWB transceivers commercialized by DecaWave have a precision of 10cm [50]. The *ground precision* (ε_d) is the bound on the ground distance error. It is always worse than the slant precision. Especially if the drone is in plumb-line above the ranged node, a small error on the slant distance will translate into a huge error on the ground one. Figure 5.5 shows an evidence of this.

If the altitude and the ground distance are sufficiently large compared to the slant precision, then the following approximate relationship will hold:

$$\varepsilon_d \approx \varepsilon_s \cdot \frac{1}{\cos(\alpha)} = \varepsilon_s \cdot \sqrt{1 + (h/d)^2}, \tag{5.6}$$

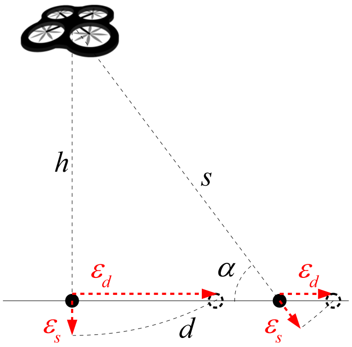


Figure 5.5: Ground precision. If the drone is above the ranged node, a small imprecision on the slant distance (ε_s) will translate into a huge imprecision on the ground one (ε_d).

where α is the angle of incidence of the slant distance to the ground (cfr. Figure 5.5). To guarantee a sufficiently precise localization, we impose an *objective ground precision* ($\bar{\varepsilon}_d$):

$$\varepsilon_d \leq \bar{\varepsilon}_d \Rightarrow \varepsilon_s \cdot \sqrt{1 + (h/d)^2} \leq \bar{\varepsilon}_d. \tag{5.7}$$

By expliciting d from (5.7) we get:

$$d \geq h \cdot \sqrt{((\bar{\varepsilon}_d/\varepsilon_s)^2 - 1)^{-1}}. \tag{5.8}$$

In other words, the ground distance must be large enough if we want to meet the objective precision. This depends on the altitude too: the more it is, the larger the ground distance must be. Also here, we have to take into account the worst case. We define a *minimal distance* (d_{min}) in this way:

$$d_{min} \triangleq h_{max} \cdot \sqrt{((\bar{\varepsilon}_d/\varepsilon_s)^2 - 1)^{-1}} + \varepsilon_N + \gamma_W. \tag{5.9}$$

If the ground distance between the planned waypoint and the supposed position is greater than or equal to the minimal distance, then we achieve the objective ground precision. In (5.9) we added ε_N and γ_W and we supposed the maximal altitude h_{max} to make sure that the requirement is respected even in the worst case.

Finally, the drone has a limited communication range. Given the maximum communication range (s_{max}), we define a *maximal distance* (d_{max}):

$$d_{max} \triangleq \sqrt{s_{max}^2 - h_{max}^2} - \varepsilon_N - \gamma_W. \tag{5.10}$$

If the ground distance between the planned waypoint and the supposed position is less than or equal to the maximal distance, then the node is within the communication range. In (5.10) we subtracted ε_N and γ_W to make sure that the requirement is respected even in the worst case.

To sum up, we identified three constraints: the tolerance radius (r_{tol}), which guarantees that the localization is secure; the minimal distance (d_{min}), which guarantees that the node is ranged with a satisfactory precision; the maximal distance (d_{max}), which guarantees that the node is within the communication range. These constraints refer to the single node, and they can be represented by three circles centered on the node's supposed position (Figure 5.6). The r_{tol}-circle must be contained inside the verifiable triangle, and the waypoints must lie at a distance between d_{min} and d_{max} from the node.

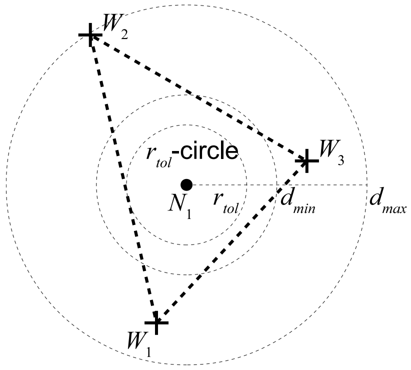


Figure 5.6: TLVP constraints representation. The r_{tol}-circle must be contained inside the verifiable triangle, and the waypoints must lie at a distance between d_{min} and d_{max} from N_i.

5.4.2 Final problem formulation

The Traveller Location Verifier Problem (TLVP) relates to the finding of the shortest path that allows a drone to securely verify the positions of a set of nodes under the r_{tol}, d_{min}, d_{max} constraints. Formally stated:

$$\begin{aligned} \underset{P}{\text{minimize}} \quad & \text{length}(P); \\ \text{subject to} \quad & \forall N_i : \\ & \text{triangle}(\text{wp}_P(N_i)) \supseteq \text{circle}(N_i, r_{tol}) \\ & \forall W_j \in \text{wp}_P(N_i)\ d_{min} \leq \|N_i - W_j\| \leq d_{max}, \end{aligned}$$

where length(P) indicates the length of the path P; $\text{wp}_P(N_i)$ indicates the triplet of waypoints which range node N_i according to the path P; triangle($\cdot$)

indicates the triangle formed by a triplet of waypoints; circle(N_i, r_{tol}) indicates the circle with center N_i and radius r_{tol}.

5.5 VerifierBee Path Planner

TLVP is a generalization of the classic Traveller Salesman Problem (TSP), which is NP-hard. We present VerifierBee, an algorithm that finds an approximate solution to TLVP. VerifierBee uses a TSP solver algorithm as a building block to find a first valid solution. Then, such a solution is iteratively improved, following a greedy strategy. The TSP solver is used as a black box. It is required to find an approximate shortest path that visits all the points in a list, and then returns to the first point (closed path). It is not required to be optimal. Approximate TSP algorithms are acceptable as well. Of course the performances of the TSP solver will affect those of VerifierBee both in terms of optimality and processing time. VerifierBee operates in three phases: (i) basic path computation; (ii) greedy improvement; (iii) waypoint reordering.

5.5.1 Basic path computation

VerifierBee computes an ordered list of waypoints: the home waypoint plus three waypoints for each node, placed at fixed positions to form a *minimal verifiable triangle* (Figure 5.7). The minimal verifiable triangle is a regular triangle centered on N_i and inscribed to a circumference of radius $\rho = \max\{2r_{tol}, d_{min}\}$. This radius is the smallest one which respects both r_{tol} and d_{min} constraints. VerifierBee orients all the minimal verifiable triangles with a vertex toward north. The angular orientation is indifferent,

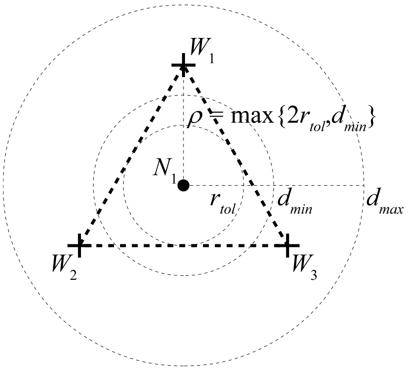


Figure 5.7: Minimal verifiable triangle.

because the successive greedy improvement phase will rotate and distort the triangles in order to find shorter paths. After having built the list of waypoints, we run the TSP solver on them to find an approximate optimal path that touches them all. The basic path is thus complete, and it is formed by $3n+1$ waypoints (where n is the number of nodes) and $3n+1$ ranged nodes sets. The first ranged nodes set is empty (it is the home waypoint), and the other ones contain a single node each. Figure 5.8 shows an example of basic path for 30 nodes.

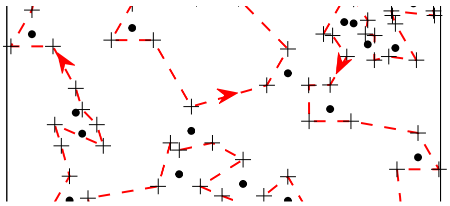


Figure 5.8: Example of basic path with 30 nodes. The black dots are the nodes, the crosses are the waypoints, the red dashed line is the path.

Note that the drone passes very close to each node. This makes the path suboptimal, since the drone does not use its full communication range. The basic path is a simple solution to TLVP. We will use it as a term of comparison to evaluate the performance of VerifierBee.

5.5.2 Greedy improvement

After having computed the basic path, VerifierBee changes it iteratively, following a greedy strategy. At each step, VerifierBee analyzes the possible changes (e.g., moving a waypoint in another position) and applies the most convenient one, that is the one that decreases more the total path length. The greedy improvement phase terminates when no change is possible or convenient anymore, meaning that we found a local minimum.

The changes are of two kinds: waypoint moving and waypoint pruning. *Waypoint moving* changes the position of a waypoint, while *waypoint pruning* removes a waypoint and "substitutes" it with another existing one. Both moving and pruning make use of the concept of *freedom space*. The freedom space of a waypoint is the area where the waypoint can be moved without violating any constraint of the problem (all the other waypoints remaining fixed). It can be computed geometrically, as illustrated in Figure 5.9. The curved borders of the freedom space are the limits of the d_{min} and d_{max} constraints. The straight borders are the limits of the r_{tol} constraint.

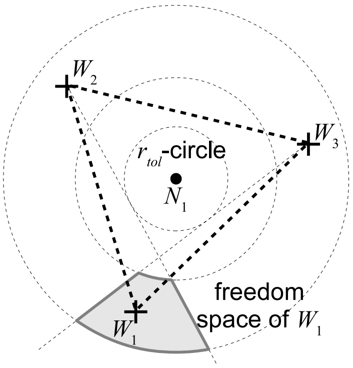


Figure 5.9: Freedom space of W_1 (the gray area). The two straight borders lie on the two rays originating from the other waypoints of the verifiable triangle (W_2 and W_3) and tangent to the r_{tol}-circle.

Waypoint moving changes the position of a waypoint, in such a way to shorten the global path. Figure 5.10 shows an example.

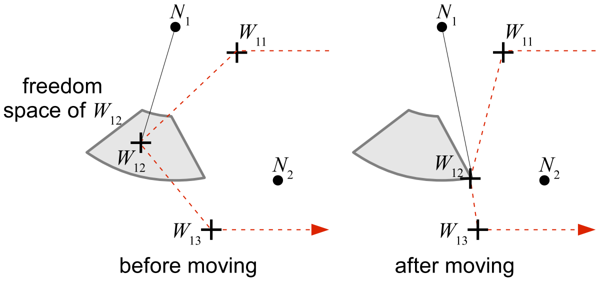


Figure 5.10: Waypoint moving. W_{12} is moved so that the drone shortens the path going from W_{11} to W_{13}.

The best position where to move a waypoint is always: (i) somewhere on the border of the freedom space (like in Figure 5.10), or (ii) coincident with another waypoint in the interior of the freedom space. In the latter case, we do not apply waypoint moving but rather waypoint pruning (see below), that is we eliminate the waypoint and substitute it with the other one. Therefore, waypoint moving always moves a waypoint along the border of the freedom space.

Waypoint pruning removes a waypoint (*pruned waypoint*) and substitutes it with another existing one (*substitute waypoint*). The drone will not visit anymore the pruned waypoint. As a consequence, it will miss to run a distance bounding protocol. The missing distance bounding is run when the drone passes through the substitute waypoint. Waypoint pruning reduces the total number of waypoints. Figure 5.11 shows an example of waypoint pruning. The pruned waypoint W_i and the correspondent ranged nodes set Rng_i are eliminated from the path, while the nodes that were in Rng_i are added to the ranged nodes set Rng_j of the substitute waypoint W_j. It is possible to prune a waypoint when its freedom space contains the substitute waypoint. The home waypoint cannot be pruned. After pruning, the substitute waypoint has to range two nodes instead of one. Consequently its freedom space will narrow, because it has to take into account the constraints relative to both nodes. The resulting freedom space is the intersection of the freedom spaces relative to the single nodes. In some cases, the freedom space of the waypoint to prune contains many waypoints. All these waypoints are suitable candidates to be the substitute waypoint. Which one to choose is indifferent in terms of path length. VerifierBee chooses the one which narrows less its freedom space, in such a way to leave more “freedom” to the next steps of the greedy improvement.

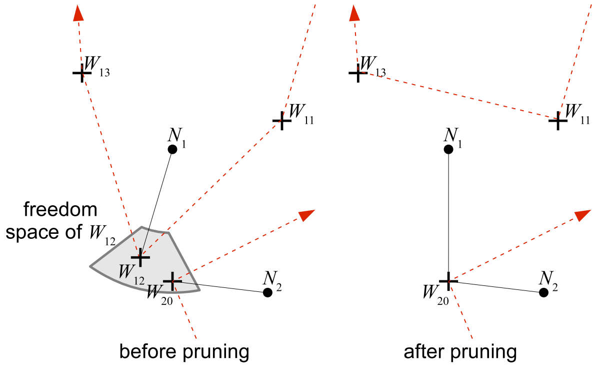


Figure 5.11: Waypoint pruning. W_{12} is pruned and substituted by W_{20}. When the drone visits W_{20}, it runs two distance bounding protocols: one with N_1 and one with N_2.

