ON SECURING WI-FI DIRECT BASED OPPORTUNISTIC NETWORKS

A Dissertation

by

ALA' (MOHAMMAD HAFEZ) BARAKAT ALTAWEEL

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	Radu Stoleru
Committee Members,	Guofei Gu
	I-Hong Hou
	Jyh-Charn Steve Liu
Head of Department,	Dilma Da Silva

May 2019

Major Subject: Computer Engineering

ABSTRACT

Today's smartphones, tablets, and notebooks are equipped with Wi-Fi Direct (the de facto adhoc communication mechanism for mobile devices) that allows users to establish a wireless network (without a wireless router) and exchange data among their devices. The Wi-Fi Direct protocol, developed by the Wi-Fi Alliance, is built upon the IEEE 802.11 infrastructure and it implements the Wi-Fi Protected Setup (WPS) protocol to establish a secure key and connection between two devices. The shipments of Wi-Fi Direct devices reached 1.7 billion in 2016, and it is predicted to reach 3 billion by 2019. With the prevalence of Wi-Fi Direct devices nowadays, the Wi-Fi Direct based Opportunistic Networks (WDON) will play a crucial role in the future mobile networks. A WDON refers to the network paradigm where mobile devices communicate with each other through the opportunistically formed Wi-Fi Direct links. The WDONs have a wide range of applications, e.g., disaster response, battlefield communications, social networks applications, etc.

In this dissertation, we identify several vulnerabilities of WDONs, which pose severe threats to the authentication and data confidentiality: a) the brute-force/dictionary attack on the PIN method of the WPS protocol; b) the *EvilDirect* attack on the Push-Button method of the WPS protocol; and c) the *CollusiveHijack* attack on routing protocols in WDONs. Consequently, in order to address the aforementioned vulnerabilities, we propose a security framework to defend against these attacks. Our framework contains a set of secure-key-establishment algorithm and protocols that aim to provide *secure* communication services in WDONs. In this framework, we propose: a) an algorithm to establish a secure key (128 bits) from the contextual sensors data of the devices in WDONs; b) a challenge-response protocol to detect the EvilDirect attack in both *dynamic* and *static* environments of WDONs; and c) two detection protocols to detect the CollusiveHijack attack in WDONs. We evaluate the proposed algorithm and protocols through extensive simulations and proof-of-concept implementations in smartphones and notebooks. The evaluation results show that, the proposed framework prevents the brute-force/dictionary attack and detects, within seconds, both EvilDirect and CollusiveHijack attacks with relatively high detection rates while maintaining low false positive rates.

DEDICATION

To My Beloved Family:

My Parents,

My Wife,

My Brothers and Sisters,

My Son and Daughter.

ACKNOWLEDGMENTS

Prophet Muhammad, peace and blessings be upon him, said, "He who does not thank people, does not thank Allah". This dissertation at the Department of Computer Science and Engineering at Texas A&M University was made possible due to the support of several agencies. I gratefully acknowledge the German Jordanian University - Jordan, the National Science Foundation, the National Institute of Standards and Technology, and the Office of Graduate Studies at TAMU.

My first talk with Dr. Radu Stoleru was in March 2012 when I was at Jordan. I was excited about the research projects he works on and happy when he accepted me into the LENSS lab. When I took the first class (CSCE 662) with him in Spring 2013, I was inspired by his style of teaching, thorough knowledge, and the exposure to recent research topics provided by his class. Also, I was astonished by the sufficient time that he spends with his students during the weekly 1-1 and group meetings. His patient mentoring, good listening, willingness to guide, emotional intelligence that tolerates my limitations during the hundreds of meetings we spent together solving research problems allowed me to learn, develop and enhance a unique set of skills necessary for successful research and made me consider research as a career. Outside the lab, his friendly, helpful, supportive and humble personality gave me the relief from the high pressure during my PhD - those flying trips and thanksgiving and barbecue parties were super fun. This dissertation would not have been done without Dr. Stoleru’s guidance. Thank you Sir. It has been a privilege to know you.

I would like to thank Dr. Guofei Gu, Subhajit Mandal, and Arnab Kumar Maity for their collaborations and contributions in this research. I would also like to thank Dr. I-Hong Hou, Dr. Jyh-Charn Steve Liu and again Dr. Guofei Gu for being part of my committee and for their questions, suggestions, positive and helpful feedback during my preliminary exam and dissertation defense. I really appreciate the 1-1 meetings that Dr. Guofei Gu spent with me answering my questions and contributing to my research. Also, I would like to thank Mr. Bruce Veals at the Department of Computer Science and Engineering at Texas A&M University for providing us with smartphones to implement and evaluate our proposed algorithms and protocols.

Former and current members of the LENSS Lab - Myounggyu, Amin, Harsha, Wei, Mahima, Jay, Chen, Mengyuan, Yukun, Suman, Mohammad and others - thank you for your help and time answering my questions.

I would like to thank my parents, Hafez and Mariam, for always being there for me. This dissertation would not have been completed without the solid foundation my mother taught me during my early schooling years. Rest in peace Mama.

Dear Sajida, you have been a constant source of love, help, support, encouragement and food. Thank you for your patience with me while we are away from our families at Jordan. And as I always say, I can not even imagine my life without you.

I would also like to thank all my friends in College Station - Mustafa, Anas, Abdullah, Yazan, Hussien, Ahmad Bashaireh, Ahmad Abu Alrub, Ashraf Alhnate, Ashraf Alzoubi, Ahmad Bani Younes, Suliman, Shadi, Issa, Mutaz, Muneer, Ibrahim, and others - those Saturdays barbecue and potluck parities were super fun and gave me the relief from the high pressure during the weekdays.

Last but not the least, I would like to thank all my previous teachers in my middle and high schools in Jordan, during my Bachelor and Master study at the Jordan University of Science and Technology - Jordan and University of Stuttgart - Germany as well as the staff at the Department of Computer Science and Engineering at TAMU.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a thesis (or) dissertation committee consisting of Professor Radu Stoleru, Professor Guofei Gu and Professor Jyh-Charn Steve Liu of the Department of Computer Science and Engineering, and Professor I-Hong Hou of the Department of Electrical and Computer Engineering.

All work for the thesis (or) dissertation was completed by the student, under the advisement of Professor Radu Stoleru of the Department of Computer Science and Engineering, and in collaboration with Professor Guofei Gu of the Department of Computer Science and Engineering, Subhajit Mandal of the Department of Computer Science and Engineering, and Arnab Kumar Maity of the Department of Statistics.

Funding Sources

Graduate study was supported by scholarships from Texas A&M University and the German Jordanian University.

This work was made possible in part by the National Science Foundation (NSF) under Grant Number #1127449, #1145858 and by the National Institute of Standards and Technology (NIST) under Grant Number #70NANB17H190. Its contents are solely the responsibility of the authors and do not necessarily represent the official views of NSF and/or NIST.

NOMENCLATURE

AP	Access Point
AS	Autonomous System
D-H	Diffie-Hellman
DoS	Denial-of-Service
GO	Group Owner
HDT	Hop Detection Technique
HRP	Hybrid Routing Protocol
ICT	Inter-Contact Time
KS2ST	Kolmogorov-Smirnov Two-Sample Test
MDS	Multi-Dimensional Scaling
P2P	Peer-to-Peer
PBC	Push-Button Configuration
PDT	Path Detection Technique
RSS	Received Signal Strength
RSSI	Received Signal Strength Indicator
SekGens	Session Key Generated from Sensors
WDON	Wi-Fi Direct based Opportunistic Network
WPS	Wi-Fi Protected Setup

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 Motivation

Recent advances in smart device technologies (i.e., smartphones, tablets, and notebooks) have equipped them with a new emerging wireless standard that allows users to establish an adhoc wireless network and exchange data among them. The new standard, the Wi-Fi Direct protocol [1], is built upon the IEEE 802.11 infrastructure and it inherits the security mechanisms that have been developed for it. Wi-Fi Direct enables the devices to form P2P groups by negotiating which device will be the Group Owner (GO) and which devices will be the clients. Wi-Fi Direct is used for data sharing, traffic offloading, video streaming, mobile printing, and gaming [2–7]. The shipments [8] of Wi-Fi Direct devices reached around 1,700 million in 2016, and it is predicted to reach three billion in 2019. Accordingly, we envision that Wi-Fi Direct based Opportunistic Network (WDON) will play a crucial role in the future mobile networks. In WDON, the mobile devices communicate with each other through the opportunistically formed Wi-Fi Direct links. Routing in WDON relies on node mobility and the store-and-forward mechanism. The WDONs have a wide range of applications, e.g., disaster response [9], battlefield communications, and social networks applications (Firechat [10], 1am [11]).

Wi-Fi Direct, like other direct device-to-device wireless technologies, is vulnerable to many security attacks against confidentiality, authentication and integrity. In order to establish a secure key and connection between two devices, Wi-Fi Direct implements the in-band configuration mode of Wi-Fi Protected Setup (WPS) protocol [12], which has two methods: the PIN method and the Push-Button method. In this dissertation, we investigate the security vulnerabilities and attacks against the WDONs. We tackle these vulnerabilities in a bottom-up approach by moving along the Datalink and Network layers of the TCP/IP protocol stack of WDONs. In the following, we describe two vulnerabilities of the Datalink layer in WDONs that motivate this research.

- In the PIN method of the WPS protocol, a device password, which is obtained from the GO and entered into the client manually using a keypad, is used to perform a Diffie-Hellman (D-H) key exchange to authenticate the two devices. The security flaws in the PIN method design, which cause its online and offline PIN brute-force/dictionary vulnerabilities were discovered by Viehbock [13] and Bongard [14], respectively. Accordingly, they recommended to disable the PIN method in the wireless routers (i.e., at that time, the PIN method was mainly used by wireless routers). Sanatinia et al. [15] studied the design flaws in the WPS's PIN method and how it can give an adversary leverage to compromise a connected network of Access Points (APs). Sanatinia et al. proposed new designs and frameworks for APs to allow more scalable and flexible administration to avoid the spreading of APs infections. However, their proposed solution can not be used in Wi-Fi Direct networks due to the absence of network administrators. Indeed, nowadays, the WPS protocol has been abandoned in the wireless routers and replaced with IEEE 802.11i [16]. IEEE 802.11i is designed for infrastructure Wi-Fi networks and cannot be used in Wi-Fi Direct. As a result, the PIN brute-force/dictionary attack is still an open research problem for the WPS PIN method.

- In the Push-Button method of the WPS protocol, the client invites the GO to start the WPS protocol and waits for its acceptance. The GO's user acceptance is used for authentication and to perform a D-H key exchange. We discovered the vulnerability of the GO devices in the Push-Button method to a new attack, we refer to as the *EvilDirect* attack. In the EvilDirect attack, a malicious attacker, Eve, intercepts the client's invitation request and accepts it before the legitimate GO. Accordingly, Eve hijacks the wireless communications between the clients and the legitimate GO. We demonstrated through real experiments that the attacker can successfully launch EvilDirect attack due to the fact that two GOs (with the same MAC address, SSID, and operating channel) are indistinguishable by the clients and treated as a single GO. Many approaches to detect spoofing and rogue APs in wireless networks like [17–22] are based on differentiating the traffic between the wired and wireless connections and comparing with an authorized list. These approaches require network administrators, which are not available in Wi-Fi Direct networks. Many wireless intrusion detection systems (WIDS) like [23–25] have been proposed to detect session hijacking attacks in wireless networks. These WIDS require training and use sniffers to detect deviations from normal behaviors. Indeed, WIDS and sniffers are not always available for Wi-Fi Direct devices. As a result, EvilDirect is still an open research problem for the WPS Push-Button method.

Due to the intermittent connectivity between the nodes in WDONs, routing protocols for the connected networks (mobile adhoc or mesh networks) cannot guarantee packet delivery in an opportunistic network scenario. Hence, WDON routing protocols adopt the store-and-forward mechanism to deliver the packets. That is, buffer the packet if the next hop is unavailable and transmit it when the next hop is available. Two of the most novel routing protocols for WDONs are the Hybrid Routing Protocol (HRP) [26] and the Prophet protocol [27] [28]. Instead of flooding the packets to all encountered nodes as is the case in Epidemic protocol [29], HRP and Prophet protocols learn from the nodes' past contact history in order to reach the destination nodes. A pair of nodes measures the contact frequencies in the past to find the probability of future contacts. The Inter Contact Time (ICT) (i.e., the time duration between two contact events between a pair of nodes) is used to measure the contact frequency. In the following, we describe one vulnerability of the Network layer in WDONs that motivate this research.

- We discovered the vulnerability of HRP and Prophet protocols to Route Hijacking attack. In this attack, a malicious attacker, Eve, compromises a set of nodes and lies about their (Inter-Contact Times) ICTs. Eve claims that her nodes meet more frequently than in reality, in order to deceive HRP and Prophet protocols. Eve aims to hijack the packets of the legitimate nodes in the WDON. The attack, called *CollusiveHijack*, can be launched due to the fact that nodes in WDONs have no way of verifying whether the claimed ICT between a pair of nodes is true or false, even if the claimed ICTs are signed (Eve can successfully launch the CollusiveHijack attack since her nodes share their keys and sign their claimed ICTs). The CollusiveHijack enables Eve to launch more severe attacks like: a) packet modification attack [30], which enables Eve to corrupt the contents of the packets, thus enforcing packet retransmissions and a decrease in the packet delivery ratio [31] (i.e., waste of network resources, power, bandwidth); b) eavesdropping and traffic analysis attack, which enables Eve to identify the types of network traffic and apps of the legitimate nodes [32] [33]; c) incentives seeking attack (i.e., if an incentive based mechanism [34] is employed in the WDON, Eve's nodes can deliver the hijacked packets to get more credit and higher reputation). To the best of our knowledge, a route hijacking attack has neither been identified nor addressed in WDONs as most of the previous research in these networks addressed flood, wormhole, and packet dropping attacks [35–39]. However, many approaches have been proposed to detect route hijacking in Internet. Approaches [40–44] collect BGP updates and routing tables from a public BGP monitoring infrastructure [45–47] and raise alarms when a change in the origin Autonomous System (AS) of a prefix or a suspicious route is observed. Approaches [40–44] require network administrators (not available in WDONs) and need to create a list of owned/reached IPs a priori (the WDONs' nodes do not know their future contacts). Other approaches [48] [49] continuously probe Internet to detect whether any data path changes. They use pings/traceroutes to monitor the connectivity of a prefix and raise an alarm when significant changes in the reachability of a prefix or the paths leading to it are detected. Due to the intermittent connectivity among the nodes in WDONs, pings/traceroutes fail to work. As a result, the CollusiveHijack attack is still an open research problem for WDONs.

To overcome the aforementioned vulnerabilities of the Datalink and Network layers in WDONs, in this dissertation, we present a security framework, as shown in Table 1.1. The proposed security framework contains a set of secure-key-establishment algorithm and challenge-response protocols. Our framework makes use of contextual information (i.e., data obtained from smartphones' sensors and wireless cards) to defend against the vulnerability of the WPS's PIN method to online/offline brute-force/dictionary attack(s) and the vulnerability of the GO devices in the WPS's Push-Button method to the *EvilDirect* attack. We decided to use the contextual information to address these attacks due to the following reasons. First, the contextual information can be directly obtained from the smartphones' on-board sensors. Second, it is difficult for the attacker to

Table 1.1: Security attacks against WDONs and our proposed solutions.

Network layer		**CollusiveHijack Attack PDT & HDT [53]**
Datalink layer	**Push Button method**	**EvilDirect Attack EvilDirectHunter [52]**
	PIN method	**Brute-Force Attack SekGens [51]**

access these information or to get (physically) close to the victim devices to get similar contextual information (i.e., in case of RSS and sound sensor data). Third, due to the fact that the properties of the radio channel are unique to the locations of any two devices in any environment [50], the RSS information are identical and unique for the two devices. We solved the problem of the PIN method's vulnerability by proposing a Session Key Generated from Sensors (SekGens) algorithm to establish a long (128 bits) secure session key between two Wi-Fi Direct devices [51]. For the Push-Button method's vulnerability, we propose a challenge-response protocol, *EvilDirectHunter*, to detect the EvilDirect attack [52], as is shown in Table 1.1. For the CollusiveHijack attack, we propose two techniques: the Path Detection Technique (PDT) and the Hop Detection Technique (HDT) [53]. Both PDT and HDT leverage the Kolmogorov-Smirnov two-sample test (KS2ST) [54] to detect this attack, but they offer a trade off between the *compatibility* with the Bundle Security Protocol (BSP) [55] (if additional steps are required at the intermediate nodes) and the *detection capability* against the CollusiveHijack attack. The KS2ST [54] is a well known test for distinguishing between two statistical distributions (by measuring the distance between the empirical distribution functions of two samples and based on that, it determines whether the two samples have been drawn from the same distribution or not). The KS2ST has been widely used by previous works for detecting covert channels, detecting selfish wireless nodes, and in intrusion detection systems [56–59].

1.2 Dissertation Statement

The design of a security framework for WDON is feasible through: a) new secure and robust link layer key establishment algorithms; and b) accurate detection and defense protocols against session and route hijacking attacks.