To sum up, the greedy improvement phase computes all the possible waypoint movings and prunings. If no change is further possible or convenient, the phase terminates. Otherwise we apply the most convenient change (either moving or pruning) and then we recompute all the possible changes again.

5.5.3 Waypoint reordering and complete algorithm

The greedy improvement may change the position and the number of the waypoints, but it does not change their order, which remains the same of the basic path. As a consequence, sometimes it is convenient to reorder the waypoints by running the TSP solver again. This is the third phase of the

Algorithm 1: VerifierBee

```
Require: $\{N_i\}$, $W_{home}$, $\gamma_W$, $\varepsilon_N$, $\varepsilon_s$, $\varepsilon_d$, $h_{max}$, $s_{max}$
Determine $r_{tol}$, $d_{min}$, $d_{max}$ by means of Eqs. 5.5, 5.9, 5.10
$Path \leftarrow$ a list of waypoints, one on $W_{home}$,
  and the others on the minimal verifiable triangles.
$Path \leftarrow$ SolveTSP($Path$) {basic mission}
loop
  $Path \leftarrow$ GreedyImprove($Path$)
  if $Path$ has not been improved then
    exit loop
  end if
  $Path \leftarrow$ SolveTSP($Path$)
  if $Path$ has not been improved then
    exit loop
  end if
end loop
return $Path$
```

VerifierBee algorithm. The greedy improvement and the waypoint reordering phases are repeated, until the path length stops decreasing.

Algorithm 1 shows a pseudo-code description of VerifierBee. The function SolveTSP(·) is our black-box TSP solver. The function takes a path, reorders the waypoints to form an (approximate) optimal path, and then returns such a new path. The function GreedyImprove(·) takes a path, performs a greedy improvement, and returns the resulting path.

Figure 5.12 shows an example of VerifierBee path for 30 nodes (the same ones of Figure 5.8). This path is much shorter than the basic path of Figure 5.8. Many waypoints have been pruned and the other ones have been moved in more convenient positions. The path passes very close to the external nodes (N_1). On the contrary, interior nodes are ranged from far away (N_2). Note also that nodes close to each other (N_3 and N_4) are ranged by the same waypoints, and enclosed by the same verifiable triangle.

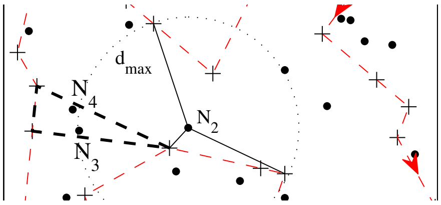


Figure 5.12: Example of VerifierBee path.

5.6 Experimental Evaluation

We implemented VerifierBee with the Matlab programming language and tested its performance under different conditions. For the TSP solver, we employed an off-the-shelf algorithm available on Mathworks [105].

We assumed the following parameters for our experiments: a slant precision of $\varepsilon_s = 10cm$ (claimed by DecaWave for their IR-UWB transceivers [50]); an objective ground precision of $\bar{\varepsilon}_d = 25cm$; a communication range of $s_{max} = 300m$ (claimed by DecaWave for their IR-UWB transceivers [50]); a maximum altitude of $h_{max} = 160m$; a drone control precision of $\gamma_W = 10m$; a precision on the supposed positions of $\varepsilon_N = 5m$. The TLVP constraints stemming from these parameters are: $r_{tol} = 15.00m$, $d_{min} = 84.83m$, $d_{max} = 238.77m$. We simulated a random deployment of the nodes on a $1000m \times 1000m$ map, and we executed VerifierBee to find a path that securely verifies their positions. We put the home waypoint on the south-west angle of the map. Figure 5.13 shows the average path length for different numbers of nodes.

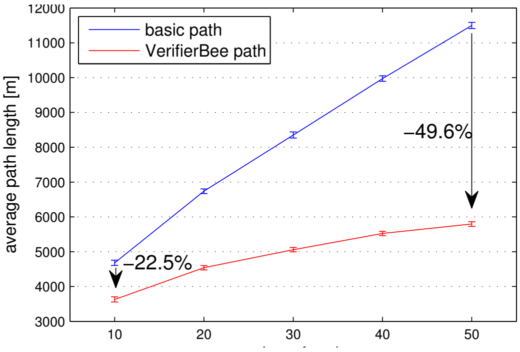


Figure 5.13: Average path length with the number of nodes. Each point stems from 100 experiments. 95%-confidence intervals are displayed in error bars.

We can see that the basic path increases linearly, whereas the VerifierBee path shows a sub-linear trend. The length saving of VerifierBee with respect to the basic path is quite significant, especially in case of many nodes (−49.6% for 50 nodes). This is because the drone ranges more nodes from a single waypoint if the nodes are denser. Figure 5.14 shows the average processing time for the basic path computation and for the complete VerifierBee algorithm, running on a 2.4GHz Intel Core i5 processor.

A path for 50 nodes takes roughly 4 minutes to be computed. This should be fully acceptable for an off-line computation. We remark that VerifierBee

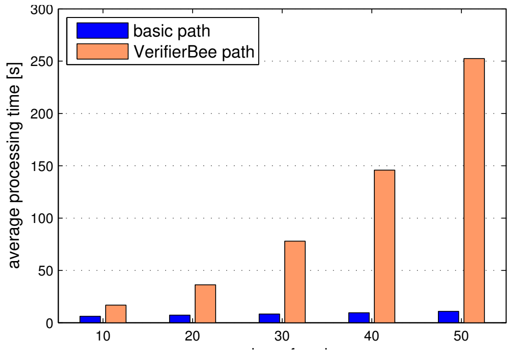


Figure 5.14: Average processing time with the number of nodes for 100 experiments.

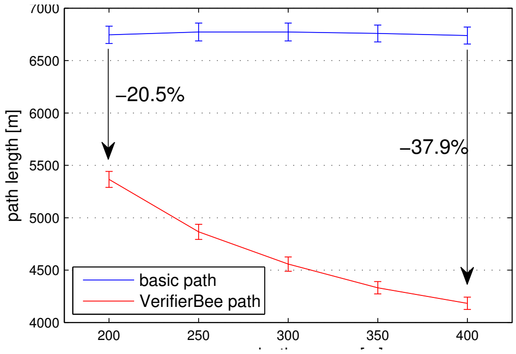


Figure 5.15: Average path length with the communication range with 20 nodes. Each point stems from 100 experiments. 95%-confidence intervals are displayed in error bars.

must be executed only once. The computed path remains valid as far as the positions of the nodes do not change. However, implementing the algorithm in C language (instead of Matlab language) should improve the performances even more.

We analyze now the influence of the parameters on the path length. Figure 5.15 shows the average path length for different values of the communication range with 20 nodes. As expected, the basic path is unable to leverage the full communication range. On the contrary, an improved communication range has a positive effect on the VerifierBee path (up to -37.9% of path length with a communication range of $400m$). Figure 5.16 shows the average path length for different drone control precisions with 20 nodes.
The path length clearly increases as the imprecision grows. This is because the drone has to pass farther away from the nodes, to be sure to enclose them in the verifiable triangle even in the worst-case control error. Veri-

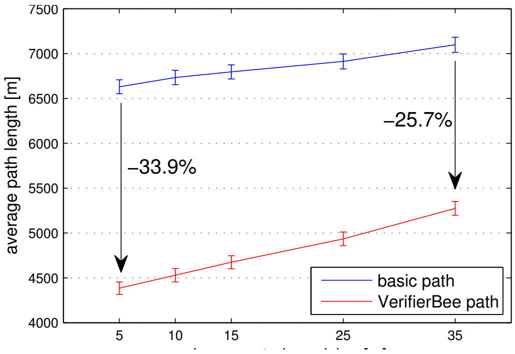


Figure 5.16: Average path length with the drone control precision with 20 nodes. Each point stems from 100 experiments. 95%-confidence intervals are displayed in error bars.

fierBee saves a roughly constant length for each value of the drone control precision.

5.7 Summary

Many dependable systems rely implicitly on the integrity of the positions of their components. In this dissertation, we explored the possibility to use the emerging drone technology in order to overcome the limitation of using several fixed anchors. In particular, our approach is to replace all the fixed anchors with a single drone that flies through a sequence of waypoints. At each waypoint, the drone “acts like” an anchor and securely verifies the positions of the devices. The main challenge here is to find a convenient path for the drone to do this. We present VerifierBee: a path planning algorithm that allows a drone to perform a secure location verification of a set of devices. VerifierBee finds a good approximation of the shortest path, and at the same time it respects a set of requirements about drone controllability, localization precision, and communication range. The results of our experiments showed that VerifierBee improves the path length of some 50% with respect to a simple solution.

Part III

Securing IoT Sensor Networks

Chapter 6

Security and Privacy Issues in IoT Devices

Internet of Things (IoT) is a new paradigm connecting billions of devices to the Internet. Falling prices and shrinking sizes of computer processors and sensors, coupled with connectivity, enable to create a global interconnection of devices which are addressable from anywhere in the world. IoT is being recognized by industry leaders and media as the next wave of innovation, disrupting the way we organize our daily life's. Currently, IoT devices are being deployed in households, industrial automation, and smart city infrastructure, most often being connected with the Internet [102]. This global interconnection of different *things* or IoT, can sense and interact with our physical environment and provide different services to help us be more productive with our daily tasks.

Many of the IoT devices are often embedded with processing, sensing, software and connectivity, enabling to communicate with other devices, and to be reached anytime from any where. However, the IoT devices are constrained in nature with limited battery and processing capabilities, therefore communication efficiency is a crucial need. IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) [56] is one such communication adaptation enabling to connect low-power and lossy wireless networks such as Wireless Sensor Networks (WSNs) to the Internet. According to Cisco [134] IoT is foreseen to connect over 50 billion low-power devices by end of 2020 communicating with each other and the possibility for secure efficient end-to-end communication among *things* is a high priority as IoT scales up. WSN enabled IoT will play a major role and most of these nodes will be equipped with non-rechargeable batteries, while being left unattended after deployment. The long term operation of the nodes with limited battery will require the WSN protocol stack to operate with minimum resources. Therefore, the security algorithms for WSNs will be designed with limited power, limited memory and limited battery life in mind [138]. IETF introduced Datagram Transport Layer Security (DTLS) [55] is becoming the *de-facto* security protocol for protecting end-to-end communications between two peers, in the presence of unreliable datagram protocols such as UDP [174, 75, 88, 109]. Furthermore, the lightweight Constrained Application Protocol (CoAP) [58], a specialized web transfer protocols for IoT, is adopting DTLS to secure the end-to-end communications among IoT client and server nodes.

Cisco and Ericsson in their individual assessments have estimated around 50 billion IoT connected devices in the world by 2020 [134, 90]. Given the great pace as IoT connected devices are growing, we are starting to experience the dark side of these connected devices. IoT devices are deployed in a large variety of devices throughout homes, businesses, hospitals and even entire cities, providing a widened attack surface for attackers offering a large number of entry points to build botnets of tens of thousands of compromised devices. These hacked devices are used as weapons in cyber attacks to bring down web services. Recent prominent attacks includes hosts such as OVH, Blizzard, Xbox, security bloggers and even the Rio Olympics website [72]. These botnets were capable of sending an extraordinary volume of traffic to a chosen target reaching a record highest of 700 Gbps [72], initiated from around 150 607 compromised Internet connected cameras and DVRs [103]. Given the vulnerable state of IoT devices, preparing counter measure to mitigate such attacks is an eminent priority. Figure 1 shows a typical IoT setup, which in reality can connect tens of thousands of innocuous IoT devices to the Internet.

Some researches have investigated the vulnerabilities in IoT networks and they have mainly focused on network layer and routing layer attacks. For example, Shahid et al. present SVELTE [156] where they present an intrusion detection system for IoT networks. However, their solution was evaluated against sinkhole and selective-forwarding attacks only, which are routing and network layer attacks. Prabhakaran et al. [99] present a work for DoS attack detection in 6LoWPAN based IoT networks. However, their solution is also not scalable beyond DoS attack detection. Our approach is different compared to previous work as our goal is to identify individually compromised nodes through the node's actions over encrypted traffic. According to Pa et al. [140], in their work IoTPOT, a honey-pot specifically targeting IoT devices, show many malware samples targeting IoT nodes. The captured malware samples are now available in the virus total. In our work, we con-

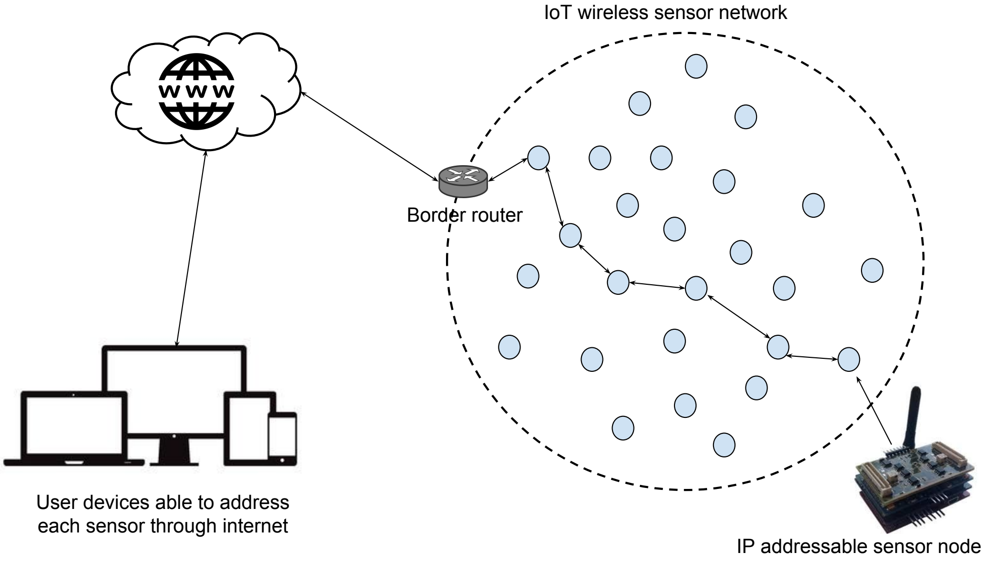

IoT wireless sensor network
WWW
Border router
User devices able to address each sensor through internet
IP addressable sensor node

Figure 6.1: An Example of an IoT wireless sensor network. Each sensor node can be addressed via the Internet.

sider a framework for identifying IoT node activities over encrypted traffic. After learning the normal node activities,this work can be easily extended for identifying any compromised nodes. However, detecting compromised node activities is beyond the scope of the present work.

In particular, this work focuses on understanding whether the node profiling done through analyzing encrypted traffic can be enhanced to understand exactly which actions the IoT node is doing. For example, we aim at identifying actions such as temperature reading, humidity reading, a routing message etc. The underlying issue we leverage in our work is that while DTLS protect the content of a packet, they do not prevent the detection of network packets patterns that instead reveal some sensitive information about the IoT node activities. Furthermore, we take advantage of specific properties IoT networks such as lack of mobility and relatively predictable traffic patterns [138] that allows for detection of activities of the nodes.

Our approach can be leveraged in several practical ways to infer activities of an IoT node. In the following, we report some possible scenarios:

- A forensic investigator may try to identify a compromised node which will be reporting different sensor readings than the expected ones. By comparing the begin actions of an IoT network against suspicious network activities it is possible to detect possible compromises. In this way the early responders can take required actions.

- By tracing the actions performed by any two nodes and taking to account it's negibouring nodes, and the communication latency, the adversary may guess (even if with some probability of error), whether a message was passed, where the message was originated, which type of message it relates to etc. Multiple observations on the network could further reduce the probability of errors.

- The network administrators can build a behavioral profile of their target IoT network to finger print the nodes in it along with their user activities. This will enable to infer the presence of any replicated nodes in the IoT network as this is a common threat against WSN nodes.

Contributions In this chapter, we propose a framework to infer particular actions of IoT nodes in a network. In particular, we assume that the traffic is encrypted and the adversary eavesdrops (without modifying them) the messages exchanged between the IoT nodes. To the best of our knowledge this is the first attempt to infer IoT node activities over encrypted traffic.

Our framework analyzes the network communications and leverages information available in CoAP/ DTLS packets (like IP addresses and ports), together with other information like the size, the direction (incoming/outgoing), and the timing. By using an approach based on machine learning, each node that is of interest is analyzed collectively along with the other network nodes. To set up our system, for each app we first pre-process a dataset of network packets labeled with the node actions that originated them, we cluster them in flow typologies that represent recurrent network flows, and finally we analyze them in order to create a training set that will be used to feed a classifier. The trained classifier will then be able to classify new traffic traces that have never been seen before. We run a thorough set of experiments to evaluate our solution. The results show that it can achieve accuracy and precision higher than 95%, for most of the considered actions.

We also discuss about the key idea underneath our traffic analysis approach for DTLS enabled IoT communications. In particular, we examine in depth the concept of network flow and the metric to evaluate the similarity between them. We also report details of the machine learning techniques we leverage in our method.

Organization The rest of this chapter is organized as follows. Section 6.1, presents the important related work in this area particularly focusing on activity detection over encrypted traffic. In Section 6.2 we introduce the traffic model and the constraints upon which our framework was built. Section 6.3, explains the machine learning background knowledge used in this study. In Section 6.4 we describe the implementation details of the IoT sensor nodes and the experimental setup. Section 6.5 introduces the evaluation criteria and the results.

6.1 Related Work

In this section we will summarize some important recent work in this area, which try to infer activities over encrypted traffic. There are many work in this domain and here we will summarize work relating to web traffic, smartphone and other application level encrypted traffic such as SSH or Skype. However, there is a research gap in work related to IoT encrypted traffic analysis.

Encrypted traffic cannot be analyzed directly, therefore statistical characteristics are used to infer useful information, which is also known as side-channel information leaks [183]. The authors of [116] have gathered encrypted communications to 2,000 web sites. Based on these profiles they propose two methods for inferring the source of the page retrieved under cover of an encrypted tunnel. The two methods that identify the traffic are naive Bayes classifier and Jaccards coefficient. Both their systems rely on packet lengths and discards the timing information. In [35], the authors are trying to report information leaks in several real world web applications. They utilize observable attributes from HTTPS traffic such as packet sizes and timing to find out various user selections on the web applications. Andriy et al. [141], focus on anonymous web browsers like Tor and JAP. They show anonymity can be broken under website finger printing attacks. They define features for website fingerprinting solely based on volume, time, and direction of the traffic. Justine et al. [166], presents BlindBox, a deep packet inspection system over encrypted traffic. In their work they propose a middle ground solution without violating privacy as in a middlebox [87], [96] and trying to preserve the end-to-end encryption. Middlebox will decrypt the traffic violating the end-to-end encryption. First they rely on exact string matching (e.g., watermarking, parental filtering etc) and if a stream is detected as suspicious they would decrypt the packet or flow and do a deep packet inspection. However, the above work have focused on applications which use standard encryption protocols which use standard encryption protocols which use standard encryption protocols like SSL/TLS. In particular, these protocols do not take into account resource constrained environments like IoT networks, therefore, these solutions cannot be applied directly to the IoT domain.

In [78], they describe a method to identify applications in IP networks, specifically looking at certain protocols. The different application protocols they have considered are ftp control, smtp, pop3, imap, https, http and ssh and have created signatures for encrypted protocols such as ssh or https using the initial handshake, which happens in clear. They were able to extract signatures for encrypted communication protocols as the initial handshake of ssh or https happens in clear. In order to determine which application a flow belongs to they inspect application layer information such as (TCP, UDP headers), an IP flow (protocol, srcIP, dstIP, srcPort, destPort) etc. They have considered the first n-Bytes of the traffic for generating the signatures. They have used Navie Bayes, AdaBoost, Regularized Maximum Entropy for signature building. Song et al. [169], analyzes the SSH traffic and in their work they reveal two vulnerabilities with SSH related to padding of packets and inter-activeness. Transmitted packets padded only to an eight-byte boundary. Interactive mode, where every keystroke user types is sent to a remote server in a separate IP packet immediately after the key press. They introduce, Herbivore, which try to learn users passwords by monitoring SSH sessions. They show nested SSH connections and their packet behavior as well. In their work, they have first described how the training data is collected and secondly show how the inter-key stroke timing can be given as a Gaussian distribution. Later they modeled the relationship of latency and character sequence as a Hidden Markov model. In [135], the authors analyze try to applications behind encrypted traffic and their method was able to detect Skype among other applications running. Wright et al. [188], in their work try to infer spoken phrases within a VoIP calls. If the same spoken word generates the same sequence of packets then the problem of identifying the instances of that word is reduced to a substring matching problem. However, since human speech exhibit high degree of variability. To handle this they have borrowed algorithms from speech recognition and bioinformatics communities. According to them hidden Markov models are useful when the training data itself exhibit high variability. Their work flow includes pronunciations from phoneme examples, which are encoded to VoIP packets, Synethesizing training data, training the HMM with Synethesized data and finally spotting the phrase in VoIP packets.

Conti et al. [42], [41] presents, apply the same identification problem into the domain of smartphones. In particular, they try to identify smartphone user actions over Android encrypted network traffic. They have considered user actions such as sending an email, reciving an email, browsing a profile on a social network, publishing a post or a tweet. The traffic features they have considered are: packet size and their order. Analyzing Android Encrypted Network Traffic to Identify User Actions. In their work they analyze encrypted network traffic to infer private information about the user actions such as sending an email, receiving an email, browsing a profile on a social network, publishing a post or a tweet. The traffic features they have considered are: packet size and their order.

According to the above related work, many work have been done in the domain of encrypted traffic analysis. However, these works are focused on domains such as web browsers, smartphone traffic and various applications (i.e., ssh, skype). In our work we focus on IoT encrypted traffic analysis and to the best of our knowledge our work is the first attempt to formalize the problem. In particular, we focus on DTLS encrypted traffic of low power IoT devices to identify their node activities.

6.2 Modeling IoT Node Activity Detection Over Encrypted Traffic

According to [138] the authors characterize the traffic in IoT sensor networks from a single node perspective and under different operation assumptions. In particular, they point out specific properties in IoT networks, such as lack of mobility and relatively predictable traffic patterns [138] that allows for detection of activities of the nodes. Taking into account these characteristics we start our initial investigations with DTLS encrypted client/server node pairs and formulate our problem and propose a solution. We built our IoT node's traffic model assuming the following:

- Sensors report periodically
- They communicate in a unicast manner
- The IP addresses are hard coded
- Each client node connects with the server periodically and the server responds with an identical and unique message (each message has a different length string) for each client/server communication
- One node performs only one activity

We will explain our implementation and the experimental setup in detail in Section 6.4. As mentioned before IETF is pushing DTLS to be the *de-facto* security protocol for protecting end-to-end communications between two IoT endpoints. DTLS being light weight and energy efficient provides security services such as preventing eavesdropping, tampering and message forgery, similar to the widely adopted TLS protocol for Internet. However, unlike other computing devices, IoT nodes are left unattended after deployment or even worse deployed in dangerous and hostile environments, therefore have higher risk of being compromised. A framework for detecting IoT activities will help detect if and when a node compromise happens. As a result of a compromise, two types of changes might happen with the nodes activities. They are,

1 The assigned activity of the node remains the same, however the values are modified

2 The assigned activity of the node changes and perform another activity, along side the expected activity or both

In the first type of compromise, an adversary is selectively modifying encrypted contents while an an external observer remains oblivious about any change in the node activities. This type of compromise detection is out of the scope of our work. In the second type of compromise which falls into the scope of this research, the node perform a different activity, i.e. sending a malicious attack traffic. For an external observer changes in the node activities are observable through modified network traffic values its traffic patterns. Our proposes method is analysing the DTLS encrypted network traffic to detect IoT nodes activities. Our method is using machine learning techniques to identify patterns of normal activities to train classifiers. We propose two classifiers for detecting IoT activities. The two classifiers have two different approaches: quickly identify the activities present in a traffic sample and the other classifier to further analyze the traffic sample for a given activity. For an example, lets assume we like to check how many nodes are performing their activities correctly after one year of deployment of the network. Assuming there were n unique nodes we will capture a traffic sample from the network. After one year several nodes might have run out of battery or got compromised hence the activity changed and as a result our model will detect only n_i activities where $i < n$. The two classifiers proposed are helpful in this scenario to identify the number of activities present. In particular, our N-class based classifier will output the activities it can detect and the binary classifier can be used to further verify the presence of a particular activity. Our models are independent of the network topology and is scalable to many nodes. When the activities of the nodes are unique our method is able to identify these activities and when the IoT network contain nodes with similar activities (e.g., multiple temperature and humidity nodes) our method limits to predict the activity class instead of a particular node.