1.3 Main Contributions

The main contributions of this dissertation are outlined as follows:

- We propose **a Session Key Generated from Sensors (SekGens) algorithm [51]** to secure the PIN method of the WPS protocol in WDONs. First, we demonstrate a successful real-world brute-force attack against Wi-Fi Direct devices in WDONs. Then, we design the SekGens algorithm to establish a secure key (128 bits) from the contextual sensors data for two Wi-Fi Direct devices in WDONs (instead of using the keypad). We also demonstrate the feasibility of SekGens through a proof-of-concept implementation on Google Nexus 5 smartphones. Moreover, the robustness of SekGens is illustrated when it runs on two different smartphones (i.e., Google Nexus 5 and Samsung Galaxy S2). Finally, we prove the effectiveness of SekGens by showing that it generates keys with low mismatch ratio (less than 3%), at a fast rate ($\sim$20 bits/sec), and with high Shanon entropy ($\sim$92%).

- We propose **a challenge-response protocol (EvilDirectHunter) [52]** to secure the Push-Button method of the WPS protocol in WDONs. First, we demonstrate a successful real-world EvilDirect attack against Wi-Fi Direct GO devices in WDONs. Then, we design the EvilDirectHunter protocol that employs the inherent randomness in the wireless channel between the clients and the GO to detect the EvilDirect attacks in dynamic environments of WDONs (i.e., mobile devices, and/or there are mobile intermediate objects , e.g., airport lounges, cafes, or restaurants). Moreover, we proposes an additional detection phase for the EvilDirectHunter that is based on Multi-Dimensional Scaling (MDS) algorithm [60] to detect the EvilDirect attacks in static environment of WDONs (i.e., static devices, and there are few mobile intermediate objects, e.g., libraries). We also demonstrate the feasibility and robustness of EvilDirectHunter through a proof-of-concept implementation on Google Nexus and Samsung Galaxy S2 smartphones. Finally, we prove the effectiveness of EvilDirectHunter by showing that it is able, within seconds, to detect EvilDirect attacks with a high detection rate (100%) while maintaining a low false positive rate (4.5%).

- We propose **two detection protocols, the Path Detection Technique (PDT) and the Hop Detection Technique (HDT) [53]** to detect the CollusiveHijack attack in WDONs. First, we demonstrates, via implementation on a testbed of Asus Eee notebooks and extensive simulations on the Opportunistic Network Environment (ONE) [61] simulator, a successful CollusiveHijack attack against HRP and Prophet protocols in WDONs. Then, we design PDT and HDT that employ the Kolmogorov-Smirnov two-sample test (KS2ST) [54] to detect the CollusiveHijack and offer a trade off between the *compatibility* with the Bundle Security Protocol (BSP) [55] (if additional steps are required at the intermediate nodes) and the *detection capability* against the CollusiveHijack attack. We also demonstrate the feasibility of both PDT and HDT through a proof-of-concept system implementation on the testbed of Asus Eee notebooks and validate their detection capabilities through extensive simulations on the ONE simulator. Finally, we prove the effectiveness of both PDT and HDT by showing that they are able, within seconds, to detect CollusiveHijack attacks with respectively 80.0% and 99.4% detection rates while maintaining a low false positive rate (3.6%).

1.4 Organization

This dissertation is organized as follows. The current section motivates our work and states the contributions of the research. In Section 2 we review state-of-the-art works related to this dissertation. Section 3 presents the system and security (attacker) models considered in the entire research. In section 4, we present SekGens, a Session Key Generated from Sensors algorithm proposed to secure the PIN method of the WPS protocol in WDONs. In Section 5, we introduce EvilDirectHunter, a challenge-response protocol proposed to secure the Push-Button method of the WPS protocol in WDONs. Section 6 presents the Path Detection Technique (PDT) and the Hop

Detection Technique (HDT), to detect the CollusiveHijack attack in WDONs. Finally, in Section 7, we conclude this dissertation and illustrate its future perspective.

2. STATE OF THE ART

In this section we present the state-of-the-art of this dissertation. We first present the brute-force/dictionary attack against the PIN method of the Wi-Fi Protected Setup (WPS) protocol. Then, we discuss the state-of-the-art algorithms that are responsible for generating session keys from sensors and how previous research exploited the smartphones' sensors in their works. Next, we discuss the evil twin AP attack that occurs in Wi-Fi infrastructure networks and the proposed solutions for it, which includes the network administrator solutions and user oriented solutions. Finally, we present the state-of-the-art works on the security threats in opportunistic networks and the route hijacking attacks in the internet.

2.1 Related Work For Brute-Force/Dictionary Attack

The WPS protocol was introduced by Wi-Fi Alliance in 2007 to enable secure pairing of Wi-Fi devices with wireless routers (or APs). The WPS's online PIN brute-force/dictionary vulnerability, was discovered by Viehbock [13] in 2011. In 2014, Bongard [14] discovered the offline PIN brute-force/dictionary vulnerability. Viehbock and Bongard recommended to disable the WPS protocol on the APs. Indeed, WPS protocol has been abandoned in wireless routers and replaced with IEEE 802.11i (also referred to as Wi-Fi Protected Access II (WPA2)) [16]. WPA2 has two versions: WPA2-Personal and WPA2-Enterprise. WPA2-Personal is designed for home networks and does not require a remote authentication server (i.e., RADIUS). Each device encrypts the network traffic using a 256 bit key, that is generated from a passphrase (i.e., ASCII characters). WPA2 specifications (as is mentioned in page 16 of [62]) state that a key that is generated from a passphrase of less than 20 characters is vulnerable to brute-force/dictionary attacks. WPA2-Enterprise is designed for enterprise networks and requires a RADIUS authentication server to defend against brute-force/dictionary attacks on short passphrases. Due to the fact that entering a long passphrase ($>$ 20 characters) is not convenient from the user's perspective in terms of memorizing it and retrying according to wrong entries, and that the are no RADIUS servers in Wi-

Fi Direct networks, it is therefore challenging to defend against the WPS's PIN online and offline brute-force/dictionary attacks in Wi-Fi Direct networks.

Recent key-establishment algorithms [63–69] utilize the sensors data (accelerometer, audio, or luminosity), or Received Signal Strength (RSS) variations to establish a key between two devices. "Smart its friends" [63] and "Are you with me" [64] benefit from the accelerometer sensors in device pairing. Smart its friends uses sensor data to connect smart-artefact devices when a user holds them and shakes them together. In [64], accelerometer data are used to determine if two ubiquitous computing devices are carried by the same person. The extended work from [63] and [64] is the Candidate Key Protocol (CKP) [65]. In CKP, the devices generate an encryption key from acceleration data by extracting feature vectors from the inputs. Then each feature vector is hashed using a standard hash function and sent to the other device. Depending on the similarity of the received and local hashed vectors, the devices determine if they are shaken together, and they only use the identical hashed vectors to create the shared encryption key. The key bit rate of CKP is 7 bits/sec, less than ours (i.e., 20 bits/sec). Hence, CKP requires more time than SekGens to generate 128 bits. A major drawback of CKP is that both devices have to exchange their own derived key parts, which makes it susceptible to offline brute-force attacks. The security analysis is missing in [65]. Moreover, CKP has no reconciliation phase. Therefore, it requires very low environmental noises to produce enough identical vectors between the devices. Also, there is no clear answer to the question of how large hashed vectors can be ignored due to unequal values without endangering the security and without increasing the time needed to generate the keys (i.e., without decreasing the key bit rate).

Another work presented in [66] aims at generating a cryptographic key by applying appropriate signal processing methods on the acceleration data of small hand-held devices. Compared to [65], the key generation algorithm in [66] does not require the two devices to exchange their acceleration characteristics. However, this algorithm aims at creating a symmetric key that is equally as strong as the typical Bluetooth PIN (i.e., three to four digits that range from 0 to 9). The proposed algorithm generates an equivalent key with 13 bits (i.e., the entropy of Bluetooth PIN of three to four digits ($10^3 \sim 10^4$) $\approx$ the entropy of 13 bits key (2^{13})). Even though the success rate of generating this 13 bits key is 80%, this short key is vulnerable to brute-force attacks in Wi-Fi Direct. Furthermore, the longest achievable key in this approach, which is 140 bits, has a low success rate (i.e., 1/88).

M. Sethi et al. [70] developed a secure mechanism for pairing devices with a touch screen or touch-sensitive surface. The user draws synchronously the same figure on the two devices using two fingers (i.e., the thumb and index finger) of the same hand. This shared input is used as the fuzzy shared secret data for secure key establishment. An important requirement for security in this mechanism is that the fuzzy secret must have sufficient entropy. However, the authors did not show if it is possible to generate a 128 bits key and the needed drawings and time for that. Moreover, there is no clear answer to the question of how this mechanism handles the case if both users want to draw the same figure separately (i.e., each one in his device).

S. Jane et al. [67] evaluated the effectiveness of secret key extraction from the received signal strength (RSS) variations in wireless channels using real-world measurements in static and dynamic environments. Their results showed that the static environments are unsuitable for generating a secret key (i.e., the mismatch ratio is around 50%). However, the key bits extracted in dynamic environments showed a higher secret bit rate with a very low mismatch ratio. These dynamic experiments require either a normal speed walking for 10 to 25 feet, riding a bike on a city streets, or connect the devices in a crowded cafeteria or across a busy road. These dynamic requirements might not be always available for Wi-Fi Direct users in WDONs (i.e., since the users might be in static environments).

S. Ali et al. [68] presented a method for generating shared secret keys for body-worn health monitoring devices based on the motion of these devices. This method has no reconciliation phase and depends on the received signal strength indicator (RSSI), a measure of signal power in logarithmic units, to quantize the key. This method takes 15 to 35 minutes to generate a 128 bit key with a 75% chance of perfect agreement between endpoints. This long time to extract the key might be acceptable in health monitoring devices, but it is very long in case of Wi-Fi Direct communications in WDONs.

M. Miettinen et al. [69] presented an approach for secure zero-interaction pairing suitable for Internet-of-Things and wearable devices. Their scheme uses sensed context fingerprints to evolve the pairing key periodically in a way that is only possible for devices co-present over extended periods of time. They use readily available context sensor modalities like audio and luminosity. Their approach might not be suitable for mobile devices users in WDONs since it requires long time periods (i.e., hours).

Many modern smartphones are equipped with a wide variety of sensors including GPS, Wi-Fi, cellular radios (capable of positioning), accelerometers, magnetic compasses, gyroscopes, light, proximity, and cameras. These sensors work with the smartphone operating system or specific software applications to perform many functions. Current smartphone platforms allow developers and applications to access these hardware sensors and get a record or a log file of their generated data for a specific period of time. These sensors make smartphones attractive for collaborative sensing applications where phones cooperatively collect sensor data to perform various tasks. Researchers and mobile application developers have developed a wide variety of such applications. BikeTastic [71] and BikeNet [72] allow bicyclists to collaboratively map and visualize biking trails. In Transitgenie [73], a user installs an app on his smartphone and with the help of built-in sensors (i.e., GPS, Wi-Fi, and accelerometer), the app automatically detects when the user is riding a bike or a vehicle and sends periodic "location" updates to a central tracking server. In ACCessory [74], it is shown that accelerometer readings can be analyzed to sufficiently extract sequences of an entered text on smartphones. A background app, which depends on the accelerometer as a side channel, can be used to spy on keystroke information during some sensitive operations, e.g., account login.

2.2 Related Work For Evil Twin AP Attack

Wi-Fi Direct inherits its vulnerability to EvilDirect GO attack from the Wi-Fi infrastructure networks. In Wi-Fi infrastructure networks [75], the evil twin AP attack occurs mainly in free public Wi-Fi areas (e.g., airport lounges, cafes, hotel lobbies). As is shown in Figure 2.1, a malicious

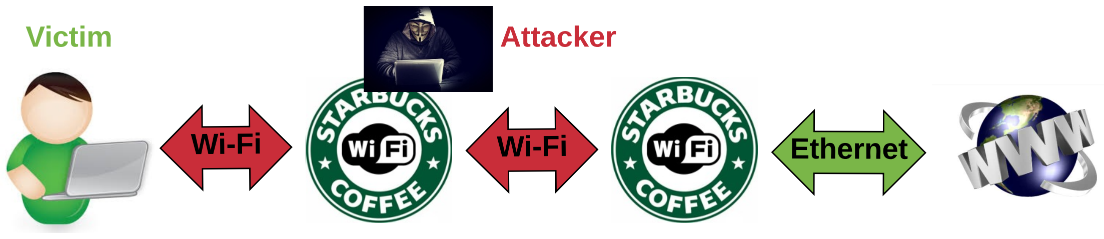


Figure 2.1: Evil Twin AP attack in Wi-Fi infrastructure networks.

attacker, Eve, can configure a laptop as a rogue access point (AP) to pretend to be the legitimate access point in a free public Wi-Fi area. Next, Eve sets her evil twin AP close the target victims. Eve can attract the victims' wireless connections, either through passively waiting or actively sending de-associate/de-authenticate frames to force victims to leave the legitimate AP. Eve can then relay victims' packets between her AP and the legitimate AP. Eve can provide internet access to victims and steal their personal information (i.e., without the need of creating additional connection to the internet such as wired or LTE). Accordingly, Eve essentially works as an "Evil Twin" AP between the victims and the legitimate AP. We discuss in the following the current solutions to this AP attack in Wi-Fi infrastructure networks and why they are not applicable to Wi-Fi Direct communications.

The existing rogue AP detection solutions can be classified into two categories: network administrator oriented (i.e., they are designed for a wireless network administrator to perform access control policies and authorization for the APs and clients), and user oriented (i.e., client-side solutions).

The network administrator solutions can be classified into two approaches. The first approaches proposed in [17–19, 76–83] detect the attacker by differentiating the traffic between the wired and wireless connections. If an unauthorized client uses a wireless network while it is not authorized to do so (i.e., by comparing with an authorized list), the AP attached to this client is classified as a rogue AP. [77] employs the round trip time (RTT) between the user and the DNS server to determine whether an AP is legitimate or not. [79] and [82] use packets inter arrival time (IAT) and RTT to distinguish between Ethernet and wireless clients. [80] assumes that the mean and variance of packets IAT is more random for a network path that has a wireless link as compared to a path that has only wired links. The [17] and [18] approaches propose algorithms that exploit the fundamental properties of the 802.11 CSMA/CA MAC protocol and the half duplex nature of wireless channels to differentiate Ethernet and WLAN TCP traffic.

The second network administrator oriented approaches proposed in [20–22, 84–86] work by monitoring Radio Frequency (RF) airwaves and additional information collected at routers/switches and compare that with a known authorized list. Jana et. al. [20] utilizes the fact that different APs usually have different clock skews to detect unauthorized APs. [21] locates suspicious APs by scanning RF airwaves from the Intranet and then compares that with specific "fingerprints" of the RF (with an authorized list). [22] uses desktop machines in monitoring by attaching them to USB-based wireless adapters. [86] uses sensors instead of sniffers to scan the RF.

The user oriented solutions proposed in [87] and [75] can detect an evil twin AP attack at the client side (i.e., without any additional support from the network administrator). CETAD [87] is designed based on the idea that the public IP address, ISP, RTT values of packets traveling through legitimate APs are similar (i.e., the same ISP), but they are different for a legitimate AP and an evil twin AP. Yang et. al. [75] proposes two algorithms to detect an evil twin AP from the client end based on server IAT. Their algorithms need a remote server within the LAN with their software installed for measuring server IAT.

Many wireless intrusion detection systems (WIDS) [23–25], which require a training phase and use sniffers, have been proposed to detect session hijacking attacks in wireless networks.

The current solutions to the evil twin AP attack in Wi-Fi infrastructure networks are not applicable to Wi-Fi Direct communications due to the following reasons:

1. There are no network administrators in Wi-Fi Direct networks (i.e., authorized list approaches are not working).

2. Differentiating the traffic at the wired and wireless connections is not possible (i.e., no wired connection in Wi-Fi Direct networks).

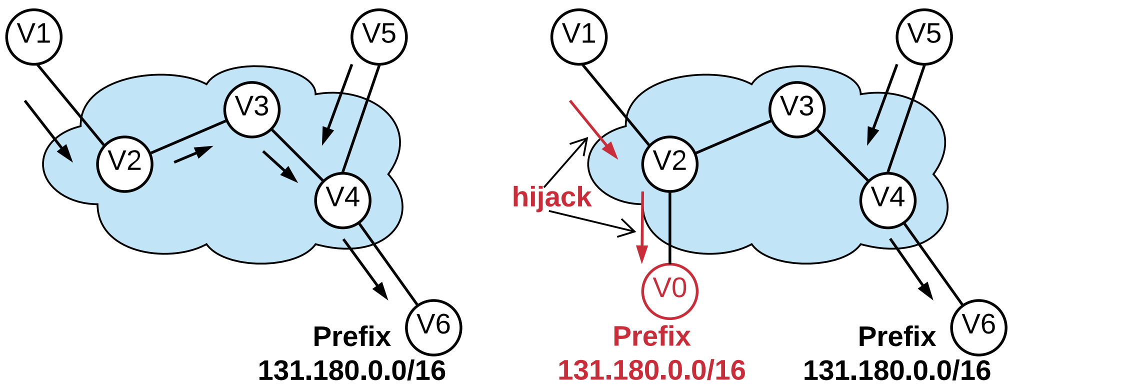
3. Intrusion detection systems, sniffers and sensors are not always available for Wi-Fi Direct devices.

4. The proposed solutions in [87] and [75] cannot be used because they depend on differentiating the wireless hops (one or two hops). However, in Wi-Fi Direct, the clients and the GO are connected by a single hop link.