6.3 Machine Learning Background

In this section, we briefly recall several machine learning techniques we used in our work, while we point the readers to appropriate references for a complete introduction on those topics. Being able to dynamically classify and identify the entities behind network traffic could help us with various network activities. There are advantages to identify information about encrypted traffic and from the point of view of network provider better classification and understanding of traffic allow better traffic shaping. Here we try to leverage knowledge to understand IoT node activities over encrypted traffic. We will call this problem as identification problem for identifying the actions behind the encrypted channels. In order to solve our problem we capitalize on the specific features of the IoT networks such as lack of mobility and relatively predictable traffic patterns [138] that allows for detection of activities of the nodes behind encrypted channels and in Section 6.2 we presented more discussion. In order to solve this identification problem first the network traffic has to be broken down into meaningful features and transform into a format which the machine learning algorithms can work with. We will organize this section under two subsections: Feature extraction and Machine Learning techniques.

6.3.1 Feature Extraction

We will discuss the features we used to break down the traffic into meaningful features to represent the IoT traffic. According to the literature [183, 116] observing the encrypted traffic we can make inferences based on statistical measures. Therefore, the features selected must reflect them well. The feature selection methods can be grouped under protocol or flow based methods and each method looks into patterns in the traffic such as packet arrival rate, the direction of traffic, packet sizes and the timing between the packets [135]. In our approach we used the later features: packet size and the timing between the packets. We extracted the packet size count over a given time window. This feature selection was influenced by [135] and it enable to implicitly encode the relations with the packet size and the time between the packets. To better understand lets consider a CCTV reporting a video stream and a temperature sensor reporting the values every five seconds. If we chose a packet count based method and selected the first 100 packets there is high probability we will loose the less frequent traffic from the temperature sensor and as a result the feature vector will not be able to capture all possible traffic samples. However, choosing a time window like 10 seconds according to the above discussed scenario allows to capture the traffic samples of all the applications in the feature vector. We generated a feature vector as a function of time and counted the number of packet sizes in the considered time window. Figure 2 presents the feature vector creation with respect to the traffic pattern.

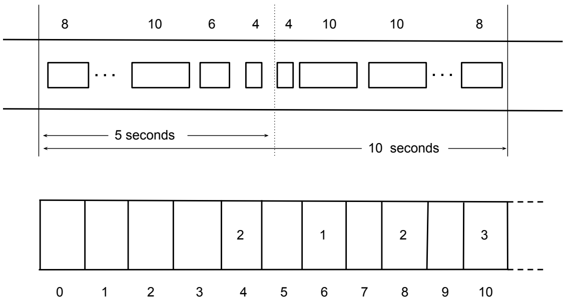


Figure 6.2: Features extracted by counting the packet sizes in a given time window. Selecting n first arriving packets will fail to represent most of the apps in the network. The feature vector enables to include timing and packet size relations implicitly.

6.3.2 Machine Learning Classifiers

In this subsection we will explain the machine learning classifiers. In our study we particularly make use of two classification approaches: N-class classification and binary class classification. We used the scikit-learn machine learning library implementations of the above classifiers.

N-class classification means a classification task with classifying the elements of a given set of data into more than two classes [85]. In our case classifying activities either as activity 1, 2, 3 or 4, which could be related to any activity such as a temperature, humidity, light measurement, etc. The predicted labels will belong to only one class, trying to associate with one activity.

Binary class classification is a classification task with classifying the elements of a given set of data into exactly two classes, normally identified as positive or negative classes [84]. In our case we will select one activity as a positive class and the rest as the negative class. The predicted labels will classify the set of data into two classes.

We used the Decision Tree algorithm as the underlying algorithm for the classifier, which learns the decision rules from the data features.

6.4 Implementation and Experimental Setting

In this section we will explain the experimental setting. We implement our IoT node prototypes on top of Contiki OS v2.7 [93] which is an open source operating system for the Internet of Things providing Internet communication for low-power wireless devices. The Contiki OS is supplied with an entire development environment called Instant Contiki. It includes different development tools like the Cooja network simulator [139] and Erbium REST Engine and CoAP implementation [110] which we use as a CoAP communication library. In order to make the communications secure, we used DTLS and we used the publicly available TinyDTLS 0.5 library [25] by Olaf Bergmann. It is a lightweigth DTLS implementation for constrained environments providing secure unicast communication. The TinyDTLS enables two nodes running the DTLS application perform the handshake and afterwards communicate securely with each other using negotiated security parameters. The ciphersuites supported by the implementation are `TLS_PSK_WITH_AES_128_CCM_8` and `TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8` where the first ciphersuite provides authenticated encryption based on AES in CBC mode and the second ciphersuite provides encryption based on elliptic curves. We used the first ciphersuite, which requires pre-shared keys to perform the handshake and derive session keying material. Finally, the library is included into Contiki OS as an application and referenced from the user program.

In our experimental scenario the IoT nodes compose unicast communications running as client/server pairs. The client acts as a CoAP client and a DTLS client and connects with the server periodically. We preconfigure the applications with a destination IP address and a port number in all the settings. The server acts as a CoAP server and a DTLS server as well and upon connecting with the client, the server will respond with a unique string. The experimental setup is shown in Figure 6.3.

The clients send a CoAP "hello" request of the GET type to their corresponding servers. The servers after processing the request, answer with a unique string message as the response. The client connects to the server using a CoAP blocking request and waits for its reply. The blocking request means that the client does not proceed to send the next request until it receives a reply for the current request. We configure the network settings as described under Section 6.2 and run all the subset configurations mentioned in Figure 3. For all these scenarios we capture the network communications

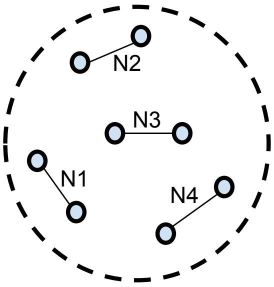


Figure 6.3: Experimental setup. The client/server nodes communicating with secure CoAP

and run the node activity detection algorithms. Table 6.1 summaries the experimental configurations.

Table 6.1: Summary of the experimental setup

Setting	Value
Wireless channel model	UDG model with Distance Loss
Communication range	5 m
Mote type	Wismote
Transport and network layer	UDP + DTLS + μIPv6 + 6LoWPAN
Application layer	CoAP
Radio interface	UDGM
Traffic pattern	Periodic (2*Clock_Seconds)
Topology	Client/Server
Activity 1	64 bytes message
Activity 2	22 bytes message
Activity 3	41 bytes message
Activity 4	109 bytes message
Total simulation time	7h

6.5 Evaluation and Discussion

In this section we will evaluate our proposed systems performance and discuss the suitably of our method for detecting IoT node activities over DTLS encrypted network traffic.

In order to evaluate our method we ran the experimental setup described in Section 6.4 and captured two sets of traffic samples for training and testing of our models. We used the first traffic sample for training our models and the second set of traffic samples for testing. Scikit learn machine learning libraries were used to build the models. In particular, we built two models: a N-class (or multi-class) classification model and a binary class classification model. We experimented with several classifiers to find the best fit for solving our problem. We tested with One Vs Rest, One vs One, Out put code and Decision tree algorithms for the multi-class and binary classification models. Decision tree classifier had better overall performance over our data and we built our model with this classifier. To build the multi-class classification model we used the traffic from all possible configurations (2^4) of all the nodes and tested with each configuration. For each configuration we used voting to predict the final activity and calculated the average performance for all the possible configurations. The final results of the multi-class classification is presented in the table 6.2. The N-class classifier enables to find the activities present in a given traffic sample. We build the binary classifier to check a particular activity against a given traffic sample. The activity we are interested represents the positive class and the rest represents the negative class and we trained our models with different examples from different configurations. We tested our model for its ability to differentiate a given activity from different activities and the results are reported in Table 6.4 and in Table 6.3.

We used the following evaluation metrics to measure the activity detection performance of our methods:

$$Precision = \frac{TP}{TP + FP} \times 100\%; \tag{6.1}$$

$$Recall = \frac{TP}{TP + FN} \times 100\%; \tag{6.2}$$

$$F - measure = \frac{2 \times Precision \times Recall}{Precision + Recall} \times 100\%; \tag{6.3}$$

where TP (true positive) and FP (false positive) are the numbers of correctly and incorrectly identified activities, respectively. FN (False negative) is the number of incorrectly identified activities as with respect to the considered activity. Precision is the fraction of retrieved instances that are relevant while recall (also called detection rate) is the fraction of relevant instances that are retrieved. F-measure considers both the precision and the recall of the test to compute the score. It can be considered as a weighted average of the precision and recall. The value of F-measure is large when both precision and recall are high, and small when either of them is poor.

Table 6.2: N-class model performance summary for detecting the activities of the nodes

Scenario	Precision	Recall	F1-Score
Activity 1	0.83	0.62	0.71
Activity 2	0.86	0.75	0.80
Activity 3	0.64	0.88	0.74
Activity 4	1.00	1.00	1.00
Average	0.83	0.81	0.81

According to the results presented in the tables above, our methods show promising results. Overall, the binary classification methods outperforms the N-class classification methods in the ability to detect activities. This could be due to having many possible out comes in the predictions, compared with the two possibilities in binary classification. Table 6.2 summarizes the average detection rate of the four activities in a given traffic sample. According to the results, our model can correctly identify the presence of activity four, however other activity detection varies in performance. However, in order to overcome this limitation we propose our second model based on binary classification. According to Table 6.3 detecting activity 1 among other activities has significantly improved. Similarly Table 6.4 tires to summarize the results of the ability to predict if an activity is not present among other activities, which is important to verify as this might indicate a compromised node (as a result the activities are different now) in the network. Our models will enable a network forensic investigator to perform network audits. For example, comparing a two network traffic samples captured in two consecutive years may yield some anomalies compared to the previous year. This could result due to compromised nodes and the expected activities are no longer present in the traffic. Our work is the first step in this direction, proposing activity detection models to predict IoT node activities behind DTLS encrypted traffic and our initial investigations reveal promising results.

6.6 Summary

In the era of the Internet of Things (IoT), many digitally connected devices are surrounding us in our day-to-day lives. In the near future, we expect 50 billion connected devices everywhere, from smart cities, factories to our

Table 6.3: Binary classification model performance for differentiating a specific activity from a given activity

Activities	Activity 1			Activity 2			Activity 3			Activity 4		
	P	R	F	P	R	F	P	R	F	P	R	F
Activity 1	-	-	-	0.99	1.00	0.99	1.00	1.00	1.00	1.00	0.99	0.99
Activity 2	1.00	0.99	0.99	-	-	-	0.66	0.77	0.71	0.79	0.88	0.83
Activity 3	1.00	1.00	1.00	0.71	0.58	0.64	-	-	-	0.85	0.85	0.85
Activity 4	0.99	1.00	0.99	0.86	0.76	0.81	0.85	0.85	0.85	-	-	-

Here we have considered the ability of our model to separate between two given activities.

Table 6.4: Binary classification model performance summary for detecting a specific activity among a given traffic sample

Scenario	Precision	Recall	F1-Score
Activity 1 among A. 2,3,4	0.94	1.00	0.97
Activity 2 among A. 1,3,4	1.00	1.00	1.00
Activity 3 among A. 1,2,4	1.00	1.00	1.00
Activity 4 among A. 1,2,3	1.00	1.00	1.00
Average	0.98	1.00	0.99

homes and even on our bodies. The advent of technologies like IPv6 and the wide deployment of WiFi networks is enabling the growth of IoT. However, many IoT devices are lacking proper security methods for safe guarding the individual nodes and these vulnerability may open up a wider attack surface for much larger attacks. In this work we propose a proof of concept framework to detect IoT node actions by observing the encrypted communication traffic. We passively observe the wireless network traffic of the nodes and try to detect the activities. The fact that the network traffic is often encrypted makes the task even more challenging. In this chapter we investigate to what extent IoT node actions can be identified by passively observing the encrypted network traffic. We designed a framework for achieving the above goal using advanced machine learning techniques. We built a complete implementation of this framework using Contiki OS, the widely used open source operating system (OS) for IoT devices. Furthermore, we run a through set of experiments using the Cooja simulator (which runs deployable Contiki code), which show that our framework can achieve high accuracy and precision for most of the considered actions. To the best of our knowledge this is the first framework for detecting IoT node actions over encrypted traffic.

Chapter 7

Conclusions

The ever increasing demand of connectivity and services that rely on distributed heterogeneous devices is populating the world with billions of devices. They are expected to generate millions of job opportunities, and trillions of dollars in economic growth and cost savings. Some key areas where these devices will be widely used is in preventive health care, enhancing public safety, patient monitoring, smart manufacturing and power generation, and seamless home and municipality infrastructure improvements [173]. Unfortunately, the increased penetration of these devices around us, is also opening new attack surfaces for cyber-criminals. In order to reap the full benefits of this emerging technological eco-system, it must be well protected against new cyber-attacks. This dissertation presents our research efforts devoted to emerging security and privacy challenges; particularly focusing on three heterogeneous devices in this technological eco-system. In the following, we summarize our contributions (Section 7.1) and discuss future research directions (Section 7.2).