2.3 Related Work For Route Hijacking Attack

Previous works on the security threats in opportunistic networks focused on flood, wormhole, and packet dropping attacks [35–39]. The adversary in the flood attack floods junk data into the network in order to deplete or overuse the limited network resources. The proposed approach in [35] employs rate limiting to defend against flood attacks. That is, each node has limits for the number of packets and replicas that it can generate in each time interval. The adversary in the wormhole attack records the packets at one location and tunnels them to another colluding node to corrupt the topology views of the network. The authors in [36] propose that the nodes reduce their transmission range for a short time to detect the presence of a forbidden topology structure that is caused by a wormhole attack. In the packet dropping attack, the adversary intentionally drops all or part of the received packets. The nodes in [37] [38] exchange signed contact records, based in which the next contact nodes can detect if the attacker has dropped any packet. In [39], a packet dropping is detected and traced back based on the Merkle tree. Indeed, none of the aforementioned works address the route hijacking attack in opportunistic networks.

However, most of the route hijacking attacks that have been addressed by recent research are in the internet, both considering Border Gateway Protocol (BGP) hijacking and interception attacks [88]. Each Autonomous System (AS) in the internet manages a number of networks, which can be expressed as IP prefixes. ASes use BGP to advertise their IP prefixes and establish inter-domain routes in the internet. BGP is a distributed protocol, lacking authentication of routes. Hence, a malicious AS can claim to own a prefix or sub-prefix that belongs to another AS causing redirection of routes from that AS to the attacker. As shown in Figure 2.2, all the packets with the IP prefix



(a) True origin AS V6 announces prefix 131.180.0.0/16 (b) False origin AS V0 announces prefix 131.180.0.0/16 and hijacks V6's route

Figure 2.2: Example of prefix hijack in the internet.

address 131.180.0.0/16 should be forwarded to AS V_6, however, a malicious AS V_0 can announce that this prefix belongs to it in order to hijack these packets. BGP hijacking detection approaches can be classified into four categories.

Control-plane approaches [40–44] collect BGP updates or routing tables from a distributed set of public BGP monitoring infrastructure and route collectors such as [45–47], and raise alarms when a change in the origin-AS of a prefix, or a suspicious route is observed. PHAS and Cyclops [42] [43] are notification systems that alert prefix owners (i.e., ISPs) when their BGP origin change. The network administrator in ARTEMIS [44] stores a configuration file that has an up-to-date list of all owned and announced prefixes. This list is continuously compared with the collected BGP updates from the monitoring services (i.e. [45–47]). Based on the result of the comparison, ARTEMIS can detect any hijacking event and generate alerts accordingly. The aforementioned control-plane approaches cannot be used against CollusiveHijack attack. Apart from the fact that there are no network administrators in WDONs, the nodes cannot create a list of "owned" or reached IP's a priori (i.e., the nodes in WDONs do not know the future contacts among each other).

Data-plane approaches [48] [49] continuously probe the internet to detect whether any data path changes. That is, by using pings/traceroutes to monitor the connectivity of a prefix and raise an alarm, when significant changes in the reachability [48] of a prefix or the paths leading to it [49] are observed. The Listen protocol [89] is a data plane verification technique that detects reachability problems in the data plane by passively probing network and checking whether the underlying routes to different destinations work. Indeed, due to the intermittent connectivity between the nodes in WDONs, the data-plane approaches cannot be used to detect CollusiveHijack attack (i.e., pings and traceroutes tools fail to monitor the connectivity of these networks).

Hybrid approaches [90–92] combine control and data plane information to detect the hijacking attack, however, they cannot detect CollusiveHijack attack due to the fact that they still need network administrators [92] and use monitor tools like pings and traceroutes [90] [91].

Cryptographic approaches [93–95] use Public Key Infrastructure (PKI) to ensure the authentication of routing announcements to minimize the risk of a single non-colluding hijacking. However, these approaches cannot defend against colluding ASes. S-BGP [95] and Whisper protocol [89] fail to detect colluding ASes that pretend the existence of a direct link between them by tunneling packets/advertisements unless the complete topology of the network is known and enforced. However, the topology of WDONs is dynamic and continuously changing. The BGPsec protocol specifications (as is shown in page 36 of [93]) state that detecting colluding ASes is beyond the scope of the BGPsec protocol.

3. SYSTEM AND SECURITY MODELS

In this section, we present the system and security (attacker) models for all proposed algorithms and protocols. Moreover, we present a real-world motivation scenario for the attacker in WDONs.

3.1 System Model

We assume a Wi-Fi Direct based opportunistic networks (WDON) that contains Wi-Fi Direct enabled mobile devices. Each device has an acceleration sensor and a sound sensor, each of which continuously senses the environment and stores its data into specific log files in the device. There is a group of clients (first responders in a disaster area, customers in cafes/restaurants, passengers in airport lounges, students in labs/libraries, etc.) interested in establishing a secure P2P group using Standard/Autonomous group formation mechanism of Wi-Fi Direct [1]. The P2P group might last for a couple of hours to share/stream images/videos, play games, print documents, etc. The clients are free to join or leave the group any time. In order to establish the secure P2P group, the mobile devices implement the Wi-Fi Protected Setup (WPS) [12] protocol and use the in-band configuration mode of operation. The mobile devices leverage the WPS protocol to create Diffe-Hellman (D-H) keys and derive from them their session keys (encryption and signing keys) in order to encrypt and sign the exchanged messages to achieve confidentiality and integrity. Our SekGens algorithm and EvilDirectHunter protocol aim to secure the PIN method and Push-Button method of the WPS protocol, as we will present in Sections 4 and 5, respectively.

In case the users want to use the PIN method of the WPS protocol, they enter the password which is obtained from our SekGens algorithm. The eight registration protocol messages for the Enrollee and Registrar devices (i.e., these are the names for the two Wi-Fi Direct devices that run the WPS protocol as is shown in Appendix A). In this section we use “the Enrollee and Registrar devices” and “the two devices” interchangeably. The messages that the two devices gradually exchange and verify each other are M_3 - M_7. If the verification of one part of the device password fails, the receiving node has to send a failure indication acknowledgment. Accordingly, both devices stop the protocol and discard all nonces and keys related to that session. If the two devices successfully exchange message M_8, this is considered as a proof-of-possession of the device password and they can securely exchange data messages. The users in the traditional WPS protocol need to enter the same PIN (maximum of eight numerical digits) at both devices to establish the secure connection. Messages M_3 and M_4, shown in Appendix A, contain the two hashed halves of the PIN. For example, if PIN value is "1234", PSK1 and PSK2 at the Enrollee and Registrar devices will result from the HMAC of "12" and "34", respectively. However, our SekGens changes the WPS protocol by replacing the two halves that are used to calculate PSK1 and PSK2 with the t-wo halves of the KEY_{Final}'s (i.e., the final key resulted from SekGens on both devices). Therefore, PSK1 and PSK2 are obtained by both Enrollee and Registrar devices as:

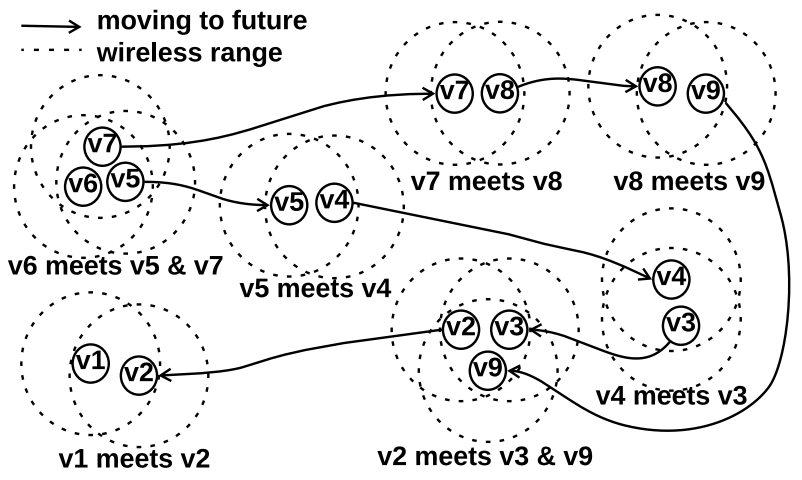
PSK1 = first 128 bits of $\text{HMAC}_{AuthKey}$(1st half of KEY_{Final})

PSK2 = first 128 bits of $\text{HMAC}_{AuthKey}$(2nd half of KEY_{Final})

SekGens algorithm runs before the WPS protocol. Apart from changing the way of obtaining PSK1 and PSK2, the remaining steps and messages of the registration protocol are not changed. An important assumption of SekGens is the existence of a collaboration phase, which has to be executed before it. During this phase, the two devices agree on the sensors involved in the calculation and their sampling rates. After that, SekGens will be ready to run on the two devices using the same number and type of sensors. Moreover, the sampling rate, sr_i, and the time periods, $[start_i, end_i]$, for each sensor (denoted as S_i) at the two devices will be the same.

In case the users want to use the Push-Button method, the client invites the GO to start the WPS protocol and waits for its acceptance. The client device includes its mode, i.e., Push-Button Configuration (PBC) in the invitation to notify the GO to enable its PBC mode. The GO's user either accepts or declines the invitation by pressing a logical button within 120 seconds (known as Walk Time). In case of acceptance, the two devices execute the eight registration protocol messages (M_1 to M_8 shown in Appendix A) and connect to each other.

In order to route the packets in the WDON, the mobile devices run either the HRP [26] or the Prophet [27] protocols. Both HRP and Prophet are probabilistic routing protocols that are based



Nodes	v2-v3-v9	v3-v4	v4-v5	v5-v6-v7	v7-v8	v8-v9
True cliam	1	1	1	1	1	1
False claim	1	1	1	1	***2***	***2***

Figure 3.1: Contact frequency for 24 hours. *Reprinted with permission from [53].*

on the fact that if two nodes have frequently contacted in the past, it is likely that they will contact again in the future. Both protocols leverage the aforementioned fact for reaching the disconnected nodes. As is shown in Figure 3.1, v_1 learns from the contact records, which are received from v_2, that either (v_2, v_3, v_4, v_5) or (v_2, v_9, v_8, v_7) can deliver v_1's packets to v_6 in the future. These intermediate nodes make use of the store-and-forward strategy to route v_1's packets. We assume that it is possible to achieve a time synchronization between all WDONs nodes at the scale of one second (the scale of one second is sufficient since the Inter-Contact Times (ICTs) in WDONs are at the scale of seconds/minutes). In the following paragraphs, we present HRP and Prophet in more detail.

1. **Hybrid Routing Protocol (HRP)** [26] [96] is a *limited* replication-based protocol that relies on the observation that path delay correlations can impact performance improvements gained from packet replication. HRP captures the potential correlation between the ICTs for different nodes and decides how much replication should be used for different network environments. HRP introduces two concepts: the *replication factor* and the *replication gain*. The replication factor is the total number of data copies created at the source for a given packet. If D_r is the random variable for routing delay when replication factor is *r*, then the

replication gain is $E[D_1]/E[D_r]$. The replication gain captures the benefit of replication in terms of delay improvement. HRP demonstrated mathematically and experimentally that the delay correlation affects the benefit of replication and it is important to capture it to estimate the replication gain and make better routing decision. HRP is implemented as a user-space daemon service [96].

2. **Prophet protocol** [27] [28] is an *unlimited* replication-based protocol that relies on a probabilistic metric called *delivery predictability*, $\Psi \in [0, 1]$. Ψ is established at each node indicating the probability of delivering a message to all other nodes. As shown in Figure 3.1, when v_6 encounters v_5, they exchange their Ψ's and update them accordingly. $\Psi_{(v_6,v_5)}$, which is v_6 delivery predictability for v_5, is updated according to $\Psi_{(v_6,v_5)}$ = $\Psi_{(v_6,v_5)_{old}}$ + (1 - $\Psi_{(v_6,v_5)_{old}}$) $\times$ Ψ_{init}. Ψ_{init} $\in$ [0,1] is a constant to ensure that nodes that frequently meet have high Ψ's. If two nodes do not meet for a while, their Ψ's must *age*. The aging equation for v_5 and v_6 is $\Psi_{(v_6,v_5)}$ = $\Psi_{(v_6,v_5)_{old}} \times \gamma^k$. $\gamma \in$ [0, 1) is a constant to decide how large impact the aging should have on Ψ and k is the number of time units that has elapsed since the last time Ψ was aged. Ψ also has a *transitive* property, as shown in Figure 3.1, which is based on the observation that if v_i frequently encounters v_j, and v_j frequently encounters v_k, then v_k is a good carrier to forward the messages to v_i. E.g., $\Psi_{(v_4,v_6)}$ = $\Psi_{(v_4,v_6)_{old}}$ + (1 - $\Psi_{(v_4,v_6)_{old}}$) $\times$ $\Psi_{(v_4,v_5)} \times \Psi_{(v_5,v_6)} \times \beta$. $\beta \in$ [0, 1] is a constant to decide how large impact the transitivity should have on Ψ. When v_i, which has a message for v_k, encounters v_j, then, v_i replicates its message to v_j only if $\Psi_{(v_j,v_k)} > \Psi_{(v_i,v_k)}$. Prophet protocol is implemented as a user-space daemon service [97] [98].

3.2 Attacker Models

Table 3.1 summarizes the types of attacks that are addressed in our security framework for W-DONs. An *Outsider* attacker is an unauthorized malicious device (i.e., not connected to WDONs) that attempts to launch either the brute-force/dictionary attack or the EvilDirect attack against the legitimate GO devices in WDONs. An *Insider* attacker is either a malicious device (i.e., that suc-

Table 3.1: Attacker models.

	Attacker		Target	
	Insider	**Outsider**	**Client**	**GO**
Brute-Force/Dictionary	✗	✓	✗	✓
EvilDirect	✗	✓	✗	✓
CollusiveHijack	✓	✗	✓	✓

cessfully launched an outside attack and join the network) or a compromised device (i.e., that is carried by a benign user). As is shown in Table 3.1, the insider attacker attempts to launch the CollusiveHijack attack against the clients and GO devices that run HRP or Prophet routing protocols in WDONs.

For the outsider attacker, the model assumes that a computationally bounded malicious attacker, Eve, is curious to discover the device password that is used between two devices in WDONs (in case the two devices use the PIN method) and is able to set up an EvilDirect GO with the same MAC address and SSID as the legitimate GO, and operates on the same channel (in case the devices use the Push-Button method). The best areas for Eve are public, indoor (airport lounges, cafes, restaurants, student community areas, libraries, etc.). Eve is free to move in the area but she cannot be very close to the legitimate GO or the clients (we only require that she is more than 1 m away from them). Moreover, Eve is unable to restrict other movements in the channel (passengers in airport lounges, customers/waiters in cafes, students in student community areas, etc.). We assume that Eve is not interested in launching a denial-of-service attack (DoS) or jamming the communication channel between the clients and the legitimate GO. Also, Eve is not equipped with full-duplex radio transceivers or directional antennas that enable the reception of a signal from the legitimate GO and jamming of the same signal at the clients, or vice versa. Eve is not interested in disrupting the execution of the SekGens algorithm or the EvilDirectHunter protocol. However, she can launch a replay attack against the EvilDirectHunter protocol as we will describe below. Consequently, our SekGens algorithm and EvilDirectHunter protocol do not address all man-in-the-middle attacks.

In case of brute-force/dictionary attacks, we assume that Eve knows our Session Key Generated from Sensors (SekGens) algorithm (i.e., we will present is Section 4) and the value of the nonce exchanged at the beginning of the algorithm. However, Eve has no access to the generated sensors' data on the two devices. Eve is able to sniff and capture the exchanged messages M_3 - M_7 that contain the two encrypted halves of the device password. Also, she is able to compromise the collaboration phase and discover the number and type of involved sensors, their sampling rate, and their time periods. Based on the assumptions described above, Eve can mount the following two types of attacks against the SekGens algorithm:

1. **Brute-force attack**: Eve can stimulate the Registrar to start the Diffie-Hellman exchange messages with it. Eve is able to capture message M_4, extract R-Hash1 & ENC(R-S1), and use brute-force to discover PSK1 (assuming that the 1st half of the device password is relatively short). By running the second round of the protocol without changing the key, the 2nd half of the device password can be discovered in the same way.

2. **Imitation attack**: Eve tries to generate the same sensors data of the Enrollee or Registrar devices and run SekGens to discover the device password. First, Eve adjusts the sampling rate and the time periods of its sensors to the same values used by the sensors of the pairing devices. Second, based on the sensor type, Eve strives to generate the same sensors data. For example, Eve can observe and imitate the gesture made by the victims while they generate data from their acceleration sensors. In case of sound sensors, Eve can (physically) get closer to the victims to have the same readings.

In case of EvilDirect attacks, we assume that Eve knows our EvilDirectHunter protocol (i.e., we will present is Section 5), but she has no access to the RSS profiles of the devices. Eve can listen to all communication between the clients and the legitimate GO. Also, she can measure both the channels between herself and the clients and between herself and the legitimate GO at the same time when the clients and the legitimate GO execute the Device Discovery and Service Discovery phases and build each other's RSS profile. Based on the assumptions described above, Eve can launch the following types of attacks against the EvilDirectHunter protocol:

1. **Physical proximity attack**: Eve aims to obtain the same RSS profile of the client or the legitimate GO in order to pass the detection of EvilDirectHunter. Eve tries to (physically) get close to the victims to have the same RSS readings.

2. **Predictable channel attack**: Another way for Eve to obtain the same RSS profile of the client or the legitimate GO is to predict the RSS readings at their physical locations to fake her RSS readings in dynamic and static environments.

3. $\oplus$ **result guessing attack**: The client and the legitimate GO divide the RSS profile that they build for each other into two subsets ($Rss_I = [rss_1, rss_2 \,..rss_k]$, $Rss_{II} = [rss_{k+1}, rss_{k+2}$ $..rss_{2k}]$). Then, they calculate the (Euclidean distance)^2, denoted as $\oplus$, between these two subsets. $\oplus$ measures the similarity between the two RSS profiles. Eve is able to launch an online/offline guessing attack against the result of $\oplus$ to pretend to be the legitimate GO.

4. **Client spoofing attack**: Eve can try to obtain the results of $\oplus$ by pretending to be one of clients. Before launching her attack, Eve waits until the client and the legitimate GO build each other's RSS profile. Then, she executes the EvilDirectHunter protocol with the legitimate GO to learn the results of $\oplus$ for that client. Finally, Eve launches her attack and passes the EvilDirectHunter protocol with that client since she is able to respond to its challenges.

5. **Replay attack**: Eve can overhear the messages between the clients and the legitimate GO (during the Device Discovery and Service Discovery phases) and replay all/some of these messages in order to influence their RSS profiles.

For the insider attacker, the model assumes that there are established pairwise keys between all devices using either our SekGens [51] (i.e., for 1-hop neighbor devices) or by leveraging our Efficient Pairwise Key Establishment Scheme (EPKES) [99] (i.e., for devices that are not within each other wireless range). Each node's key is only known by itself to guarantee the authentication and message integrity. There are two types of devices: non-compromised devices and *malicious/compromised devices*. A malicious attacker, Eve has successfully launched either the brute-force/dictionary attack or the EvilDirect attack and connected her devices to the WDON. Or, Eve has control of some compromised devices that is carried by benign users and infected by a malware a priori. We assume that Eve is not interested in launching a packet dropping or jamming attacks. However, Eve's nodes share each other's keys and lie about their ICTs. For example, as shown in Figure 3.1, v_1 has a packet for v_6 and the intermediate nodes (v_7, v_8, v_9), that are compromised, decrease their ICTs. That is, instead of informing v_1 that v_8 encountered v_9 one time during the last day, which is the truth, they claim that they encountered each other twice during the last day. The ICT between v_7 and v_8 is also decreased, as shown in Figure 3.1. We also assume that Eve knows our Path Detection Technique (PDT) and Hop Detection Technique (HDT) (i.e., we will present is Section 6).

In case of HRP, decreasing the ICTs of (v_8, v_9) and (v_7, v_8) improves their Ψ's to v_6 because HRP uses ICTs to measure nodes' delivery capabilities (i.e., HRP replicates the packets to the nodes with lower ICTs). In case of Prophet, Eve exploits the transitive and aging properties by decreasing the ICTs of its nodes to increase their Ψ's. As shown in Figure 3.1, decreasing the ICT of v_8 and v_9 resets k to 0 and increases $\Psi_{(v_9,v_8)}$ due to the aging property. As a result of the transitive property ($\Psi_{(v_9,v_6)}$ = $\Psi_{(v_9,v_6)_{old}}$ + (1 - $\Psi_{(v_9,v_6)_{old}}$) $\times$ $\Psi_{(v_9,v_8)}$ $\times$ $\Psi_{(v_8,v_6)}$ $\times$ β) increases as well. This makes $\Psi_{(v_9,v_6)} > \Psi_{(v_2,v_6)}$ and results in forwarding v_1's packets to v_8 and v_9. Based on the assumptions described above, Eve can launch the following types of attacks against PDT and HDT:

1. **Fake Ids attack against PDT**: Eve might pretend that two or more of her nodes have the same Id to not be detected by PDT. Without loss of generality, if v_2 and v_3 are compromised in $P_{s,d}$ = $\textcircled{s} \xleftrightarrow{\lambda_{s,v_1}} \textcircled{v_1} \xleftrightarrow{\lambda_{v_1,v_2}} \textcircled{v_2} \xleftrightarrow{\lambda_{v_2,v_3}} \textcircled{v_3} \xleftrightarrow{\lambda_{v_3,v_4}} \textcircled{v_4} \xleftrightarrow{\lambda_{v_4,d}} \textcircled{d}$, then v_3 pretends to be v_2. That is, Eve aims to hide v_3 identity to not be detected by PDT.

2. **Fake PRTs attack against HDT**: Eve's nodes might fake their PRTs to not be detected by

Table 3.2: Wi-Fi Direct Applications.

	Android app or Industrial/Research Product
Data Sharing	Wi-Fi direct file transfer [2] HiFi [3]
Video Streaming	Miracast [5]
Gaming	Binary2study [7]
Mobile Printing	HP Mobile Printing [6]
Traffic Offloading	QATO [4]

HDT. Without loss of generality, in $P_{s,d}$ = $\textcircled{s}\xleftrightarrow[\tau_1]{\lambda_{s,v_1}}\textcircled{v_1}\xleftrightarrow[\tau_2]{\lambda_{v_1,v_2}}\textcircled{v_2}\xleftrightarrow[\tau_3]{\lambda_{v_2,v_3}}\textcircled{v_3}\xleftrightarrow[\tau_4]{\lambda_{v_3,v_4}}\textcircled{v_4}\xleftrightarrow[\tau_5]{\lambda_{v_4,d}}\textcircled{d}$, where τ's are the links delays, if s sends a packet at τ_0, the PCT and PRTs for this packet = $[\tau_0, \sum_{i=0}^{1}\tau_i, \sum_{i=0}^{2}\tau_i, \sum_{i=0}^{3}\tau_i, \sum_{i=0}^{4}\tau_i, \sum_{i=0}^{5}\tau_i]$. These PCT and PRTs are respectively signed by $[(s), (s, v_1), (v_1, v_2), (v_2, v_3), (v_3, v_4), (v_4, d)]$. If Eve compromises v_2 and v_3 and fakes their contact frequencies to $2 \times \lambda_{v_2,v_3}$, she also can fake the PRT at v_3 to $\frac{\tau_3}{2}$ to not be detected by HDT on the $\textcircled{v_2}\xleftrightarrow[\frac{\tau_3}{2}]{2\times\lambda_{v_2,v_3}}\textcircled{v_3}$ hop. That is, Eve fakes the PRT at v_3 to match her claimed contact frequency, $2 \times \lambda_{v_2,v_3}$.

3.3 Real-World Motivation Scenario for the Attacker

In this section, we present a real-world motivation scenario for the attacker that launches the brute-force/dictionary attack, the EvilDirect attack, and the CollusiveHijack attack in WDONs.

For a group of users (e.g., friends, colleagues, students, family members, customers, first responders, etc.) that meet on a daily/weekly basis at different places/ events (e.g., work, restaurants, cafes, stadiums, schools, disaster areas, etc.), they might leverage different Wi-Fi Direct's apps or industrial/research products for different purposes as is shown in Table 3.2. In the following paragraphs, we present different use-case scenarios that illustrate why the attacker launches the aforementioned security attacks in WDONs.

1. **Brute-force/dictionary attack**: if a user, Alice, wants to set his device as a GO using the

Standard or Autonomous group formation mechanisms (that we will present in Section 5.1), then, Alice has to set an eight digit numeric PIN for his device. Also, Alice should tell his friends/colleagues about his PIN so they can connect to his device as clients and leverage Wi-Fi Direct application(s) (shown in table 3.2). Based on WPS specifications (page 42 of [12]), it is recommended that each time Alice wants to reestablish his P2P group in the future, he should delete the old PIN, generate a new one, and share it with his friends/colleagues. However, entering a fresh PIN each time is not convenient from Alice and his friends/colleagues' perspectives in terms of memorizing it and retrying according to wrong entries. Hence, Alice might store the P2P group's PIN so his friends/colleagues can directly and automatically connect to his device every time they meet. If the adversary, Eve, launch an online/offline brute-force/dictionary attack against Alice P2P group's PIN, as we will present in 4.1, she will be able to discover the PIN and launch the following attacks:

   (a) *Free data offloading attack*: in case that Alice wants to allow his friends/colleagues to freely use his cellular network's data plan, as is shown in [4], Eve is able to freely access the Internet via Alice device.

   (b) *Exploit flaws in benign apps*: some Android apps have buffer-overflow and data-base vulnerabilities [100] [101] that might be exploited by Eve to cause data leaks, to get a root access, or to inject/install malware into Alice's device.

2. **EvilDirect Attack**: If Alice wants to use the Push-Button method, his friends have to send invitation requests to his device. Then, Alice has to accept the invitation requests by clicking the "Accept" button on his device. However, as we will describe in 5.1, Eve is able to launch the EvilDirect attack and hijack the wireless communication of Alice's friends/colleagues. As a result, Eve is able to launch the following attacks:

   (a) *Breaking data confidentiality*: since the Wi-Fi Direct communications of Alice's friends/colleagues can be hijacked by Eve. Alice's friends/colleagues might send private data/images/videos to Eve (since they believe its Alice). Hence, Eve breaks the data confidentiality of Alice's friends/colleagues in WDONs.

(b) *Exploit flaws in benign Apps*: Same as mentioned above.

3. **CollusiveHijack Attack**: If Eve successfully launched the brute-force/dictionary attack or the EvilDirect attack, she will have a set of malicious devices in the WDON. Then, she can launch the CollusiveHijack attack and hijacks the packets of the legitimate nodes in the network, as we described in Section 3.1. As a result, Eve is able to launch the following attacks:

   (a) *Packet modification attack [30]*: After receiving the packets, Eve can (selectively) modify their contents to waste other nodes' resources (e.g., power and buffer) and network transmission bandwidth. Obviously, this also affects the packet delivery ratio of the HRP and Prophet routing protocols [31].

   (b) *Eavesdropping and traffic analysis attacks [32] [33]*: As a passive eavesdropper, Eve can feasibly identify types of network traffic and the applications that generated this traffic [33]. This may reveal sensitive information about a user like medical conditions, hobbies, preferences, etc.

   (c) *Incentives seeking attack*: If an incentive mechanism for routing [34] is employed and the nodes compete for these incentives, the malicious/ compromised nodes can deliver more packets in the network and get credit and reputation from the source nodes.

4. SECURING THE PIN METHOD IN WDONs*

In this section, we first show the vulnerability of Wi-Fi Protected Setup (WPS) [12] protocol to a brute-force or a dictionary attack [13]. This vulnerability depends upon the device password's length. Even though it is recommended to use an eight digit numeric PIN, this length does not guarantee the large amount of entropy that is needed for strong mutual authentication. That is why it is recommended to use a fresh PIN at each running time of the registration protocol. Entering a fresh and long PIN each time is not convenient from the user's perspective in terms of memorizing it and retrying according to wrong entries. Nevertheless, if the user uses the same keys many times or enters short keys, the attacker's task of compromising the connection becomes easier.

Next, in order to address the vulnerability of WPS [12] protocol to a brute-force/dictionary attack, we propose the idea of using contextual information (i.e., data obtained from mobile device's sensors) to establish a long (128 bits) secure session key between two Wi-Fi Direct enabled devices in WDONs, instead of using the keypad. Our solution, Session Key Generated from Sensors (SekGens) employs three phases. In the *Quantization Phase*, the key is iteratively generated based on different sensors' data. In the *Reconciliation Phase*, the two devices eliminate minor differences in the bits of their keys by using the Cascade reconciliation mechanism. In the *Privacy-Amplification-and-Hashing Phase*, the two devices omit all bits exposed during the reconciliation phase and apply hashing to the remaining secret bits to generate the final secret 128 bits key.

4.1 Background and Motivation

In Wi-Fi network, the role of each device in the network is either a client or an AP. Different sets of functionalities are assigned to each role. A major feature of Wi-Fi Direct [1] is that these roles in it are dynamic. Any Wi-Fi Direct device has to implement the role of client and the role of AP (soft AP). These, are just logical roles that could be executed simultaneously by a single device

*Reprinted with permission from "On secure shared key establishment for mobile devices using contextual information" by Ala Altaweel, Radu Stoleru, and Subhajit Mandal, In Proceedings of 2015 IEEE 34th International Performance Computing and Communications Conference (IPCCC), pages 1-10, Nanjing, China, 2015, Copyright 2015 by IEEE.

either by using different frequencies or time-sharing the channel via virtualization techniques.

Wi-Fi Direct devices agree on these roles by forming P2P groups, which are equivalent in term of functionalities to the Wi-Fi infrastructure networks. The Wi-Fi Direct devices, which are also known as P2P devices, are classified as: P2P Group Owner (P2P GO) and P2P clients. P2P GO is the device that implements the AP-like functionality in the P2P group. Other devices, which act as clients, are called P2P clients. There are many mechanisms for the two devices to establish a P2P group. The selected mechanism depends on either the negotiation of P2P GO's role, or if there is any pre-shared security information available. These group formation mechanisms are: Standard formation (the most complex way) and other two simple mechanisms (Autonomous and Persistent).

The group formation mechanisms consist of different phases, which are shared and executed by the two devices: Discovery, Wireless Protected Setup (WPS) Provisioning [12], and Address Configuration. The Standard formation mechanism has an additional phase, the GO Negotiation. WPS, which is implemented by all mechanisms, aims at supporting a secure connection between devices. WPS is developed by the Wi-Fi Alliance as an optional certification program to simplify the setup of security-enabled Wi-Fi networks in small areas like offices. WPS has two modes of operation: in-band and out-of-band configurations. In case of in-band mode, a Diffie-Hellman key exchange is accomplished and authenticated via a shared secret password. This password is obtained from the Enrollee and it is entered into the Registrar using a keypad entry. In case of out-of-band mode, WLAN credentials are sent across an out-of-band channel, e.g., USB flash drive, to the Enrollee.

In this research, we focus on the in-band configuration mode of operation using a shared secret key that is entered manually using the keypad. The goal of registration Protocol messages (M3-M7, described in Appendix A) is to incrementally prove the mutual knowledge of the device password between the two devices. When the Registrar and Enrollee devices have proven knowledge of the password, the encrypted configuration data is exchanged. Cryptographic protection for the messages depends on a key derivation key (KDK) [12], which is computed from the nonces, Diffie-

Hellman secret, and MAC address of Enrollee.

We motivate our research by demonstrating a brute-force attack against Wi-Fi Direct. An inherent flaw in the WPS protocol makes the key discovery easier [13]. There are two ways of cracking the authentication key. Tools like Reaver [102] can crack the PIN in less than 4 hours. On average an attacker can succeed in around 2 hours, this is real-time online way of cracking the WPS PIN. Another method is where the attacker takes a passive strategy by sniffing the packet transmissions M3-M7 that take place between the two devices. Then run an offline tool like aircrack-ng [103] that can look up a dictionary and recreate the PIN used by the Enrolee and Registrar devices. Depending on the configuration of the machine used to run the penetration tool and the complexity of the PIN it may take between 4-10 hours to discover the PIN.

We used an offline method to discover the PIN used by the Registrar to authenticate the Enrolee. On a low end notebook (i.e., without sophisticated hardware capabilities), we configured the wireless interface in monitor mode. Then we used a packet capturing tool, Wireshark to capture the packet transmission between the Registrar and Enrolee, run on two LG Optimus Black smartphones. We generated an exhaustive list of possible PINs and stored it in a dictionary. We used airodump-ng to find the BSSID of our Wi-Fi Direct group. Just providing our prepared PIN dictionary and the BSSID we are interested in, the aircrack-ng tool was able to find out the PIN in just over 3 hours. With a more powerful system the time can be reduced further.

The WPS protocol was introduced by Wi-Fi Alliance in 2007 to enable secure pairing of Wi-Fi devices with compatible APs. The security flaw in its design, which causes the PIN brute force vulnerability, was discovered by S. Viehböck [13] in 2011. At that time, the WPS protocol was used in the wireless routers. S. Viehböck recommended to disable the WPS protocol on the APs.

The WPS PIN brute force vulnerability has been studied by some security researchers. A. Sanatinia et al. [15] studied the spreading of Wi-Fi APs infections using WPS flaws. They proposed new designs and frameworks for APs to allow more scalable and flexible administration to avoid the spreading of APs infections. D. Zisiadis et al. [104] proposed an enhancement for the WPS protocol by introducing the ViDPSec, a user-based device that runs a paring protocol relying on human visual out-of-band verification. ViDPSec requires from the AP to be equipped with a small LED to display the PIN and SUM values and a button for SUM acknowledgement. Through six steps, ViDPSec requires the users to acknowledge a small number through the visual channel, instead of typing the PIN.

Due to the unavailability of wireless network administrators in adhoc networks, the proposed solution in [15] can not solve the WPS brute-force attacks for Wi-Fi Direct devices. The ViDPSec device [104] is designed for the wireless routers and it is not convenient for smartphone users due to the six steps needed and the need to carry this additional device. Indeed, many intrusion detection systems such as Waidps [105] can detect WPS brute-force attacks. However, these intrusion detection systems are not always available for Wi-Fi Direct devices.

SekGens is a convenient and usable approach for smartphone users, which aims at solving the PIN brute force vulnerability in the traditional WPS, by establishing a 128 bits secure session key between two Wi-Fi Direct enabled smartphones. In the following Section, we present the Near Field Communication (NFC) technology and discuss the following valid question: why we don't leverage the NFC technology in our SekGens algorithm?