7.1 Summary of Contribution

This dissertation presents our research efforts under three parts, focusing on three heterogeneous devices. In the following, we will summarize the contributions under each part.

7.1.1 Smartphone Security and Privacy Issues

In Part I of the dissertation we focused on smartphone related security and privacy issues. Given the ubiquitous nature of the smartphones, users are always carrying the smartphones with them and hence accumulate a lot of personal data. Furthermore, with the increasing capabilities with smartphones (e.g., processing, storage, connectivity) there are many applications which are built around smartphones providing value added services for the users. For example, online banking, Internet access and interfacing with third party IoT devices (e.g., fitness trackers, thermostats, etc). Smartphone becoming an epicenter of daily life poses many security challengers and side channel privacy leaks through its communication traffic. For example, the communication pattern might reveal user habits or de-anonymize the end-users resulting in a privacy leak. In this part we tried to address these main issues and tired to help improve the usability of the smartphone:

- *Android Malware Detection Through Network Analysis:* In this study we implemented a Network based malware detection framework to detect the rising Android malware. In particular we proposed two methods to detect maliciously behaving apps through the apps network characteristics. First the network traffic of the smartphone was converted into a set of state sequences to represent the network behavior of an app. These state sequences can also be characterized as strings if we consider each state as a single character. Next, we propsed two methods based on Markov chain and string kernel to build malware detection models. These models take state sequences as the input to learn how to recognize malware. Finally, the obtained models are used to predict the label for every new application. In particular, our methods leveraged: i) retaining all important features of network communications (without filtering traffic), ii) modeling the data as state sequences that can be used in many machine learning techniques, iii) use of efficient, widely applied classifiers to process the state sequences as the input, iv) the employment of various malware families to allow our models to learn a generalized network behavior.

- *Preserving anonymous end-to-end communications in adversarial mobile clouds:* We presented a protocol for anonymous end-to-end communications among users in a mobile cloud environment, where the cloud clones handle part of the communication towards destination. The attack model considered is unprecedented. It includes devices, network operators, and the cloud provider behaving as malicious entities, and the possibility of all of them to collude. In this scenario, we built a delay-tolerant solution that provably guarantees (α, β)- anonymity, and evaluated its performance on a real-life testbed.

- *CaT- Cloud aided TLS:* In this work we investigated the TLS protocols cryptographic operational cost on a smartphone. Smartphones are widely used for operations like accessing email, e-banking, web browsing, which most of the time are encrypted communication. As the number of encrypted communications are increasing, the number of cryptographic operations are also increasing proportionately. However, crypto operations such as public key crypto stress the CPU and hence is energy intensive. We extensively measured the energy consumption of TLS full handshake and session-resume. Full handshake use asymetric crypto for establishing session keys and session-resume reuse the previously negotiated keys. We noticed a significant reduction in the energy consumption for session resumption. Furthermore, we evaluated whether offloading the asymmetric key exchange of the TLS protocol to a cloud instance could bring some benefit from the mobile device point of view. In particular, we designed a possible solution (named CaT) for offloading the TLS asymmetric key exchange, and conducted an extensive experimental analysis. The results show that CaT is actually less energy-efficient than performing the normal full TLS handshake on a smartphone. We identified that CaT consumes less power as the computation was offloaded, however required more time to finish the operation with respect to the standard approach, hence as the net result ended up consuming more energy.

7.1.2 Drone Application for Secure Localization of Distributed Devices

In Part II, we presented the problem of secure localization in distributed systems. The dependability of many of these systems rely on the integrity of the position information of the distributed sensing devices. An adversary capable of changing the position of one or more devices (*displacement attack*) can deeply affect the system behavior with little effort, yet it will be difficult to detect. In our work, we explored the approach of using the emerging drone technology to securely verify a set of positions. We formally stated the Traveller Location Verifier Problem (TLVP), with regard to finding the shortest path for a drone to securely verify a set of nodes by means of verifiable multilateration. We proposed *VerifierBee*, a path planning algorithm that finds an approximate solution to TLVP. The results of our experiments showed that VerifierBee improves the path length of some 50% with respect to a simple solution.

7.1.3 Securing IoT Sensor Networks

In Part III we introduced a framework for detecting IoT node activities over DTLS encrypted traffic. DTLS is expected to be the *de-facto* standard for securing low-power IoT devices. To the best of our knowledge, our work is the first attempt with an approach to detect IoT node activities. Furthermore, an essential goal also is to fill a research gap in activity detection of constrained networks. Our framework observers the network traffic samples and predict the node activities present in the traffic. Our proposed framework leverages: (1) the relatively predictable traffic patterns of IoT sensor nodes, (2) extract important features which reflect the node activities and model the feature vectors to be used by machine learning techniques, (3) use of efficient, widely applied classifiers to process the features and infer node activities present in a traffic sample. Our evaluation showed that our framework is able to detect most of the considered activities with high precision.

7.2 Future Work

The work presented in this dissertation represents an initial effort. We now discuss possible future developments that naturally follow form the research contributions presented in this dissertation.

7.2.1 Smartphone Security and Privacy Issues

Smartphones are constantly being exposed to new and sophisticated security threats. It is always a battle between the malware authors and the security providers. In our work we tried to address some of the issues related to security and privacy as well as enhancing the overall efficiency of the smartphone user experience, however, we find the following challenges and some limitations to be addressed.

The first step towards proper protection against smartphone malware is a proper mindset from the users. Users must be educated to stick to official app markets (e.g., Google Play), enabling basic security settings (e.g., enabling app verification before install, using a password) and keeping the smartphone upto date by installing the relevant software updates. Furthermore, better user habits such as not connecting with open and suspicious WiFi hotspots. Therefore, making the smartphone users aware about the security and privacy issues is an important step in securing this eco-system.

However, to face the threats of complicated and social engineered attacks of malware the detection algorithms need to be improved as well. In the methods we proposed in Chapter 2 we identify the following: Improving the Markov-chain based method and gram matrix in the string kernel based method. In particular, we plan to investigate the possibility of choosing the most representative strings (the longest or the two longest) instead of using the whole set of strings. For the gram matrix in string kernel-based method, we aim to introduce a new string kernel, that allows to appropriately compute similarities between strings in this context. Generally we see further room for improvement in the detection algorithms. Also, we identify the need to compare our proposed methods with existing methods to compare the efficiency.

For the privacy solution proposed in Chapter 3, we would like to consider that the files exchanged might contain information on sender/receiver and also to depart from the assumption of having standalone app available for each user in the Internet. Finally, we would like to provide a formal security analysis of our protocol using automatic cryptographic protocol verifiers, such as ProVerif.

7.2.2 Drone Application for Secure Localization of Distributed Devices

Displacement attack is a major problem on distributed dependable systems and verifiable multilateration is a well accepted [31] approach by the research community for securing the positions. In our work we proposed a drone based secure localization approach, which is more cost-effective (by eliminating fixed anchor nodes) however, it opens many challenges still to be addressed. Currently, our work presents a path planning algorithm to securely verify a set of positions. Our proposed solution, VerifierBee finds an approximate solution to TLVP. However, there still exists a gap in moving from path planning to mission planning. We identify filling this research gap will enable to perform verifiable multilateration more cost-effectively using the emerging drone technology.

7.2.3 Securing IoT devices: Activity detection of IoT devices

The IoT activity detection framework we presented in Chapter 6 presents an initial effort and has some limitations that open new research challenges for future in terms of the framework and the underlying protocols.

First, we identify the need to validate our system on real IoT sensor nodes, to better fine tun the system. However, in order to fully test in a real life sensor deployment the existing protocols also need to improve. For example, the existing implementations for multicast group communications for DTLS needs to support all the features discussed in [174] for secure multicast group communication. The current publicly available tinygroupdtls library [136] by Kirill Nikitin provides an open source extension for Contiki OS, however, the operation of this library is limited to a single hop. We are aware the framework we have proposed is an initial step towards detecting compromised nodes in an IoT network, partially due to the above mentioned protocol level limitations. However, to the best of our knowledge our work is the first attempt in this domain and we see lot of potential for improvements. Our framework has the potential to be extended as an intrusion detection system or as a forensic investigation framework depending on the operation mode (online vs. offline). With the growing cyber threats [184, 137], protecting the IoT echo system needs comprehensive efforts in the development of the underlying protocols and detection frameworks like ours. More generally, it is expected to have around 50 billion connected devices by 2020 and in order to protect these devices: i) the devices need secure and light weight protocols, and ii) detection tools which can be automated and scalable for large number of devices is needed. Our work provides an initial step towards this direction and as future work we plan to improve our framework and provide comparisons.

Bibliography

[1] Mozilla firefox root certificate authorities. http://developer.android.com/reference/android/os/Process.html. [Online; accessed September 2016].

[2] Rfc 2616. https://tools.ietf.org/html/rfc2616. [Online; accessed September 2016].

[3] Rfc 5077. https://tools.ietf.org/html/rfc5077. [Online; accessed September 2016].

[4] Rfc 5246. https://tools.ietf.org/html/rfc5246. [Online; accessed September 2016].

[5] tpacketcapture. https://play.google.com/store/apps/details?id=jp.co.taosoftware.android.packetcapture\&hl=en. [Online; accessed September 2016].

[6] Yousra Aafer, Wenliang Du, and Heng Yin. Droidapiminer: Mining api-level features for robust malware detection in android. In *Proceedings of the International Conference on Security and Privacy in Communication Systems*, pages 86–103. Springer, 2013.

[7] Channel News Aisa. Mail sent by drone in world-first singpost trial. http://www.channelnewsasia.com/news/business/mail-sent-to-pulau-ubin/2177406.html. [Online; accessed October 2016].

[8] M. Alizadeh, S. Abolfazli, M. Zamani, S. Baharun, and K. Sakurai. Authentication in mobile cloud computing: A survey. *Journal of Network and Computer Applications*, 61:59–80, 2015.

[9] Brandon Amos, Hamilton Turner, and Jules White. Applying machine learning classifiers to dynamic android malware detection at scale. In

Proceedings of the 9th international wireless communications and mobile computing conference (IWCMC), pages 1666–1671. IEEE, 2013.

[10] George Apostolopoulos, Vinod Peris, Prashant Pradhan, and Debanjan Saha. Securing electronic commerce: reducing the ssl overhead. *IEEE Network*, 14(4):8–16, 2000.

[11] George Apostolopoulos, Vinod Peris, and Debanjan Saha. Transport layer security: How much does it really cost? In *Proceedings of the 8th IEEE INFOCOM'99*, pages 717–725. IEEE, 1999.

[12] Axelle Apvrille and Tim Strazzere. Reducing the window of opportunity for android malware gotta catch them all. *Journal in Computer Virology*, 8(1-2):61–71, 2012.

[13] Claudio A Ardagna, Mauro Conti, Mario Leone, and Julinda Stefa. An anonymous end-to-end communication protocol for mobile cloud environments. *IEEE Transactions on Services Computing*, 7(3):373–386, 2014.

[14] Claudio A Ardagna, Sushil Jajodia, Pierangela Samarati, and Angelos Stavrou. Providing usersâĂŹ anonymity in mobile hybrid networks. *ACM Transactions on Internet Technology (TOIT)*, 12(3):7, 2013.

[15] Orlando Arias, Jacob Wurm, Khoa Hoang, and Yier Jin. Privacy and security in internet of things and wearable devices. *IEEE Transactions on Multi-Scale Computing Systems*, 1(2):99–109, 2015.

[16] Kanishka Ariyapala, Hoang Giang Do, Huynh Ngoc Anh, Wee Keong Ng, and Mauro Conti. A host and network based intrusion detection for android smartphones. In *Proceedings of the 30th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, pages 849–854. IEEE, 2016.

[17] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, and Konrad Rieck. Drebin: Effective and explainable detection of android malware in your pocket. In *Proceedings of the 20th Annual Network & Distributed System Security Symposium (NDSS)*, 2014.

[18] Ines Ayadi and Simoni Noëmie. Adaptive provisioning of connectivity-as-a-service for mobile cloud computing. In *Proceedings of the 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, pages 169–175. IEEE, 2014.

[19] Marco V Barbera, Sokol Kosta, Alessandro Mei, Vasile C Perta, and Julinda Stefa. Cdroid: towards a cloud-integrated mobile operating system. In *Proceedings of IEEE INFOCOM'13*, pages 47–48. IEEE, 2013.

[20] Marco V Barbera, Sokol Kosta, Alessandro Mei, Vasile C Perta, and Julinda Stefa. Mobile offloading in the wild: Findings and lessons learned through a real-life experiment with a new cloud-aware system. In *Proceedings of the IEEE INFOCOM'14*, pages 2355–2363. IEEE, 2014.

[21] Marco V Barbera, Sokol Kosta, Alessandro Mei, and Julinda Stefa. To offload or not to offload? the bandwidth and energy costs of mobile cloud computing. In *Proceedings of IEEE INFOCOM'13*, pages 1285–1293. IEEE, 2013.