4.2 Leveraging Near Field Communication (NFC) in SekGens

Near field communication (NFC) [106] [107] is a proximity communication technology that is based on RFID radio with 13.56 MHz frequency and an operating distance of $\sim$10cm [108]. NFC is used to exchange a small amount of data or to authenticate other technologies, e.g., Bluetooth [109]. NFC has numerous applications like contact-less payment applications, NFC enabled credit card systems, e-passports, and smart card access tokens [110–113].

There are two entities participate in any NFC communication, the initiator and the target. The initiator (or the reader) starts the communication by selecting a target device. Then, the target (or the tag) responds with its data. There are two forms of the NFC communication (passive and active). In the passive form, the initiator generates its own power and the target is powered by the RF field of the initiator. That is, when the target is within the RF range of the initiator, the target uses the power from the initiator's RF field to communicate. The target returns to idle state when it is out of range of the initiator. In the active form, the target and the initiator generate their own power and intermittently communicate by turning their RF field on and off to send and receive data, respectively.

NFC security is based on the assumption that, since the devices have to be physically close to each other, it is not practical to launch security attacks like Man-in-the-Middle or eavesdropping [109]. However, many research [107], [109], [114], [115] has demonstrated that the eavesdropping range of NFC communication can be extended to 5-10 meters. This increase in the distance range of which the NFC communication can be overheard is exploited to launch security attacks that had been previously excluded (i.e., due to lack of proximity). For example, works like [106, 114–116] demonstrated that attacks like eavesdropping, relay, and DoS can be easily and successfully launched with minimal additional equipment. The Signal Amplification Relay Attack (SARA) [117], which is based on the idea of "relaying" the RFID radio signal across a longer distance, enables the attacker to break and even start cars with key-less entry systems. The researchers found that there are 24 vehicles from 19 different manufactures vulnerable to SARA [117].

Eavesdropping attacker requires an antenna (i.e., or an additional reader) to overhear the NFC communication. Hence, the attacker can compromise the NFC users' privacy by obtaining sensitive information (i.e., if the NFC information is not encrypted). Relay attacks require a malicious (i.e., a proxy) initiator to communicate with the legitimate target and relay the information to a malicious target, which communicates with the legitimate initiator. That is, relaying the communication between initiators and targets that are not within each other NFC communication range (i.e., invalidating the basic assumption of close proximity of NFC devices). Accordingly, the attacker establishes an NFC connection and processes transactions between a reader and an NFC enabled credit card of a legitimate owner. DoS attacks can be accomplished by corrupting the data via an additional RF field that transmits random information at 13.56 MHz or by keeping the initiator or target busy (i.e., hence they can not establish NFC communication with each other [109], [116]). To address the aforementioned security attacks against NFC devices, a suite of standards known as NFC-SEC have been proposed to achieve a secure communication between these devices [118–122].

The NFC-SEC-01 and NFC-SEC-02 standards [119], [120] leverage the Elliptic Curve Diffie-Hellman (ECDH) protocol to establish a secure channel and encrypt the exchanged data via the Advanced Encryption Scheme (AES). The NFC-SEC-03 standard [121] is based on asymmetric key cryptography (i.e., it requires certifications that are provided by a trusted third party). Differently, the NFC-SEC-04 standard [122] is a symmetric key cryptographic scheme that requires a pre-shared key to be established (i.e., manually entered) between the two devices a priori.

Due to the limited practicality of NFC-SEC-03 and NFC-SEC-04 (i.e., unavailability of the trusted third party and the pre-shared keys in WDONs), we limit our discussion to NFC-SEC-01 and NFC-SEC-02 standards. As we mentioned above, in these standards, the exchanged data is encrypted using a shared secret key that is derived by the Elliptic Curve Diffie-Hellman (ECDH) protocol. Hence, the eavesdropping attack is prevented. However, in case of Man-in-the-Middle or Relay attacks, there is no mechanism to verify if the communicating device is authentic or a malicious entity. Hence, NFC-SEC-01 or NFC-SEC-02 requires a method to authenticate the two communicating devices (i.e., similar to Wi-Fi Direct protocol which employs the WPS protocol to authenticate the Diffie-Hellman key establishment via the PIN method as we described in Section 4.1). In other words, NFC technology can not be used to authenticate two Wi-Fi Direct devices since its security standards require additional mechanisms in order to prevent the Man-in-the-Middle or Relay attacks [107].

4.3 Session Key Generated from Sensors (SekGens) Design

The SekGens algorithm has three phases: Quantization, Reconciliation, and Privacy- Amplification -and-Hashing. Both Enrollee and Registrar run quantization and privacy-amplification-and-hashing phases independently. The reconciliation phase, however, involves exchanging of specific messages between the two devices. Before we give a detailed description of each phase of SekGens. In the following, we present the notations that we use in describing these phases (i.e., in Table 4.1) and the definitions of the Pearson's correlation coefficient, the Shannon Entropy, and the normalization of a set of numbers:

Table 4.1: Notations and definitions for SekGens. *Reprinted with permission from [51].*

E: The Enrollee device.
R: The Registrar device .
N: Total number of shared sensors between the two devices.
J: Number of iterations in the quantization phase.
P: The labeled period ([0,100]). Each sensor labels a sub-period of P based on its entropy value.
NC: Nonce sent from Enrollee to Registrar.
$\mathbf{S}_{\boldsymbol{i}}$: Sensor number *i*, $1 \leq$ i $\leq N$.
$\boldsymbol{[start_i, end_i]}$: Time period of sensor S$_i$ (in sec).
$\mathbf{S}_i\boldsymbol{[start_i, end_i]}$: Data of sensor number *i* during the time period: $[start_i, end_i]$.
$\mathbf{KEY}_{\boldsymbol{Q}}$: The raw key generated from the quantization phase (320 bits).
$\mathbf{Key}_{\boldsymbol{j}}$: Key generated from iteration number *j*. Its size depends on J (i.e., concatenating all Key$_j$'s creates KEY$_Q$).
$\mathbf{KEY}_{\boldsymbol{Rec}}$: The reconciled key resulted after the reconciliation phase (320 bits).
$\mathbf{KEY}_{\boldsymbol{Final}}$: The final key resulted after the privacy-amplification-and-hashing phase (128 bits).
$^1\boldsymbol{\rho(X, Y)}$: Pearson's Correlation Coefficient between two sets of data X and Y.
$^2\boldsymbol{H(S_i[start_i, end_i])}$: Shannon Entropy of sensor S$_i$ during the time period: $[start_i, end_i]$.
$^3\boldsymbol{Norm(x_1, x_2, x_3 ... x_n)}$: Normalize a set of numbers $(x_1, x_2, x_3, ...x_n)$ into the period P.
$^4\boldsymbol{Label(P, x_1 ... x_n)}$: Divide the period P into different sub-periods and label each one with a corresponding number x_i.
A || B: Concatenation A and B bit-strings.
r = GenNum(s): Generate a random number *r* : $0 \leq r \leq 100$ using the uniform distribution function with seed = *s*.
$\mathbf{sr}_{\boldsymbol{i}}$: Sample rate of sensor number *i* (in samples/sec).
GenRandData($\boldsymbol{[start_i, end_i]}$**, s,** $\boldsymbol{sr_i}$**)**: Generate random data in the $[start_i, end_i]$ period using the uniform distribution function with seed = *s* and sample rate = sr$_i$.
$[\boldsymbol{d_{start_i}}, \boldsymbol{d_{end_i}}]$: Random data in the $[start_i, end_i]$ period.
GenKey(s): Generate a key for specific iteration (say *j*, Key$_j$) randomly using the uniform distribution function with seed = *s*. Each bit is generated independently.

- [1] The Pearson's correlation coefficient between X and Y sets of data, where $\overline{X}$ and $\overline{Y}$ are the mean values of X and Y respectively, is defined as:

$$\rho(X,Y) = \frac{\sum_{i=1}^{n}\left[(X_i - \overline{X}) \times (Y_i - \overline{Y})\right]}{\sqrt{\sum_{i=1}^{n}(X_i - \overline{X})^2} \times \sqrt{\sum_{i=1}^{n}(Y_i - \overline{Y})^2}} \tag{4.1}$$

- [2] The Shannon Entropy of sensor S$_i$ during the $[start_i, end_i]$ time period, where *n* is the total number of sampled data and $p(y_i)$ is the probability of sample y_i, is defined as:

$$H(S_i[start_i, end_i]) = -\sum_{i=1}^{n} p(y_i) \times log_2 p(y_i) \tag{4.2}$$

- [3] To normalize a set of numbers: $(x_1, x_2, x_3, ...\ x_n)$ into the period *P*, each x_i is normalized to a sub-period of *P* as:

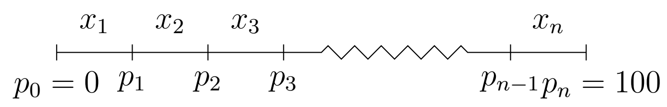
  $x_1 \xrightarrow{normalized\ to} [p_0, p_1],\ \ x_2 \xrightarrow{normalized\ to} [p_1, p_2],$

  $x_3 \xrightarrow{normalized\ to} [p_2, p_3], ...\ x_n \xrightarrow{normalized\ to} [p_{n-1}, p_n].$

  Where p_i's are calculated as, let $\Lambda = \frac{100}{\sum_{i=1}^{n} x_i}$,:

  $p_0 = 0, p_1 = p_0 + x_1 \times \Lambda, p_2 = p_1 + x_2 \times \Lambda, ...\ p_n = p_{n-1} + x_n \times \Lambda$

- [4] After the normalization, each sub-period (p_{i-1}, p_i), s.t. $1 \leq i \leq n$, is labelled by x_i as:



4.3.1 Quantization Phase

The quantization phase aims at generating the raw key KEY$_Q$ at the Enrollee and Registrar devices independently. This phase starts when the Enrollee sends a random nonce *NC* to the Registrar before the eight messages of WPS protocol described in Appendix A. In SekGens, *NC* is used as a seed input for the uniform distribution function (*GenNum()*). Each time *NC* is used, it is incremented by one.

Algorithm 1 Quantization.

```
1:  KEY$_Q$ = $\Phi$
2:  for $i$ = 1 to $J$ do
3:      Key$_j$ = $\Phi$
4:  for $i$ = 1 to $N$ do
5:      find $H(S_i[start_i, end_i])$
6:  P = Norm ($H(S_1[start_1, end_1])$, $H(S_2[start_2, end_2])$, $H(S_3[start_3, end_3])$ ....
    $H(S_N[start_N, end_N])$)
7:  Label(P[0,100], $S_1, S_2 ... S_N$)
8:  for j = 1 to J do
9:      r = GenNum(NC)
10:     Locate r in P, then select the corresponding sensor (say $S_k$)
11:     $[d_{start_k}, d_{end_k}]$ = GenRandData($[start_k, end_k], NC, sr_k$)
12:     coef = $\rho(S_k[start_k, end_k], [d_{start_k}, d_{end_k}])$
13:     Key$_j$ = GenKey(coef)
14:     KEY$_Q$ = KEY$_Q$ $\|$ Key$_j$
```

Algorithm 1 shows the pseudo code of the quantization phase. Lines 1-3 set the KEY$_Q$ and all other keys Key$_j$'s, which are generated from each iteration *j*, to null. This initialization is necessary to delete the previous session keys each time SekGens is initiated. The entropy $H(S_i[start_i, end_i])$ of each sensor *i* is calculated as shown in line 5. These entropy values are used to decide which sensor is selected to generate Key$_j$ in each iteration. The sensor, which has a higher entropy for its data, has a higher probability to be involved in generating the KEY$_Q$. Line 6 normalizes all entropy values to be in the period *P*. Depending on the calculated entropy values, each sensor labels (line 7) a sub-period of *P* as it is illustrated in Table 4.1. Assuming that there is a high level of similarities between the sensors data of both devices, the labeling of the *P* period on both devices is likely the same. Based on the normalization principle, the sensor with high entropy has a larger sub-period. As a result, sensors with high entropy values are more likely to be selected in each iteration for generating the partial key Key$_j$ as a part of KEY$_Q$. Involving more sensors with high entropy values increases the randomness of the generated bits of KEY$_Q$.

Lines 9-14 are responsible for generating the KEY$_Q$ by creating Key$_j$ in each iteration *j*. Firstly, a specific sensor, which is supposed to be the same on both devices, is selected. By using the same *NC* as an input to *GenNum()* function in line 9, both devices will generate the same random number *r*. In line 10, the two devices find the sub-period of *P* that contains *r*, then they select the corresponding sensor S_k, which labels that sub-period. Generating random data (line 11) in the period $[start_k, end_k]$ using the shared nonce *NC* and based on the sampling rate sr_k, will be the same on both devices.

The correlation coefficient, *coef*, is calculated in line 12 after performing the correlation between the random data $[d_{start_k}, d_{end_k}]$ and the sensor data $S_k[start_k, end_k]$ in the period $[start_k,$ $end_k]$ on both devices. We assume that both devices have already agreed upon the same time interval $[start_k, end_k]$ in the collaboration phase and they share the same random data $[d_{start_k}, d_{end_k}]$ (line 11). Then if their sensors data $S_k[start_k, end_k]$ for sensor S_k is likely the same, then both devices compute the same *coef*. At each iteration in line 13, Key_j is generated using the *GenKey(coef)* function seeded with *coef*. The length of this key depends on the number of iterations *J*, SekGens produces 320 bits from the quantization phase. To increase the chance of involving more sensors in the quantization phase, we set *J* to 160 with the length of each Key_j = 2 bits. We justify in Section 4.4.2.1 why we need to generate 320 bits. Line 14 cumulatively concatenates all Key_j's to form the 320 bits of KEY_Q.

4.3.2 Reconciliation Phase

Reconciliation mechanisms can be used by any communicating devices to eliminate any minor bit-differences in their data using the Cascade protocol [123]. For our SekGens, the reconciliation is the second phase. Similar to the concept of cyclic redundancy check, the two devices have to exchange meta-data to identify any mismatching bits and reconcile their bit-strings. Meanwhile, they attempt to minimize the potential leakage of information about their bit-strings to an eavesdropper. When any mismatching bits are found, the two devices correct them accordingly.

The Cascade protocol aims at fixing the different bits in two n bit-strings ($n > 2$) by performing multiple parity checks and shuffling the bit sequence before each pass. The main part of the Cascade is the Binary protocol, which can be applied to any two equal length bit-strings ($s_1 \& s_2$) with different parities. By recursively applying the divide-and-conquer strategy, the Binary protocol keeps comparing the parities of the two bit-strings and is able to detect and correct one erroneous bit.

Before starting the Binary protocol, both devices have to find out if there are any differences in their KEY_Q's. Each device divides its KEY_Q into a number of blocks and calculates the even parity of each block, the size of each block is called the initial block size (denoted as k_0). The Enrollee sends the even parity of its first block and the Registrar compares that to the even parity of its first block, if they are different then the same process is applied to the successive halves of the block at both devices until the location of the erroneous bit is narrowed down by a binary search. If the parity bits are equal, the registrar sends the parity of its second block. The same process is repeated on all blocks, which we call the first pass of the reconciliation. For any two blocks (each of size n) that have different parity bits, the total number of exchanged messages needed to correct one erroneous bit is $\log_2(n)$ + 1.

In order to solve the masked error cases (i.e., two different blocks with equal even parity bits), the Enrollee and Registrar devices run one additional pass after randomly shuffling their KEY_Q's using the same random shuffling function seeded with *NC*. The initial block size for SekGens (i.e., k_0) is 8 bits and the block size for the second pass is 16 bits.

Algorithms 2 and 3 show the reconciliation steps of one pass at the Enrollee and Registrar devices, respectively. K represents the total number of blocks (i.e. if k_0 = 8 bits, $K = (320/8)$ blocks). blk_num variable contains the number of main blocks being reconciled by both devices (for our last example, $0 \leq blk\_num < 40$). Moreover, each device has a pointer $(current\_block)$ to the current block/sub-block being reconciled. blk_ord contains the first and last bit-index of $current\_block$, $position$ function returns these indexes.

The Enrollee device starts the protocol (line 5) by sending the parity bit, block number and block order of $block_0$ ($\langle pb\ of\ current\_block,\ blk\_num,\ blk\_ord \rangle$) to the Registrar device. The Registrar checks (line 13) the parity bit of its first block ($block_0$) and finds out if it is different from the received parity ($rcvd\_pb$), if so, the Registrar sends the parity bit of the left significant half (LSH) of its $current\_block$. Otherwise, it increments the block number to point to the next block and it sends its parity bit to the Enrollee.