[22] Marco V Barbera, Sokol Kosta, Julinda Stefa, Pan Hui, and Alessandro Mei. Cloudshield: Efficient anti-malware smartphone patching with a p2p network on the cloud. In *Proceedings of the 12th IEEE International Conference on Peer-to-Peer Computing (P2P)*, pages 50–56. IEEE, 2012.

[23] Ulrich Bayer, Paolo Milani Comparetti, Clemens Hlauschek, Christopher Kruegel, and Engin Kirda. Scalable, behavior-based malware clustering. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, pages 8–11, 2009.

[24] Diana Berbecaru. On measuring SSL-based secure data transfer with handheld devices. In *Proceedings of the 2nd International Symposium on Wireless Communication Systems*, pages 409–413. IEEE, 2005.

[25] Olaf Bergmann. TinyDTLS. https://projects.eclipse.org/proposals/tinydtls/. [Online; accessed October 2016].

[26] Deepak Bhadauria, Onur Tekdas, and Volkan Isler. Robotic data mules for collecting data over sparse sensor fields. *Journal of Field Robotics*, 28(3):388–404, 2011.

[27] Gustaf Bjorksten. Defending users at risk from ddos attacks: An evolving challenge. https://www.accessnow.org/defending-users-at-risk-from-ddos-attacks-an-evolving-challenge/. [Online; accessed October 2016].

[28] Stefan Brands and David Chaum. Distance-bounding protocols. In *Proceedings of the Workshop on the Theory and Application of of Cryptographic Techniques*, pages 344–359. Springer, 1993.

[29] Iker Burguera, Urko Zurutuza, and Simin Nadjm-Tehrani. Crowdroid: behavior-based malware detection system for android. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, pages 15–26. ACM, 2011.

[30] Fernando Caballero, Luis Merino, P Gil, Ivan Maza, and Aníbal Ollero. A probabilistic framework for entire WSN localization using a mobile robot. *Robotics and Autonomous Systems*, 56(10):798–806, 2008.

[31] Srdjan Capkun and J-P Hubaux. Secure positioning of wireless devices with application to sensor networks. In *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, pages 1917–1928. IEEE, 2005.

[32] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[33] Chien-Yen Chang, Belinda Lange, Mi Zhang, Sebastian Koenig, Phil Requejo, Noom Somboon, Alexander A Sawchuk, and Albert A Rizzo. Towards pervasive physical rehabilitation using microsoft kinect. In *Proceedings of the 6th International Conference on Pervasive Computing Technologies for Healthcare (PervasiveHealth)*, pages 159–162. IEEE, 2012.

[34] David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[35] Shuo Chen, Rui Wang, XiaoFeng Wang, and Kehuan Zhang. Side-channel leaks in web applications: A reality today, a challenge tomorrow. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P'10)*, pages 191–206. IEEE, 2010.

[36] Yang Chen, Mo Ghorbanzadeh, Kevin Ma, Charles Clancy, and Robert McGwier. A hidden markov model detection of malicious android applications at runtime. In *Proceedings of the 23rd Wireless and Optical Communication Conference (WOCC)*, pages 1–6. IEEE, 2014.

[37] Zhenxiang Chen, Hongbo Han, Qiben Yan, Bo Yang, Lizhi Peng, Lei Zhang, and Jin Li. A first look at android malware traffic in first

few minutes. In *Proceddings of the IEEE Trustcom/BigDataSE/ISPA*, pages 206–213. IEEE, 2015.

[38] Bill Childers. Hacking the parrot AR drone. *Linux Journal*, 2014(241):1, 2014.

[39] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems*, pages 301–314. ACM, 2011.

[40] Cristian Coarfa, Peter Druschel, and Dan S Wallach. Performance analysis of TLS web servers. *ACM Transactions on Computer Systems (TOCS)*, 24(1):39–69, 2006.

[41] Mauro Conti, Luigi V Mancini, Riccardo Spolaor, and Nino Vincenzo Verde. Can't you hear me knocking: Identification of user actions on android apps via traffic analysis. In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, pages 297–304. ACM, 2015.

[42] Mauro Conti, Luigi Vincenzo Mancini, Riccardo Spolaor, and Nino Vincenzo Verde. Analyzing android encrypted network traffic to identify user actions. *IEEE Transactions On Information Forensics and Security*, 11(1):114–125, 2016.

[43] Mauro Conti, Vu Thien Nga Nguyen, and Bruno Crispo. CRePE: Context-related policy enforcement for Android. In *Proceedings of the International Conference on Information Security*, pages 331–345. Springer, 2010.

[44] Peter Corke, Ron Peterson, and Daniela Rus. Coordinating aerial robots and sensor networks for localization and navigation. In *Proceedings of the Distributed Autonomous Robotic Systems*, pages 295–304. Springer, 2007.

[45] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[46] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 49–62. ACM, 2010.

[47] Aron Culotta and Jeffrey Sorensen. Dependency tree kernels for relation extraction. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, pages 423–430. Association for Computational Linguistics, 2004.

[48] P Dang, FL Lewis, and DO Popa. Dynamic localization of air-ground wireless sensor networks. In *Proceedings of the Advances in Unmanned Aerial Vehicles*, pages 431–453. Springer, 2007.

[49] Jörg Daubert, Leon Böck, Panayotis Kikirasy, Max Mühlhäuser, and Mathias Fischer. Twitterize: Anonymous micro-blogging. In *Proceedings of the 11th IEEE/ACS International Conference on Computer Systems and Applications (AICCSA)*, pages 817–823. IEEE, 2014.

[50] DecaWave. ScenSor SWM1000 Module. http://www.decawave.com/products/dwm1000-module.

[51] Gianluca Dini, Fabio Martinelli, Andrea Saracino, and Daniele Sgandurra. Madam: a multi-level anomaly detector for android malware. In *International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security*, pages 240–253. Springer, 2012.

[52] Egham. Worldwide device shipments by segment. http://m0droid.netai.net/modroid/. [Online; accessed September 2016].

[53] Steven Englehardt, Dillon Reisman, Christian Eubank, Peter Zimmerman, Jonathan Mayer, Arvind Narayanan, and Edward W Felten. Cookies that give you away: The surveillance implications of web tracking. In *Proceedings of the 24th International Conference on World Wide Web*, pages 289–299. ACM, 2015.

[54] Halit Ergezer and Kemal Leblebicioglu. Path planning for uavs for maximum information collection. *IEEE Transactions on Aerospace and Electronic Systems*, 49(1):502–520, 2013.

[55] E. Rescorla et al. Datagram transport layer security version 1.2. https://tools.ietf.org/html/rfc6347. [Online; accessed October 2016].

[56] G. Montenegro et al. Transmission of ipv6 packets over ieee networks. https://tools.ietf.org/html/rfc4944. [Online; accessed October 2016].

[57] Osamah Rawashdeh et al. Aerial-surface-underwater reconnaissance drone. https://dronesforgood.ae/finals/

aerial-surface-underwater-reconnaissance-drone. [Online; accessed October 2016].

[58] Z. Shelby et al. The constrained application protocol (coap). https://tools.ietf.org/html/rfc7252. [Online; accessed October 2016].

[59] Adrian Mettler et al. for FireEy. Ssl vulnerabilities. https://www.fireeye.com/blog/threat-research/2014/08/ssl-vulnerabilities-who-listens-when-android-applications-talk.html. [Online; accessed September 2016].

[60] Ramsey M Faragher, Oliver RA Chick, Daniel T Wagner, Timothy Goh, James Snee, and Brian Jones. Captain buzz: An all-smartphone autonomous delta-wing drone. In *Proceedings of the First Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use*, pages 27–32. ACM, 2015.

[61] Adrienne Porter Felt, Serge Egelman, and David Wagner. I've got 99 problems, but vibration ain't one: a survey of smartphone users' concerns. In *Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices*, pages 33–44. ACM, 2012.

[62] Yu Feng, Saswat Anand, Isil Dillig, and Alex Aiken. Apposcopy: Semantics-based detection of android malware through static analysis. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 576–587. ACM, 2014.

[63] Hossein Fereidooni, Veelasha Moonsamy, Mauro Conti, and Lejla Batina. Efficient classification of android malware in the wild using robust static features. *Protecting Mobile Networks and Devices: Challenges and Solutions*, 1:181–209, 2016.

[64] Kendra Moyer for APCNews. Attacks on social movements increase online, tech support comes to the rescue. https://www.apc.org/en/news/attacks-social-movements-increase-online-tech-supp. [Online; accessed October 2016].

[65] Mark Prigg for MailOnline. The ambulance drone that could save your life: Flying defibrillator can reach speeds of 60mph. http://www.dailymail.co.uk/sciencetech/article-2811851/The-ambulance-drone-save-life-Flying-defibrillator-reach-speeds-60mph.html. [Online; accessed October 2016].

[66] Ernst Wittmann for memeburn. The internet of things is here, and it will revolve around the smartphone. http://memeburn.com/2015/12/the-internet-of-things-is-here-and-it-will-revolve-around-the-smartphone/. [Online; accessed October 2016].

[67] Tim Greene for Network World. Largest ddos attack ever delivered by botnet of hijacked iot devices. http://www.networkworld.com/article/3123672/security/largest-ddos-attack-ever-delivered-by-botnet-of-hijacked-iot-devices.html. [Online; accessed October 2016].

[68] Ingrid Lunden for Tech Crunch. 6.1 billion smartphone users globally by 2020, overtaking basic fixed phone subscriptions. https://goo.gl/rwabC2. [Online; accessed October 2016].

[69] Rina Marie Doctor for Tech Times. Faa predicts drones will number 7 million by 2020. http://www.techtimes.com/articles/144405/20160326/faa-predicts-drones-will-number-7-million-by-2020.htm. [Online; accessed October 2016].

[70] Micah Lee for The Intercept. Secret badass intelligence program spied on smartphones. https://theintercept.com/2015/01/26/secret-badass-spy-program/. [Online; accessed October 2016].

[71] Dean Takahashi for Venturebeat. The app economy could double to usd101 billion by 2020. http://venturebeat.com/2016/02/10/the-app-economy-could-double-to-101b-by-2020-research-firm-says/. [Online; accessed October 2016].

[72] Thomas Fox-Brewste. How hacked cameras are helping launch the biggest attacks the internet has ever seen. https://goo.gl/WOSs0V. [Online; accessed October 2016].

[73] Sebastián García, Vojtěch Uhlíř, and Martin Rehak. Identifying and modeling botnet c&c behaviors. In *Proceedings of the 1st International Workshop on Agents and CyberSecurity*, pages 1–8. ACM, 2014.

[74] Sebastián García, Alejandro Zunino, and Marcelo Campo. Survey on network-based botnet detection methods. *Security and Communication Networks*, 7(5):878–903, 2014.

[75] Oscar Garcia-Morchon, Sye Loong Keoh, Sandeep Kumar, Pedro Moreno-Sanchez, Francisco Vidal-Meca, and Jan Henrik Ziegeldorf. Securing the ip-based internet of things with hip and dtls. In *Proceedings*

of the sixth ACM conference on Security and privacy in wireless and mobile networks, pages 119–124. ACM, 2013.

[76] Chad Goerzen, Zhaodan Kong, and Bernard Mettler. A survey of motion planning algorithms from the perspective of autonomous UAV guidance. *Journal of Intelligent and Robotic Systems*, 57(1-4):65–100, 2010.

[77] Michael Grace, Yajin Zhou, Qiang Zhang, Shihong Zou, and Xuxian Jiang. Riskranker: scalable and accurate zero-day android malware detection. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pages 281–294. ACM, 2012.

[78] Patrick Haffner, Subhabrata Sen, Oliver Spatscheck, and Dongmei Wang. ACAS: automated construction of application signatures. In *Proceedings of the 2005 ACM SIGCOMM workshop on Mining network data*, pages 197–202. ACM, 2005.

[79] Creighton TR Hager, Scott F Midkiff, J-M Park, and Thomas L Martin. Performance and energy efficiency of block ciphers in personal digital assistants. In *Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications*, pages 127–136. IEEE, 2005.

[80] Jiyong Han, Minkeun Ha, and Daeyoung Kim. Practical security analysis for the constrained node networks: Focusing on the DTLS protocol. In *Proceedings of the 5th International Conference on the Internet of Things (IOT)*, pages 22–29. IEEE, 2015.

[81] Guillermo Heredia, Fernando Caballero, Iván Maza, Luis Merino, Antidio Viguria, and Aníbal Ollero. Multi-unmanned aerial vehicle (UAV) cooperative fault detection employing differential global positioning (DGPS), inertial and vision sensors. *Sensors*, 9(9):7566–7579, 2009.

[82] Tsung-Hsuan Ho, Daniel Dean, Xiaohui Gu, and William Enck. PREC: practical root exploit containment for android devices. In *Proceedings of the 4th ACM conference on Data and application security and privacy*, pages 187–198. ACM, 2014.

[83] Geoffrey A Hollinger, Sunav Choudhary, Parastoo Qarabaqi, Christopher Murphy, Urbashi Mitra, Gaurav S Sukhatme, Milica Stojanovic, Hanumant Singh, and Franz Hover. Underwater data collection using robotic sensor networks. *IEEE Journal on Selected Areas in Communications*, 30(5):899–911, 2012.

[84] Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, et al. A practical guide to support vector classification. 2003.

[85] Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multiclass support vector machines. *IEEE transactions on Neural Networks*, 13(2):415–425, 2002.

[86] Yih-Chun Hu, Adrian Perrig, and David B Johnson. Packet leashes: a defense against wormhole attacks in wireless networks. In *Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. INFOCOM'03*, pages 1976–1986. IEEE, 2003.

[87] Lin Shung Huang, Alex Rice, Erling Ellingsen, and Collin Jackson. Analyzing forged SSL certificates in the wild. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P'14)*, pages 83–97. IEEE, 2014.

[88] René Hummen, Hanno Wirtz, Jan Henrik Ziegeldorf, Jens Hiller, and Klaus Wehrle. Tailoring end-to-end ip security protocols to the internet of things. In *Proceedings of the 21st IEEE International Conference on Network Protocols (ICNP)*, pages 1–10. IEEE, 2013.

[89] IEEE Computer Society. IEEE Std 802.15.4a-2007 (Amendment 1: Add Alternate PHYs), 2007.

[90] Ericsson Inc. Connected devices. https://www.ericsson.com/openarticle/mwc-connected-devices_1686565587_c. [Online; accessed October 2016].

[91] Google Inc. Google glass. https://developers.google.com/glass/. [Online; accessed October 2016].

[92] Sophos Inc. Transmission of ipv6 packets over ieee networks. https://www.sophos.com/en-us/medialibrary/PDFs/other/sophos-security-threat-report-2014.pdf. [Online; accessed October 2016].

[93] Thingsquare Inc. Contiki: The open source os for the internet of things. http://www.contiki-os.org/. [Online; accessed October 2016].

[94] Javier Irizarry, Masoud Gheisari, and Bruce N Walker. Usability assessment of drone technology as safety inspection tools. *Journal of Information Technology in Construction*, 17:194–212, 2012.

[95] David Jaramillo, Neil Katz, Bill Bodin, William Tworek, Robert Smart, and Thomas Cook. Cooperative solutions for bring your own device (BYOD). *IBM journal of research and development*, 57(6):5–1, 2013.

[96] Jeff Jarmoc and DSCT Unit. SSL/TLS interception proxies and transitive trust. *Presentation at Black Hat Europe*, 2012.

[97] Somesh Jha, Kymie MC Tan, and Roy A Maxion. Markov chains, classifiers, and intrusion detection. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop*, pages 206–209. IEEE, 2001.

[98] Jinfang Jiang, Guangjie Han, Chuan Zhu, Yuhui Dong, and Na Zhang. Secure localization in wireless sensor networks: A survey. *Journal of Communications*, 6(6):460–470, 2011.

[99] Prabhakaran Kasinathan, Claudio Pastrone, Maurizio A Spirito, and Mark Vinkovits. Denial-of-service detection in 6lowpan based internet of things. In *Proceedings of International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 600–607. IEEE, 2013.

[100] John G Kemeny, James Laurie Snell, et al. *Finite markov chains*, volume 356. van Nostrand Princeton, NJ, 1960.

[101] Farid Kendoul. Survey of advances in guidance, navigation, and control of unmanned rotorcraft systems. *Journal of Field Robotics*, 29(2):315–378, 2012.

[102] Sye Loong Keoh, Sandeep S Kumar, and Hannes Tschofenig. Securing the internet of things: A standardization perspective. *IEEE Internet of Things Journal*, 1(3):265–275, 2014.

[103] Swati Khandelwal. World's largest 1 tbps ddos attack launched from 152,000 hacked smart devices. http://thehackernews.com/2016/09/ddos-attack-iot.html. [Online; accessed October 2016].

[104] Syed A Khayam and Hayder Radha. Markov-based modeling of wireless local area networks. In *Proceedings of the 6th ACM international workshop on Modeling analysis and simulation of wireless and mobile systems*, pages 100–107. ACM, 2003.

[105] Joseph Kirk. Traveling Salesman Problem - Genetic Algorithm. http://www.mathworks.com/matlabcentral/fileexchange/13680-traveling-salesman-problem-genetic-algorithm. [Online; accessed November 2016].

[106] Sokol Kosta, Andrius Aucinas, Pan Hui, Richard Mortier, and Xinwen Zhang. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *Proceedings of IEEE INFOCOM'12*, pages 945–953. IEEE, 2012.

[107] Sokol Kosta, Vasile Claudiu Perta, Julinda Stefa, Pan Hui, and Alessandro Mei. Clone2clone (c2c): Peer-to-peer networking of smartphones on the cloud. In *Proceedings of the 5th USENIX Workshop on Hot Topics in Cloud Computing*, 2013.

[108] Sokol Kosta, Vasile Claudiu Perta, Julinda Stefa, Pan Hui, and Alessandro Mei. Clonedoc: exploiting the cloud to leverage secure group collaboration mechanisms for smartphones. In *Proceedings of IEEE INFOCOM'13*, pages 19–20. IEEE, 2013.

[109] Thomas Kothmayr, Corinna Schmitt, Wen Hu, Michael Brünig, and Georg Carle. DTLS based security and two-way authentication for the Internet of Things. *Elsevier Ad Hoc Networks*, 11(8):2710–2723, 2013.

[110] Matthias Kovatsch, Simon Duquennoy, and Adam Dunkels. Erbium (Er) REST engine and CoAP implementation for Contiki. *ETH Zurich, Switzerland*, 2015.

[111] Tadahiro Kuroda, Kojiro Suzuki, Shinji Mita, Tetsuya Fujita, Fumiyuki Yamane, Fumihiko Sano, Akihiko Chiba, Yoshinori Watanabe, Koji Matsuda, Takeo Maeda, et al. Variable supply-voltage scheme for low-power high-speed cmos digital design. *IEEE Journal of Solid-State Circuits*, 33(3):454–462, 1998.

[112] Hyeokjin Kwon, Jiye Park, and Namhi Kang. Challenges in deploying CoAP over DTLS in resource constrained environments. In *Proceedings of International Workshop on Information Security Applications*, pages 269–280. Springer, 2015.

[113] Hyun Jung La and Soo Dong Kim. Technical challenges and solutions to realize ultra-heterogeneous mobile computing. *Advanced Science Letters*, 21(3):290–296, 2015.

[114] Christina S Leslie, Eleazar Eskin, and William Stafford Noble. The spectrum kernel: A string kernel for SVM protein classification. In *Pacific symposium on biocomputing*, number 7, pages 566–575, 2002.

[115] Yuan Li, Mingjun Hou, Heng Liu, and Yi Liu. Towards a theoretical framework of strategic decision, supporting capability and information

sharing under the context of internet of things. *Information Technology and Management*, 13(4):205–216, 2012.

[116] Marc Liberatore and Brian Neil Levine. Inferring the source of encrypted http connections. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 255–263. ACM, 2006.

[117] Marino Linaje and Luis Miguel Dominguez-Peinado. U-AirPoll: Mobile Distributed and Collaborative Air Pollution Measurement. *ERCIM News*, (93):42–43, 2013.

[118] Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. Text classification using string kernels. *Journal of Machine Learning Research*, 2(2):419–444, 2002.

[119] Lookout. The new notcompatible: Sophisticated and evasive threat harbors the potential to compromise enterprise networks. https://blog.lookout.com/blog/2014/11/19/notcompatible/. [Online; accessed April 2016].

[120] M0Droid. M0droid. http://m0droid.netai.net/modroid/. [Online; accessed April 2016].

[121] Ming Ma, Yuanyuan Yang, and Miao Zhao. Tour planning for mobile data-gathering mechanisms in wireless sensor networks. *IEEE Transactions on Vehicular Technology*, 62(4):1472–1483, 2013.

[122] Akm Jahangir Alam Majumder, Piyush Saxena, and Sheikh Iqbal Ahamed. Your Walk is My Command: Gait Detection on Unconstrained Smartphone Using IoT System. In *Proceedings of the 40th IEEE Annual Computer Software and Applications Conference (COMPSAC)*, pages 798–806. IEEE, 2016.

[123] Zbynek Michlovskỳ, Shaoning Pang, Nikola Kasabov, Tao Ban, and Youki Kadobayashi. String Kernel Based SVM for Internet Security Implementation. In *International Conference on Neural Information Processing*, pages 530–539. Springer, 2009.

[124] Pedro Miranda, Matti Siekkinen, and Heikki Waris. TLS and energy consumption on a mobile device: A measurement study. In *Computers and Communications (ISCC), 2011 IEEE Symposium on*, pages 983–989. IEEE, 2011.

[125] Prateek Mittal, Matthew Wright, and Nikita Borisov. Pisces: Anonymous communication using social networks. In *Proceedings of Network and Distributed System Security Symposium (NDSS)*, pages 1417–1433, 2012.

[126] Mohammad-Mahdi Moazzami, Daisuke Mashima, Ulrich Herberg, Wei-Pen Chen, and Guoliang Xing. SPOT: a smartphone-based control app with a device-agnostic and adaptive user-interface for IoT devices. In *Proceedings of the ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*, pages 670–673. ACM, 2016.

[127] Abedelaziz Mohaisen and Yongdae Kim. Dynamix: anonymity on dynamic social structures. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, pages 167–172. ACM, 2013.

[128] Aziz Mohaisen, Huy Tran, Abhishek Chandra, and Yongdae Kim. Trustworthy distributed computing on social networks. pages 333–345. IEEE, 2014.

[129] Luca Mottola. Real-world drone sensor networks: A multi-disciplinary challenge. In *Proceedings of the 6th ACM Workshop on Real World Wireless Sensor Networks*, pages 1–1. ACM, 2015.

[130] Luca Mottola, Mattia Moretta, Kamin Whitehouse, and Carlo Ghezzi. Team-level programming of drone sensor networks. In *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*, pages 177–190. ACM, 2014.

[131] Chase C Murray and Amanda G Chu. The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery. *Transportation Research Part C: Emerging Technologies*, 54:86–109, 2015.

[132] Alexios Mylonas, Vasilis Meletiadis, Bill Tsoumas, Lilian Mitrou, and Dimitris Gritzalis. Smartphone forensics: A proactive investigation scheme for evidence acquisition. In *IFIP International Information Security Conference*, pages 249–260. Springer, 2012.

[133] Fairuz Amalina Narudin, Ali Feizollah, Nor Badrul Anuar, and Abdullah Gani. Evaluation of machine learning classifiers for mobile malware detection. *Soft Computing*, 20(1):343–357, 2016.

[134] Plamen Nedeltchev. The internet of everything is the new economy. http://www.cisco.com/c/en/us/solutions/collateral/enterprise/cisco-on-cisco/Cisco_IT_Trends_IoE_Is_the_New_Economy.html. [Online; accessed October 2016].

[135] Brandon Niemczyk and Prasad Rao. Identification over encrypted channels. *Presentation at BlackHat USA*, 2014.

[136] Kirill Nikitin. DTLS Adaptation for Efficient Secure Group Communication, 2015.

[137] Huansheng Ning, Hong Liu, et al. Cyber-physical-social based security architecture for future internet of things. *Advances in Internet of Things*, 2(01):1, 2012.

[138] Ilker Onat and Ali Miri. A real-time node-based traffic anomaly detection algorithm for wireless sensor networks. In *Proceedings of Systems Communications (ICW'05)*, pages 422–427. IEEE, 2005.

[139] Fredrik Osterlind, Adam Dunkels, Joakim Eriksson, Niclas Finne, and Thiemo Voigt. Cross-level sensor network simulation with Cooja. In *Proceedings of 31st IEEE Conference on Local Computer Networks*, pages 641–648. IEEE, 2006.

[140] Yin Minn Pa Pa, Shogo Suzuki, Katsunari Yoshioka, Tsutomu Matsumoto, Takahiro Kasama, and Christian Rossow. IoTPOT: analysing the rise of IoT compromises. In *Proceedings of 9th USENIX Workshop on Offensive Technologies (WOOT 15)*, 2015.

[141] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. Website fingerprinting in onion routing based anonymization networks. In *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society*, pages 103–114. ACM, 2011.