Algorithm 2 Reconciliation of one pass at the Enrollee.

```
1:  $blk\_num = 0$
2:  $K = total\ number\ of\ blocks$
3:  $current\_block \longrightarrow block_{blk\_num}$
4:  $blk\_ord \longrightarrow position(current\_block)$
5:  send $\langle pb\ of\ current\_block,\ blk\_num,\ blk\_ord\rangle$ to $R$
6:  while $blk\_num < K$ do
7:      wait for $\langle rcvd\_pb,\ rcvd\_blk\_num,\ rcvd\_blk\_ord\rangle$ from $R$
8:      switch ($position(rcvd\_blk\_ord)$)
9:      case $\langle 1\ bit\rangle$:
10:         correct the erroneous bit
11:         $blk\_num = blk\_num + 1$
12:         $current\_block \longrightarrow block_{blk\_num}$
13:     case $\langle block_{blk\_num+1}\rangle$:
14:         $blk\_num = blk\_num + 1$
15:         $current\_block \longrightarrow block_{blk\_num}$
16:         if error in $current\_block$ then
17:             $current\_block \longrightarrow LSH\ of\ current\_block$
18:         else
19:             $blk\_num = blk\_num + 1$
20:             $current\_block \longrightarrow block_{blk\_num}$
21:     case $otherwise$:
22:         $current\_block \longrightarrow position(rcvd\_blk\_ord)$
23:         if error in $LSH\ of\ current\_block$ then
24:             $current\_block \longrightarrow LSH\ of\ current\_block$
25:         else
26:             $current\_block \longrightarrow MSH\ of\ current\_block$
27:     end switch
28:     $blk\_ord \longrightarrow position(current\_block)$
29:     send $\langle pb\ of\ current\_block,\ blk\_num,\ blk\_ord\rangle$ to $R$
```

If two blocks have different parities, both devices narrow down the position of the erroneous bit by using binary search (**case** *otherwise*), until it is located and corrected (**case** $\langle 1\ bit\rangle$). Both devices run each pass of the protocol on all blocks (until $blk\_num = K$). When moving to next block (**case** $\langle block_{blk\_num+1}\rangle$) and in case of erroneous bit, the receiving device can arbitrary select to send the parity bit of either the left significant half (LSH) or most significant half (MSH)

Algorithm 3 Reconciliation of one pass at the Registrar.

```
1:  $blk\_num = 0$
2:  $K = total\ number\ of\ blocks$
3:  $current\_block \longrightarrow block_{blk\_num}$
4:  $blk\_ord \longrightarrow position(current\_block)$
5:  while $blk\_num < K$ do
6:      wait for $\langle rcvd\_pb,\ rcvd\_blk\_num,\ rcvd\_blk\_ord \rangle$ from $E$
7:      switch ($position(rcvd\_blk\_ord)$)
8:      case $\langle 1\ bit \rangle$:
9:          correct the erroneous bit
10:         $blk\_num = blk\_num + 1$
11:         $current\_block \longrightarrow block_{blk\_num}$
12:     case $\langle block_{blk\_num} \rangle$:
13:         if error in $current\_block$ then
14:             $current\_block \longrightarrow LSH\ of\ current\_block$
15:         else
16:             $blk\_num = blk\_num + 1$
17:             $current\_block \longrightarrow block_{blk\_num}$
18:     case $\langle block_{blk\_num+1} \rangle$:
19:         $blk\_num = blk\_num + 1$
20:         $current\_block \longrightarrow block_{blk\_num}$
21:         if error in $current\_block$ then
22:             $current\_block \longrightarrow LSH\ of\ current\_block$
23:         else
24:             $blk\_num = blk\_num + 1$
25:             $current\_block \longrightarrow block_{blk\_num}$
26:     case $otherwise$:
27:         $current\_block \longrightarrow position(rcvd\_blk\_ord)$
28:         if error in $LSH\ of\ current\_block$ then
29:             $current\_block \longrightarrow LSH\ of\ current\_block$
30:         else
31:             $current\_block \longrightarrow MSH\ of\ current\_block$
32:     end switch
33:     $blk\_ord \longrightarrow position(current\_block)$
34:     send $\langle pb\ of\ current\_block,\ blk\_num,\ blk\_ord \rangle$ to $E$
```

Algorithm 4 Error-Estimation at the Enrollee & Registrar.

```
1:  if Enrollee then
2:      send $\langle POS,\ E\_subsets \subseteq KEY_{Rec}\rangle$ to $R$
3:      wait for $R\_subsets$ from $R$
4:      calculate $e_{obs}$
5:      if $e_{obs} > 0.0$ then
6:          abort
7:      else
8:          continue to Privacy-Amplification-and-Hashing phase
9:  if Registrar then
10:     wait for $\langle POS,\ E\_subsets \subseteq KEY_{Rec}\rangle$ from $E$
11:     send $\langle R\_subsets \subseteq KEY_{Q}\rangle$ to $E$
12:     calculate $e_{obs}$
13:     if $e_{obs} > 0.0$ then
14:         abort
15:     else
16:         continue to Privacy-Amplification-and-Hashing phase
```

of that block, we choose to send the (LSH) of the block (lines: 17 & 22 in Algorithms 2 & 3, respectively).

When both devices finish the two passes, they have to find out whether their KEY$_{Rec}$'s are equal or not. Therefore, they run the last sub-phase of the reconciliation (Error-Estimation). The estimating of the discrepancy between their KEY$_{Rec}$'s is done by exchanging random subsets of bits picked randomly from their KEY$_{Rec}$'s. The Enrollee starts the error-estimation sub-phase (Algorithm 4) by sending a random subsets ($E\_subsets$) of its KEY$_{Rec}$ and their positions (POS) to the Registrar. The Registrar replies by sending its subsets ($R\_subsets$) from the same positions of its KEY$_{Rec}$. The size of each subset is 32 bits. After that, both devices calculate the observed error rate (e_{obs}). If e_{obs} is greater than zero, then both devices stop establishing the secure channel. Else, they pursue by starting the privacy-amplification-and-hashing phase.

4.3.3 Privacy-Amplifcation-and-Hashing Phase

This phase aims at improving the privacy of the KEY$_{Rec}$'s in the Enrollee and Registrar devices by omitting all corrected bits during the reconciliation phase and all random subsets (32 bits) of the

KEY_{Rec}'s exposed during the error-estimation sub-phase. This omission is needed for removing eavesdropper's knowledge of the KEY_{Rec}'s. Finally, both devices apply the MD5 hashing on the remaining secret bits to form the final 128 bits KEY_{Final}.

4.4 Evaluation

4.4.1 Implementation and Experimental Setup

In order to evaluate SekGens algorithm, we implemented it on Google Nexus 5 and Samsung Galaxy S2 smartphones. On these devices, the Android kernel code, that is responsible for creating the Wi-Fi Direct connection between any two devices, implements the WPS protocol. We downloaded, modified and built the LineageOS [124] Android kernel code for Google Nexus 5 and Samsung Galaxy S2 smartphones. The header and C files that implement the WPS protocol are located under the path: "/android/system/external/wpa_supplicant _8/src/wps".

In order to generate sensors data on the smartphones, we installed the "Accelerometer Monitor" [125], which is an Android app that uses the vibration sensor (accelerometer). This app generates three sensors data, the device acceleration data for each space axis: X, Y and Z (m/s^2). SekGens uses the value of $\sqrt{X^2 + Y^2 + Z^2}$ at each reading. We also installed the "Sound Meter" [126], which is Android app that uses the microphone to measure sound levels in dB (decibels). Each app stores its data in a log file that is saved on the SD card.

The quantization phase of SekGens is initiated when the user navigates to Wi-Fi Direct screen and searches for nearby available Wi-Fi Direct devices. We modified the WPS protocol such that the first message sent from the Enrollee to the Registrar contains the *NC*. Then, both devices start the quantization phase after reading the sensors data from the SD card.

For the reconciliation phase, in order to enable both devices to exchange parity bit messages, block number, block order of the current block being reconciled, and the random subsets of the reconciled KEY_Q, we defined new message types to the existing WPS protocol. Also, to enable the two devices to exchange reconciliation messages, we set the "EAP_MAX_AUTH_ROUNDS" variable to 200. After compiling the modified Android kernel code, we downloaded it on smartphones using the Android Debug Bridge [127] tool.

We evaluated our SekGens when the Enrollee and Registrar are smartphones of either the same or different hardware components using the following metrics, commonly used to evaluate the performance of secret key generation schemes:

1. **Pairwise Key Mismatch:** the fraction of different bits of the generated key at both ends, ideally 0%.

2. **Key Entropy:** this metric (shown in Equation 4.2) measures the randomness of the generated keys. A value close to 1.0 indicates high entropy with high random keys.

3. **SekGens Execution Time (msec):** the time delay resulted from running SekGens in smartphones.

4. **Secret Bit Rate:** the average number of secret key bits extracted from sensors per second. This depends on sensors sampling rate (i.e., 50 and 5 sample(s)/sec for "Accelerometer Monitor" and "Sound Meter", respectively) and SekGens execution time.

5. **Key Generation Diversity:** the key mismatch ratio among different successful runs of SekGens. Even if SekGens successfully generates keys with 0% mismatch ratio, it is still important to show that SekGens is able to generate very random keys for each run.

We are interested in the impact of the following on the performance of SekGens: a) the amount of sensor data collected for key generation; b) the type of sensor data used for key generation. For this, we performed the 12 test cases shown in Table 4.2. In each test case, the users either bump or handshake their smartphones together for 2 to 3 seconds and then they keep their smartphones either close (i.e., 0 m distance) or distant (i.e., 1 m or 2 m distances). These test cases were conducted in a cafeteria during and after a busy lunch hour (12:00 PM - 1:00 PM).

For each test case, we repeated the experiment five times and we averaged these five runs per each test case. The users start each experiment together by starting the "Accelerometer Monitor"

Table 4.2: Test cases for SekGens. *Reprinted with permission from [51].*

Test Case Number	Movement Shape	Distance Between Smartphones (m)	During Lunch Hour
TC_1	Bump	0	No
TC_2	Bump	0	Yes
TC_3	Bump	1	No
TC_4	Bump	1	Yes
TC_5	Bump	2	No
TC_6	Bump	2	Yes
TC_7	Handshake	0	No
TC_8	Handshake	0	Yes
TC_9	Handshake	1	No
TC_{10}	Handshake	1	Yes
TC_{11}	Handshake	2	No
TC_{12}	Handshake	2	Yes

app on their smartphones at the same time. However, the "Sound Meter" app was always running on each device. SekGens starts reading its output from the time when the users started the experiment.

4.4.2 Security Analysis

4.4.2.1 Brute-force attack defense

Claim 1. *The attacker, Eve, has to perform* 2^{128} *guesses in order to mount a brute-force attack against the* KEY_{Final} *generated from SekGens.*

Proof. To prove Claim 1., we have to show that if Eve tries to mount a brute-force attack against any key generated from any phase of SekGens, the difficulty of her guessing is at least equal to the difficulty of guessing a 128 bits key.

- **The quantization phase** generates the KEY_Q that is 320 bits. There are 160 iterations, at each iteration *j*, Key_j with length 2 bits is generated using the correlation coefficient, *coef*, which can take any value in the interval [-1.0, 1.0]. In our implementation, we multiply *coef* by 10 and round it to the nearest integer before we use it as a seed input to the *GenKey(coef)* function. So, Eve has two ways to brute force KEY_Q. Either by guessing different seed inputs, in this case she has to try 20^{160} guesses. Or by guessing each Key$_j$ separately, in this case she has to try $(2^2)^{160}$ guesses.

- **The reconciliation phase** might make Eve's task easier, when considering the exposed parity bit messages and the 32 bits that are needed for the error-estimation sub-phase. If Eve is able to discover the secret KEY$_{Rec}$, she can apply the MD5 hashing function to generate KEY$_{Final}$. Eve is able to sniff the parity bit messages ($\langle pb\ of\ current\_block$, $blk\_num,\ blk\_ord\rangle$), the subsets ($E\_subsets$ and $R\_subsets$) of KEY$_{Rec}$, as well as the random nonce *NC* before she starts guessing the KEY$_{Final}$.

  For each captured reconciliation message, Eve can omit half of the guessing space of the block being reconciled for both Enrollee and Registrar devices. After exchanging 4 messages ($log_2(8)$ + 1) for each 8 bit block, Eve ends up with 2^4 guesses for the block being reconciled, and, thus, she has to try 2^4 guesses for each 8 bits block. The ideal scenario for Eve is when the Enrollee and Registrar devices have different parities for all their 8 bit blocks. In that case, Eve can reduce its guessing space for all 8 bit blocks. Moreover, Eve knows that there are 32 bits that are needed for error-estimation.

  We have to find the length of KEY$_{Rec}$ such that after Eve narrows its guessing space based on the captured parity bits messages, and omits the 32 error-estimation bits, so that she still needs to make at least 2^{128} guesses. Finding the value of *Number_Of_Blocks* such that $\frac{(2^4)^{Number\_Of\_Blocks}}{2^{32}} \geq 2^{128}$ is satisfied guarantees that the difficulty of guessing the remaining secret bits after the reconciliation phase is at least equivalent to the difficulty of guessing a 128 bits key. The numerator in the above equation represents the number of guesses for each 8 bit block, when we have *Number_Of_Blocks* blocks. And the denominator represents the 32 error-estimation bits which are known by Eve. When solving the above equation, *Number_Of_Blocks* = 40, we found that the length of KEY$_{Rec}$ (*Number_Of_Blocks* $\times$ *block size*) has to be at least 320 bits (40 $\times$ 8). That is why the length of KEY$_Q$ is 320 bits.

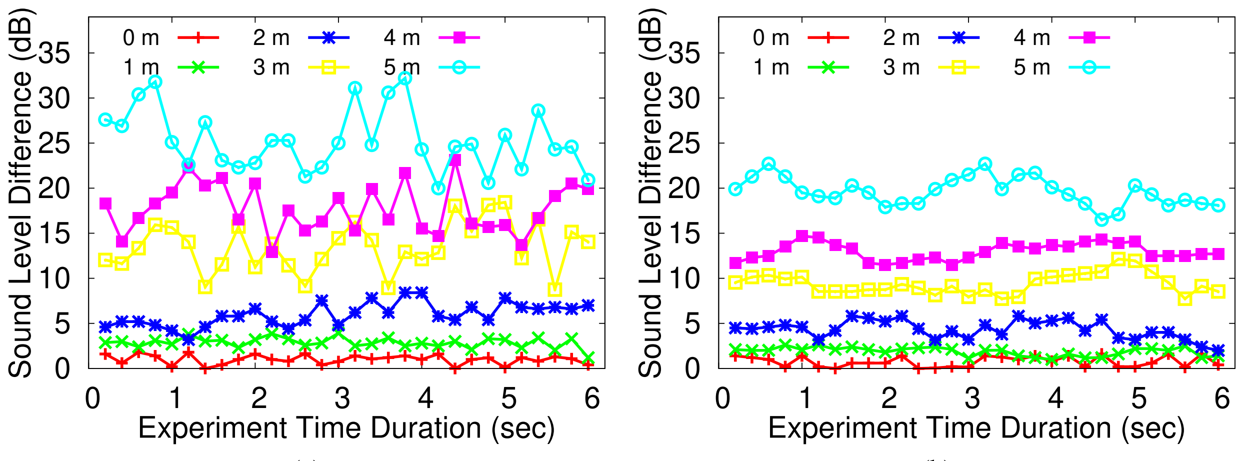
4.4.2.2 Imitation attack defense

In order to accomplish the imitation attack, Eve tries to generate the same sensors data of the Enrollee or Registrar devices. We used two sensors in Section 4.4.1. For the sound sensor, the sound level (chapter 2 of [128]) in the free space environments (i.e., the open air or the anechoic rooms) follows the inverse square law (i.e., the sound level will decrease 6.02 dB each time the distance from the source is doubled). In such environments, it might be easy for Eve to predict the sound level at the Enrollee location and fake its sound sensor data to pretend as the legal Enrollee.

In real-world, Eve prefers to attack the victims in crowded indoor areas (e.g., restaurants, cafes, airports lounges, or student community areas), where it is difficult to be noticed. In such crowded indoor areas, there are many sound sources (i.e., music, talking people and airport announcements) and the sound wave for each source is reflected and diffracted (chapter 10 of [129]) many times due to the walls, doors, tables and moving objects (i.e., customers and waiters in cafes and restaurants, passengers in airports lounges, students in student community areas). As a result, the inverse square law is not strictly applicable and the sound level is very random. Therefore, it is difficult for Eve to predict the sound level values unless she (physically) gets closer to the victims to have the same readings.

In order to investigate the distance that enables Eve to generate the same sound readings as the Enrollee, we performed six experiments shown in Figure 4.1 at different distances between Eve and the Enrollee to calculate the differences between their sound readings. For each distance, we repeated the experiment five times and we averaged these five runs per distance. These experiments were conducted in a cafeteria during and after a busy lunch hour.

As shown in Figures 4.1a and 4.1b, in order to make the sound readings for Eve $\approx$ the sound readings generated at the Enrollee location, Eve has to be close within 1 m distance to the Enrollee. Eve must depend on the reconciliation phase to eliminate the minor differences in the bits of its KEY_Q and the Enrollee KEY_Q. However, Eve can not get so close (i.e., $\sim$1 m) to the victims, otherwise, the victims will notice and discover her especially if she tries to imitate the same gesture (i.e., bump, handshake, etc.) made by the victims to generate data from the acceleration sensors.



(a)

(b)

Figure 4.1: Sound level difference between Eve and the Enrollee. (a) During Lunch Hour. (b) After Lunch Hour. *Reprinted with permission from [51].*

For more than 1 m distances, the difference in the sound readings increases and this makes Eve's task of discovering KEY$_Q$ more difficult.

Accordingly, we recommend the users to be within 1 m distance when they use SekGens. Indeed, 93 out of the total 98 successful experiments in Sections 4.4.3.2 and 4.4.3.3 are conducted when the Enrollee and Registrar devices are within 1 m distance. If Eve conceals herself in the environment and she is a skilful attacker such that she is able to imitate the exact gesture made by the victims, it is difficult for her to generate the same sound readings generated from the victims devices.