[142] Jiye Park and Namhi Kang. Lightweight secure communication for CoAP-enabled internet of things using delegated DTLS handshake. In *Proceedings of IEEE International Conference on Information and Communication Technology Convergence (ICTC)*, pages 28–33. IEEE, 2014.

[143] Pubudu N Pathirana, Nirupama Bulusu, Andrey V Savkin, and Sanjay Jha. Node localization using mobile robots in delay-tolerant sensor networks. *IEEE Transactions on Mobile Computing*, 4(3):285–296, 2005.

[144] Sarah Perez. Majority of digital media consumption now takes place in mobile apps. http://techcrunch.com/2014/08/21/majority-of-digital-media-consumption-now-takes-place-in-mobile-apps/. [Online; accessed September 2016].

[145] Sophia Petridou and Stylianos Basagiannis. Towards energy consumption evaluation of the ssl handshake protocol in mobile communications. In *Proceedings of 9th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*, pages 135–138. IEEE, 2012.

[146] Heloise Pieterse and Martin S Olivier. Android botnets on the rise: Trends and characteristics. In *2012 Information Security for South Africa*, pages 1–5. IEEE, 2012.

[147] Georgios Portokalidis, Philip Homburg, Kostas Anagnostakis, and Herbert Bos. Paranoid android: versatile protection for smartphones. In *Proceedings of the 26th Annual Computer Security Applications Conference*, pages 347–356. ACM, 2010.

[148] Nachiketh R Potlapally, Srivaths Ravi, Anand Raghunathan, and Niraj K Jha. A study of the energy consumption characteristics of cryptographic algorithms and security protocols. *IEEE Transactions on mobile computing*, 5(2):128–143, 2006.

[149] Nachiketh R Potlapally, Srivaths Ravi, Anand Raghunathan, and Niraj K Jha. A study of the energy consumption characteristics of cryptographic algorithms and security protocols. *IEEE Transactions on mobile computing*, 5(2):128–143, 2006.

[150] Niels Provos, Dean McNamee, Panayiotis Mavrommatis, Ke Wang, Nagendra Modadugu, et al. The ghost in the browser: Analysis of web-based malware. *Proceedings of the first USENIX workshop on hot topics in Bot-nets (HotBots'07*, 7:4–13, 2007.

[151] Krishna PN Puttaswamy, Alessandra Sala, Omer Egecioglu, and Ben Y Zhao. Rome: Performance and anonymity using route meshes. In *Proceedings of IEEE INFOCOM'09*, pages 2861–2865. IEEE, 2009.

[152] Krishna PN Puttaswamy, Alessandra Sala, and Ben Y Zhao. Starclique: guaranteeing user privacy in social networks against intersection attacks. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pages 157–168. ACM, 2009.

[153] Mazedur Rahman, Jerry Gao, and Wei-Tek Tsai. Energy saving in mobile cloud computing. In *Proceedings of IEEE International Conference on Cloud Engineering (IC2E)*, pages 285–291. IEEE, 2013.

[154] Kasper Rasmussen, Mani Srivastava, et al. Secure location verification with hidden and mobile base stations. *IEEE Transactions on Mobile Computing*, 7(4):470–483, 2008.

[155] Shahid Raza, Ludwig Seitz, Denis Sytenkov, and Göran Selander. S3K: Scalable Security With Symmetric Keys: DTLS Key Establishment for the Internet of Things. *IEEE Transactions on Automation Science and Engineering*, (99):1–11, 2016.

[156] Shahid Raza, Linus Wallgren, and Thiemo Voigt. SVELTE: Real-time intrusion detection in the Internet of Things. *Elsevier Ad hoc networks*, 11(8):2661–2674, 2013.

[157] Piotr Rudol and Patrick Doherty. Human body detection and geolocalization for uav search and rescue missions using color and thermal imagery. In *Proceedings of IEEE Aerospace Conference*, pages 1–8. IEEE, 2008.

[158] CNN Ryan Bergeron. Your personal underwater drone. http://edition.cnn.com/2013/11/06/tech/innovation/underwater-drones/. [Online; accessed October 2016].

[159] Bhaskar Pratim Sarma, Ninghui Li, Chris Gates, Rahul Potharaju, Cristina Nita-Rotaru, and Ian Molloy. Android permissions: a perspective combining risks and benefits. In *Proceedings of the 17th ACM symposium on Access Control Models and Technologies*, pages 13–22. ACM, 2012.

[160] Marius Senftleben, Mihai Bucicoiu, Erik Tews, Frederik Armknecht, Stefan Katzenbeisser, and Ahmad-Reza Sadeghi. Mop-2-mop–mobile private microblogging. In *International Conference on Financial Cryptography and Data Security*, pages 384–396. 2014.

[161] Stefaan Seys and Bart Preneel. Arm: Anonymous routing protocol for mobile ad hoc networks. *International Journal of Wireless and Mobile Computing*, 3(3):145–155, 2009.

[162] Asaf Shabtai, Uri Kanonov, Yuval Elovici, Chanan Glezer, and Yael Weiss. Andromaly: a behavioral malware detection framework for android devices. *Journal of Intelligent Information Systems*, 38(1):161–190, 2012.

[163] Asaf Shabtai, Lena Tenenboim-Chekina, Dudu Mimran, Lior Rokach, Bracha Shapira, and Yuval Elovici. Mobile malware detection through analysis of deviations in application network behavior. *Computers & Security*, 43:1–18, 2014.

[164] Rishabh Sharma, Sanjay Kumar, and Munesh Chandra Trivedi. Mobile cloud computing: A needed shift from cloud to mobile cloud. In *5th International Conference on Computational Intelligence and Communication Networks (CICN)*, pages 536–539. IEEE, 2013.

[165] Justine Sherry, Chang Lan, Raluca Ada Popa, and Sylvia Ratnasamy. Blindbox: Deep packet inspection over encrypted traffic. In *ACM SIGCOMM Computer Communication Review*, pages 213–226. ACM, 2015.

[166] Justine Sherry, Chang Lan, Raluca Ada Popa, and Sylvia Ratnasamy. Blindbox: Deep packet inspection over encrypted traffic. In *ACM SIGCOMM Computer Communication Review*, pages 213–226. ACM, 2015.

[167] Wolfgang Slany. Tinkering with pocket code, a scratch-like programming app for your smartphone. *Proceedings of Constructionism 2014, Vienna*, 2014.

[168] David C Snowdon, Stefan M Petters, and Gernot Heiser. Accurate on-line prediction of processor and memoryenergy usage under voltage scaling. In *Proceedings of the 7th ACM & IEEE international conference on Embedded software*, pages 84–93. ACM, 2007.

[169] Dawn Xiaodong Song, David Wagner, and Xuqing Tian. Timing Analysis of Keystrokes and Timing Attacks on SSH. In *Proceedings of USENIX Security Symposium (HotSec'01)*, 2001.

[170] Guillermo Suarez-Tangil, Juan E Tapiador, Pedro Peris-Lopez, and Jorge Blasco. Dendroid: A text mining approach to analyzing and classifying code structures in android malware families. *Expert Systems with Applications*, 41(4):1104–1117, 2014.

[171] Paul Syverson, R Dingledine, and N Mathewson. Tor: the second-generation onion router. In *Proceedings of USENIX Security Symposium (HotSec'04)*, 2004.

[172] Sasu Tarkoma, Matti Siekkinen, Eemil Lagerspetz, and Yu Xiao. *Smartphone energy consumption: modeling and optimization.* Cambridge University Press, 2014.

[173] Adam Thierer and Andrea Castillo. Projecting the Growth and Economic Impact of The Internet of Things. *George Mason University, Mercatus Center, June*, 15, 2015.

[174] Marco Tiloca. Efficient protection of response messages in dtls-based secure multicast communication. In *Proceedings of the 7th International Conference on Security of Information and Networks*, pages 466–473. ACM, 2014.

[175] Marten Van Dijk and Ari Juels. On the impossibility of cryptography alone for privacy-preserving cloud computing. In *Proceedings of USENIX Security Symposium (HotSec'10).*

[176] Vladimir Vapnik. *The nature of statistical learning theory.* Springer Science & Business Media, 2013.

[177] Vladimir N Vapnik. Statistical learning theory. adaptive and learning systems for signal processing, communications, and control. *Simon Haykin*, 1998.

[178] Edwin Vattapparamban, İsmail Güvenç, Ali İ Yurekli, Kemal Akkaya, and Selçuk Uluağaç. Drones for smart cities: Issues in cybersecurity, privacy, and public safety. In *Proceedings of IEEE International Wireless Communications and Mobile Computing Conference (IWCMC'16)*, pages 216–221. IEEE, 2016.

[179] Srdjan Čapkun and Jean-Pierre Hubaux. Secure positioning in wireless networks. *IEEE Journal on Selected Areas in Communications*, 24(2):221–232, 2006.

[180] Tim Verbelen, Pieter Simoens, Filip De Turck, and Bart Dhoedt. Cloudlets: bringing the cloud to the mobile user. In *Proceedings of the third ACM workshop on Mobile cloud computing and services*, pages 29–36. ACM, 2012.

[181] John Villasenor. Observations from above: unmanned aircraft systems and privacy. *Harv. JL & Pub. Pol'y*, 36:457, 2013.

[182] VirusTotoal. Virustotoal. https://www.virustotal.com/. [Online; accessed April 2016].

[183] Xiaolei Wang, Jie He, and Yuexiang Yang. Identifying p2p network activities on encrypted traffic. In *Proceedings of IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pages 893–899. IEEE, 2014.

[184] Lee J Wells, Jaime A Camelio, Christopher B Williams, and Jules White. Cyber-physical security challenges in manufacturing systems. *Elsevier Manufacturing Letters*, 2(2):74–77, 2014.

[185] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics bulletin*, 1(6):80–83, 1945.

[186] Duncan S Wong, Hector Ho Fuentes, and Agnes Hui Chan. The performance measurement of cryptographic primitives on palm devices. In *Proceedings 17th Annual Computer Security Applications Conference (ACSAC'01).*, pages 92–101. IEEE, 2001.

[187] Duncan S Wong, Hector Ho Fuentes, and Agnes Hui Chan. The performance measurement of cryptographic primitives on palm devices. In *Proceedings of the 17th Annual Computer Security Applications Conference (ACSAC).*, pages 92–101. IEEE, 2001.

[188] Charles V Wright, Lucas Ballard, Scott E Coull, Fabian Monrose, and Gerald M Masson. Spot me if you can: Uncovering spoken phrases in encrypted VoIP conversations. In *IEEE Symposium on Security and Privacy (S&P'08)*, pages 35–49. IEEE, 2008.

[189] Liang Xie, Xinwen Zhang, Jean-Pierre Seifert, and Sencun Zhu. pBMDS: a behavior-based malware detection system for cellphone devices. In *Proceedings of the third ACM conference on Wireless network security*, pages 37–48. ACM, 2010.

[190] Ke Xu, Yi Qu, and Kun Yang. A tutorial on the internet of things: from a heterogeneous network integration perspective. *IEEE Network*, 30(2):102–108, 2016.

[191] Rubin Xu, Hassen Saïdi, and Ross Anderson. Aurasium: Practical policy enforcement for android applications. In *Proceedings of the 21st USENIX Security Symposium (USENIX Security'12)*, pages 539–552, 2012.

[192] Nong Ye et al. A markov chain model of temporal behavior for anomaly detection. In *Proceedings of the IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop*, volume 166, pages 169–170. West Point, NY, 2000.

[193] Yanchao Zhang, Wei Liu, Yuguang Fang, and Dapeng Wu. Secure localization and authentication in ultra-wideband sensor networks. *IEEE Journal on Selected areas in communications*, 24(4):829–835, 2006.

[194] Yanchao Zhang, Wei Liu, Wenjing Lou, and Yuguang Fang. MASK: anonymous on-demand routing in mobile ad hoc networks. *IEEE transactions on wireless communications*, 5(9):2376–2385, 2006.

[195] Min Zhao, Tao Zhang, Fangbin Ge, and Zhijian Yuan. Robotdroid: a lightweight malware detection framework on smartphones. *Journal of Networks*, 7(4):715–722, 2012.

[196] Wu Zhou, Yajin Zhou, Xuxian Jiang, and Peng Ning. Detecting repackaged smartphone applications in third-party android marketplaces. In *Proceedings of the second ACM conference on Data and Application Security and Privacy*, pages 317–326. ACM, 2012.

[197] Yajin Zhou and Xuxian Jiang. Dissecting android malware: Characterization and evolution. In *IEEE Symposium on Security and Privacy (S&P'12)*, pages 95–109. IEEE, 2012.

[198] Alexander Zien, Gunnar Rätsch, Sebastian Mika, Bernhard Schölkopf, Thomas Lengauer, and K-R Müller. Engineering support vector machine kernels that recognize translation initiation sites. *Bioinformatics*, 16(9):799–807, 2000.