4.4.3 Performance Evaluation

4.4.3.1 Impact of Experiment Time Duration on KEYQ Mismatch

In order to investigate the best time duration for each experiment, we performed all experiments for all test cases for different time duration using two Google Nexus 5 smartphones. Figures 4.2a and 4.2b show the mismatch ratio of KEY$_Q$'s for all test cases. We found that the mismatch ratio of KEY$_Q$ is decreasing when we increase the time duration of each experiment, however, this mismatch ratio does not improve after 6 seconds. We aim at not increasing the experiment time

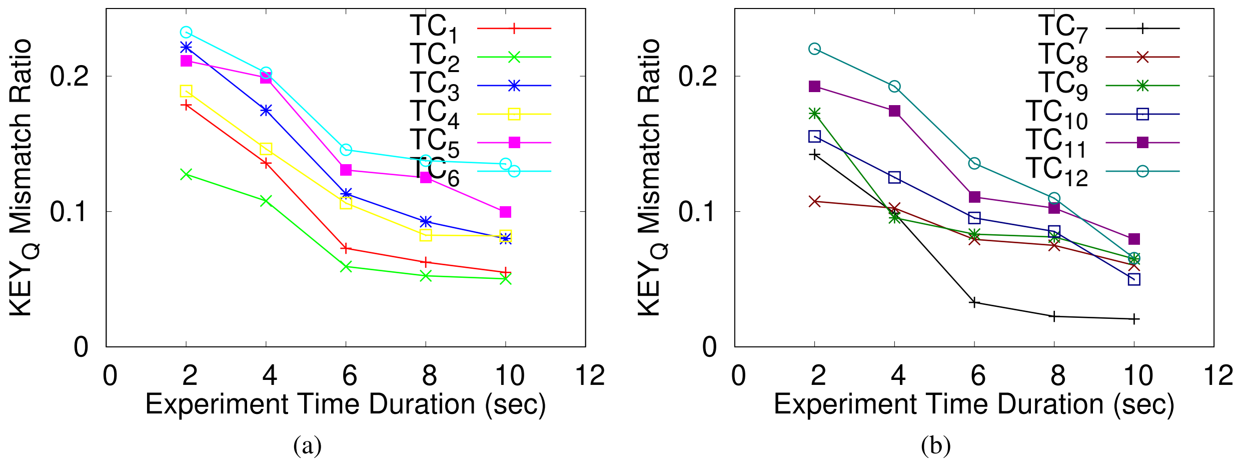


Figure 4.2: KEY$_Q$ Mismatch Ratio for different time durations. (a) TC$_1$ to TC$_6$ experiments. (b) TC$_7$ to TC$_{12}$ experiments. *Reprinted with permission from [51].*

duration because this lowers the secret bit rate of SekGens. We set the time duration for each experiment to 6 seconds and we guarantee that by reading the first 300 and 30 sensors data from the log files of "Accelerometer Monitor" and "Sound Meter", respectively.

4.4.3.2 SekGens on Same Smartphones

To perform the experiments on smartphones of the same hardware, we used two Google Nexus 5 smartphones to find out the impact of the type of data Enrollee and Registrar use to establish the KEY$_{Final}$ for their devices. We compare the performance of SekGens in terms of the metrics we mentioned above for all test cases. Figures 4.3 and 4.4 show the evaluation results of our metrics for all test cases, with error bars showing standard deviation.

Figures 4.3a and 4.4a show the *key mismatch ratio* for both KEY$_Q$'s and KEY$_{Rec}$'s. This represents the mismatch ratio after the quantization phase (KEY$_Q$) and the effect of the reconciliation phase on improving the key agreement between the Enrollee and Registrar devices (KEY$_{Rec}$). The mismatch ratio for all experiments after the reconciliation phase is less than 2%. Out of 60 experiments for all test cases, there were 53 experiments in which SekGens could generate the same KEY$_{Rec}$ on both devices (i.e., the success rate is 88%).

The *key mismatch ratio* increases for 1 m and 2 m distances between the two smartphones. This

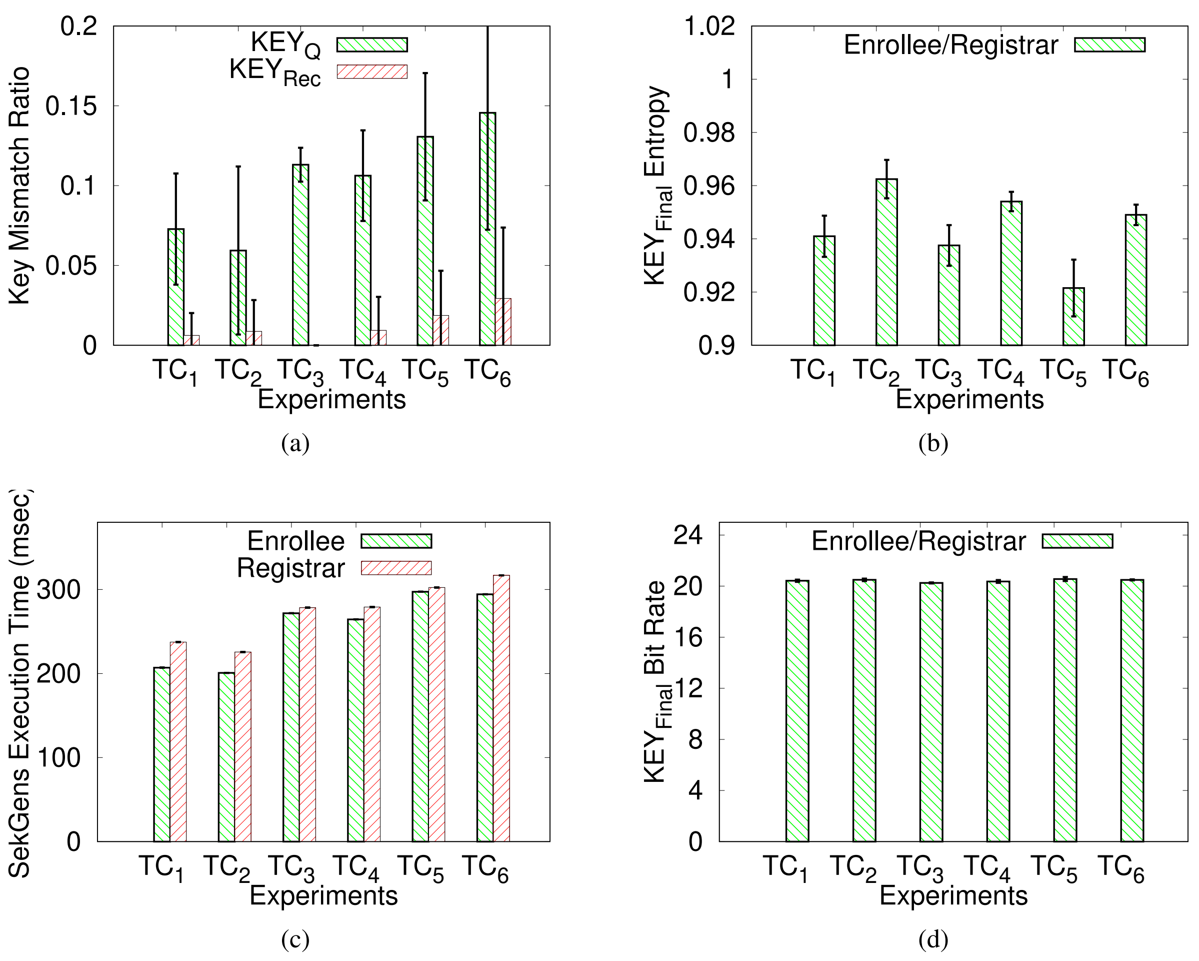
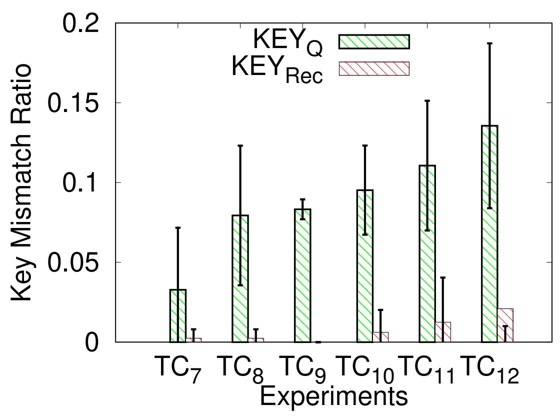
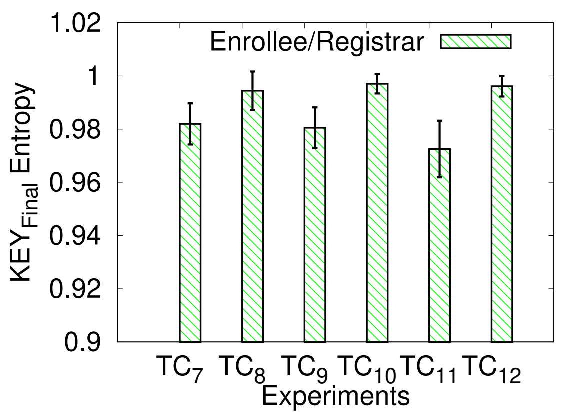


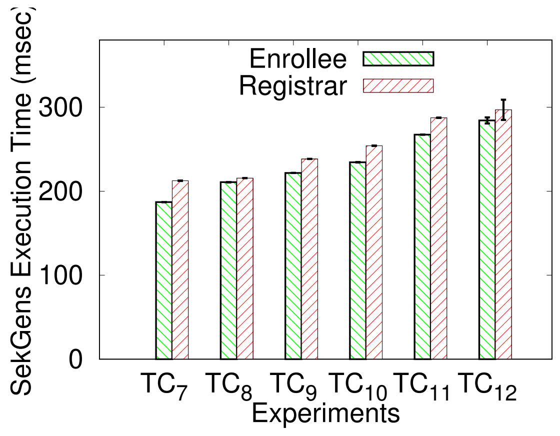
Figure 4.3: TC_1 to TC_6 on same hardware. (a) Mismatch Ratio. (b) KEY_{Final} Entropy. (c) Execution Time. (d) KEY_{Final} Bit Rate. *Reprinted with permission from [51].*



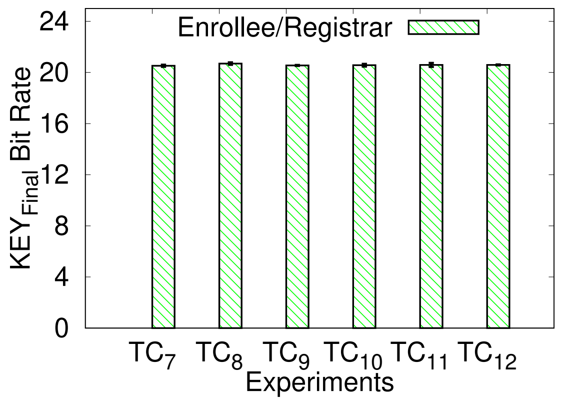
(a)



(b)



(c)



(d)

Figure 4.4: TC$_7$ to TC$_{12}$ on same hardware. (a) Mismatch Ratio. (b) KEY$_{Final}$ Entropy. (c) Execution Time. (d) KEY$_{Final}$ Bit Rate. *Reprinted with permission from [51].*

increase in the mismatch ratio is due to the fact that the readings of the "Sound Meter" app depend mainly on the location of the smartphones (i.e., the readings are different for distinct locations). However, this mismatch ratio is reduced after the reconciliation phase.

Figures 4.3b and 4.4b show the *entropy* of KEY$_{Final}$ resulted from the successful experiments of all test cases. The generated KEY$_{Final}$'s entropy is higher than 92% for all experiments, which indicates a high level of randomness of KEY$_{Final}$. The KEY$_{Final}$ entropy values resulted from the handshake experiments (Figure 4.4b) are higher compared to the bump experiments (Figure 4.3b). This is due to the higher entropy of "Accelerometer Monitor" readings when the users handshake their smartphones rather than bump them. Also, the KEY$_{Final}$ entropy values are higher for the experiments that were conducted during the lunch hour. This is due to the higher entropy of "Sound Meter" readings for these experiments.

Figures 4.3c and 4.4c show the *execution time* of SekGens needed to generate the KEY$_{Final}$ for the successful experiments of all test cases. The execution time is higher for the test cases with higher KEY$_Q$ mismatch ratio due to the additional reconciliation messages needed to reconcile the KEY$_Q$. In all experiments, the execution time for the Registrar is higher than the execution time for the Enrollee. This is due to an implementation choice of SekGens in which the Enrollee is the terminator of the last phase of SekGens that is the privacy-amplification-and-hashing phase. The Enrollee finishes SekGens when it applies the MD5 hashing to form its KEY$_{Final}$ and then it sends message M$_1$ of WPS protocol to the Registrar as an indication of pursuing the communication. However, the Registrar still needs to apply the MD5 hashing to form its KEY$_{Final}$ before finishing SekGens, then it replies with message M$_2$ of WPS protocol.

The KEY$_{Final}$ *bit rate* for the successful experiments of all test cases are shown in Figures 4.3d and 4.4d. Because it is longer than the Enrollee, we use the execution time of the Registrar in calculating the KEY$_{Final}$ bit rate. Even though this execution time is less than 1 sec, the time needed to generate sensors data (i.e., 6 seconds) dominates the bit rate and lowers it to be around 20 bits/sec. This KEY$_{Final}$ bit rate might be increased with sensors with higher sampling rate.

4.4.3.3 SekGens on Different Smartphones

To investigate the robustness of SekGens on smartphones with different hardware components, we repeated the 12 test cases but on different smartphones (i.e., Google Nexus 5 and Samsung Galaxy S2). Figures 4.5 and 4.6 show the evaluation results of these experiments, with error bars showing standard deviation.

Figures 4.5a and 4.6a show the *key mismatch ratio* for both KEY$_Q$'s and KEY$_{Rec}$'s for all test cases. The mismatch ratio for all experiments after the reconciliation phase is less than 3%, which proves that SekGens is robust when running on different smartphones. Out of 60 experiments for all test cases, there were 45 experiments in which SekGens could successfully generate the same

Table 4.3: NIST statistical test suite results. The *p* value from each test for the successful experiments of all test cases is shown below. To pass a test, the *p* value for that test has to be > 0.01. *Reprinted with permission from [51].*

Statistical Test	**Section 4.4.3.2 Experiments**	**Section 4.4.3.3 Experiments**
Frequency	0.03	0.04
Block Frequency	0.28	0.04
Cumulative Sums (Fwd)	0.28	0.04
Cumulative Sums (Rev)	0.04	0.02
Runs	0.05	0.04
Longest Run of Ones	0.06	0.20
Approximate Entropy	0.13	0.70
Serial	0.04, 0.25	0.02, 0.04

KEY$_{Rec}$ on both smartphones.

The *entropy* of KEY$_{Final}$ for the successful experiments of all test cases are shown in Figures 4.5b and 4.6b. SekGens creates KEY$_{Final}$ with high entropy (i.e., $> 92\%$). The *execution time* of SekGens for the successful experiments is between 200 and 350 msec as shown in Figures 4.5c and 4.6c. And the KEY$_{Final}$ *bit rate* for the same experiments is around 20 bits/sec as shown in Figures 4.5d and 4.6d.

By comparing the performance of SekGens to the most related previous works [65] and [66], SekGens is better in terms of success rate, key bit rate, execution time, security against offline attacks and robustness on different hardware sensors.

4.4.3.4 Key Diversity Validation for SekGens

To ensure the randomness of the generated KEY$_{Final}$ bit streams by SekGens, we run the randomness tests available in the NIST test suite [130]. Even though there are 15 different statistical tests in the NIST test suite, we run only 8 tests due to the fact that the KEY$_{Final}$ bit streams that we obtain from SekGens meet the input size recommendations of the 8 NIST tests only. We find that the SekGens generated KEY$_{Final}$ bit streams pass all the 8 tests as shown in Table 4.3. The remaining 7 tests require a very large input bit stream ($\approx 10^6$ bits).

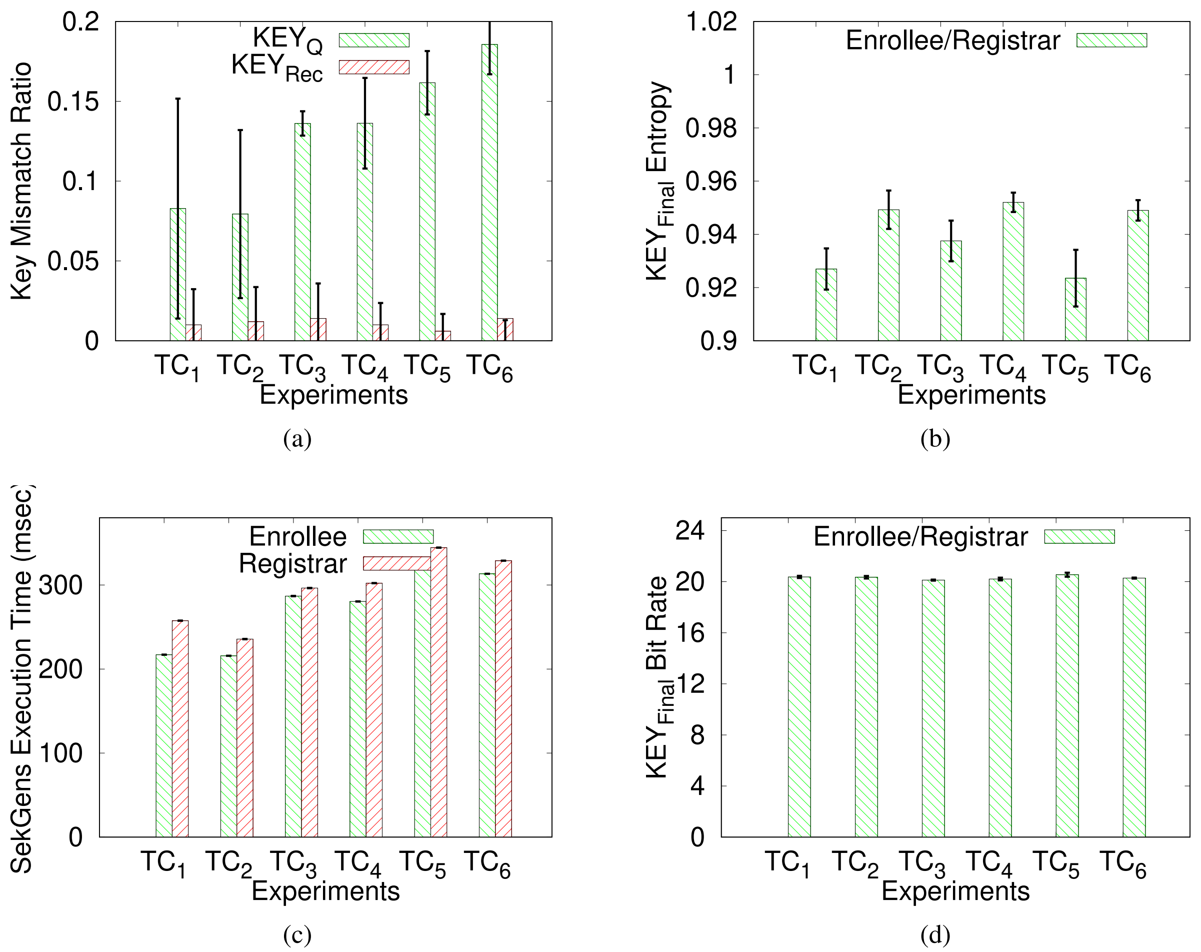
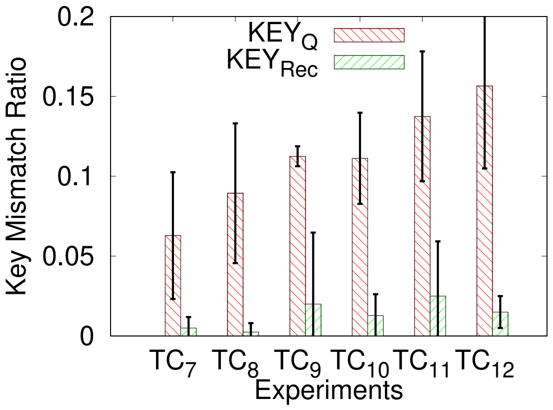
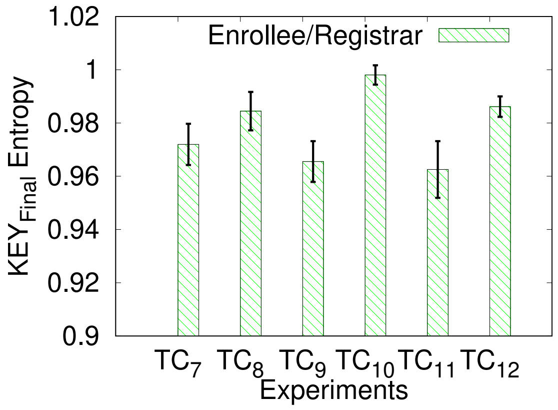


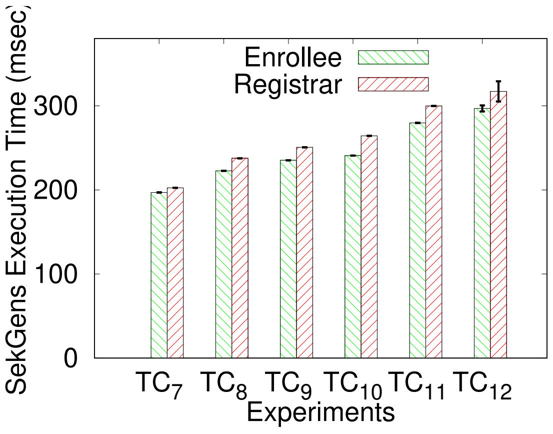
Figure 4.5: TC$_1$ to TC$_6$ on different hardware. (a) Mismatch Ratio. (b) KEY$_{Final}$ Entropy. (c) Execution Time. (d) KEY$_{Final}$ Bit Rate. *Reprinted with permission from [51].*



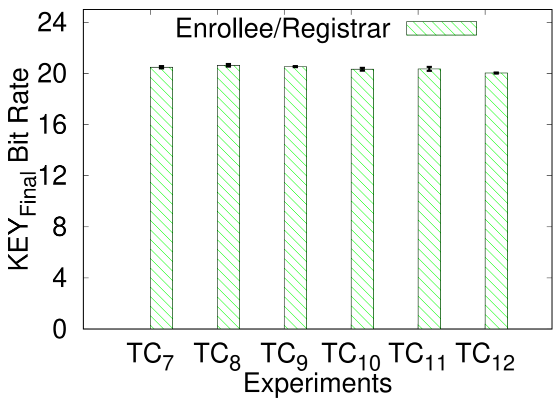
(a)



(b)



(c)



(d)

Figure 4.6: TC$_7$ to TC$_{12}$ on different hardware. (a) Mismatch Ratio. (b) KEY$_{Final}$ Entropy. (c) Execution Time. (d) KEY$_{Final}$ Bit Rate. *Reprinted with permission from [51].*

5. SECURING THE PUSH-BUTTON METHOD IN WDONs*

In this section, we first show that Group Owner (GO) devices in Wi-Fi Direct are vulnerable to the *EvilDirect* attack. In the EvilDirect attack, a rogue GO is set up by Eve to look like the legitimate GO (with the same MAC address, SSID, and operating channel). Eve intercepts the clients' invitation requests and accepts them before the legitimate GO. Accordingly, Eve hijacks the wireless communications between the clients and the legitimate GO. The best areas for EvilDirect attacker are public, indoor areas. These areas can be either *dynamic environments* (mobile devices, and/or there are mobile intermediate objects in the environment, e.g., airport lounges, cafes, restaurants, or student community areas), or *static environments* (static devices, and there are few mobile intermediate objects in the environment, e.g., libraries).

Next, in order to address the EvilDirect for the dynamic environments in WDONs, we propose to employ the inherent randomness in the wireless channel between the clients and the GO. The properties [50] of the radio channel are unique to the locations of any two devices because: a) The multi-path properties of the channel at any point in time are identical in both directions of the link; b) The temporal variations in the channel (i.e., the multi-path channel changes due to motion of people in the environment). As a result, the adversary, Eve, will measure a different and uncorrelated radio channel due to the following reasons: a) He cannot be very close to the legitimate GO or the client (i.e., closer than one half of the wavelength, λ); b) He cannot restrict other movements in the area (passengers, customers, etc.). Most of current wireless cards of smartphones are able to measure the received signal strength (RSS). Thus, we exploit this capability in our detection. Our proposed protocol, EvilDirectHunter, is an interactive protocol that executes between each client in the P2P group and the GO. Each client records an RSS profile for the legitimate GO, which in turn records a profile for each client in the P2P group. EvilDirectHunter checks whether the RSS profiles of both the client and potential GO devices are similar to each other by exchanging chal-

*Reprinted with permission from "EvilDirect: A New Wi-Fi Direct Hijacking Attack and Countermeasures" by Ala Altaweel, Radu Stoleru, and Guofei Gu, In Proceedings of 2017 26th International Conference on Computer Communication and Networks (ICCCN), pages 1-11, Vancouver, BC, Canada, 2017, Copyright 2017 by IEEE.

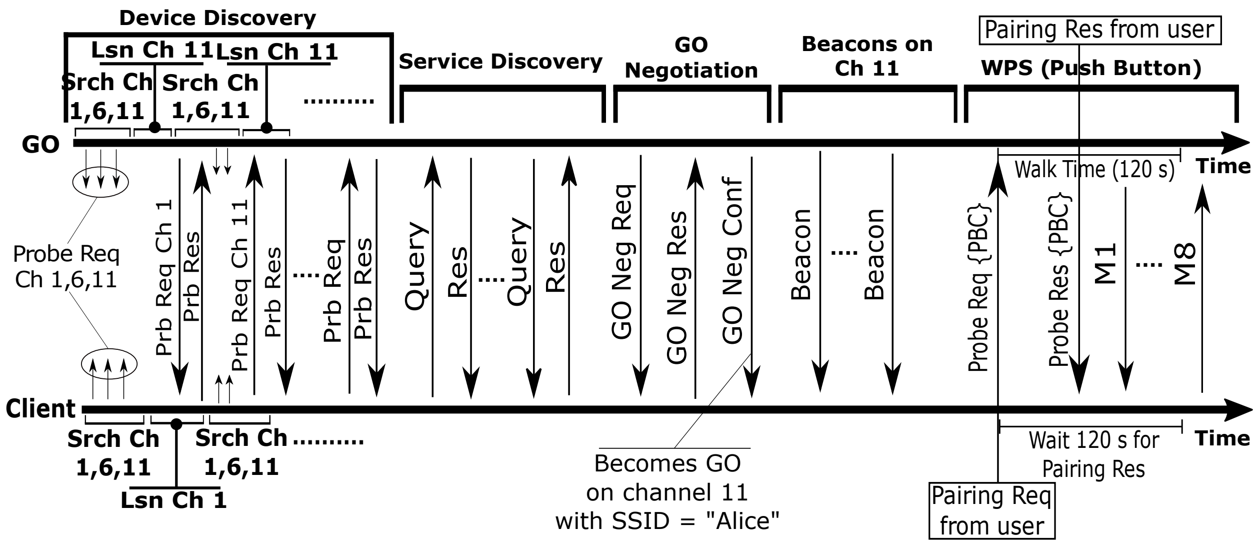
lenge and response packets. For the static environments in WDONs, due to the lack of randomness in the wireless channel, there are low variations in the RSS profiles. These low RSS variations are not sufficient to detect the EvilDirect attacks. Accordingly, our EvilDirectHunter executes an additional detection phase, which is designed for the static environments. This phase is based on Multi-Dimensional Scaling (MDS) algorithm [60] and involves a cooperation among all clients in the P2P group. Moreover, this phase requires that each client calculates the average of the RSS values of the last five packets that were overheard by it when any other client exchanged packets with the legitimate GO.

5.1 Background and Motivation

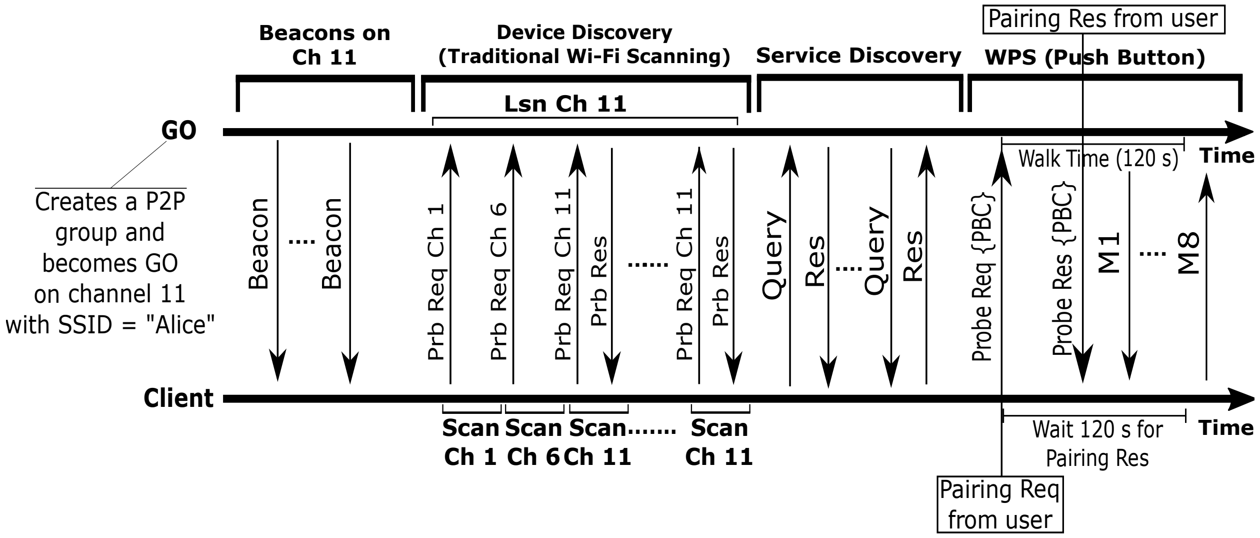
The role of a device in a Wi-Fi network is either an AP or a client. Each role has different sets of functionalities. However, these roles are dynamic in Wi-Fi Direct [1]. Any Wi-Fi Direct device implements the role of GO (soft AP) and the role of client. These logical roles could be executed simultaneously by a single device either by time-sharing the channel via virtualization techniques, or using different frequencies.

Wi-Fi Direct devices form P2P groups, which are equivalent in terms of functionalities to the Wi-Fi infrastructure networks. In any P2P group, P2P GO implements the AP-like functionality, and the other devices are P2P clients. There are three group formation mechanisms [1] to establish a P2P group: Standard, Autonomous (which are both our focus in this dissertation), and Persistent. Each mechanism has three main phases: *Device Discovery*, *Service Discovery*, and *Wireless Protected Setup (WPS) Provisioning*, as shown in Figure 5.1.

The P2P devices in the Standard formation mechanism start by executing a “discovery algorithm” as shown in Figure 5.1a. First, each P2P device selects one of the channels (i.e., channels 1, 6, or 11 in the 2.4 GHz band) as its listening channel. Second, a P2P device alternates between the search and listen states. In the search state, the device performs active scanning by sending probe requests in each channel. In the listen state, the device listens for probe requests in its listening channel to respond with probe responses. After the two P2P devices find each other, they start the Service Discovery phase.



(a)



(b)

Figure 5.1: Frame exchange sequences for (a) Standard mechanism. (b) Autonomous mechanism. *Reprinted with permission from [52].*

The Service Discovery phase in the Standard formation mechanism (shown in Figure 5.1a) is an additional feature for Wi-Fi Direct devices, which was not present in the traditional Wi-Fi networks (i.e., the only service in which clients are interested in, is Internet connectivity). Prior to the establishment of a P2P group, P2P devices exchange queries to discover their available services and, based on this, decide whether to continue the group formation or not. Service discovery queries are generated by a higher layer protocol (e.g., UPnP or Bonjour [131]), and employing the link layer Generic Advertisement Protocol (GAS) specified by 802.11u [132]. GAS is a layer two query/response protocol implemented through the use of public action frames, that allows two non-associated P2P devices to exchange queries belonging to a higher layer protocol.

The Standard formation mechanism has an additional phase, the GO Negotiation (shown in Figure 5.1a). This phase is implemented using three-way handshake messages (i.e., Request, Response, and Confirmation). These messages enable the two devices to decide which one will be the GO and on which channel the P2P group will operate. Each device sends a numerical parameter (GO Intent Value (IV)) and a tiebreaker bit within the handshake messages. The device with the highest IV becomes the GO. The tiebreaker bit prevents conflicts when two devices declare the same IV. In Figure 5.1a, the upper device wins the GO Negotiation, and selects channel 11 for its P2P group with SSID="Alice" (i.e., the SSID is set by the GO's user). After that, this GO starts to beacon on channel 11 and waits for other clients to join its group (i.e., the new joining clients will not execute the GO Negotiation phase).

The Persistent formation mechanism [1] enables the P2P device to declare a group as persistent by using a flag (P2P capabilities attribute) in the sent beacons, probe responses, and GO negotiation frames. The P2P devices that form the persistent group store its network credentials, the P2P GO, and P2P clients for subsequent re-forming of the group.

The Wireless Protected Setup (WPS) Provisioning [12] phase is implemented by the Standard formation mechanism (as shown in Figure 5.1a). WPS aims to achieve a secure connection between P2P clients and P2P GO. WPS has two modes of operation: in-band mode and out-of-band mode. There are two methods for the in-band mode: Push-Button method and PIN method. Both methods are used for authentication and to establish Diffie-Hellman keys between the client and the GO.

In the Push-Button method the client invites the GO to start the WPS protocol and waits for its acceptance. As shown in Figure 5.1a, the P2P client's user presses a logical button to invite his friend's GO device (i.e., with SSID="Alice") to start the WPS protocol. The client device includes its mode (i.e., Push-Button Configuration (PBC)) in the invitation to notify the GO to enable it. The GO's user either accepts or declines the invitation by pressing a logical button within 120 seconds (known as Walk Time). In case of acceptance, the two devices execute the eight registration protocol messages (M_1 to M_8 in Figure 5.1a) of the WPS protocol and connect to each other (page 33 of [12]).

The Autonomous formation mechanism [1], shown in Figure 5.1b, enables the P2P device's user to immediately set his device as a GO and create a P2P group (i.e., with specific SSID and channel). Then, this GO starts to beacon on the specified channel. Other devices can discover the P2P group using traditional scanning mechanisms and directly proceed with the Service Discovery and WPS Provisioning phases (i.e., no GO Negotiation phase is required).

In the following paragraph, we present our successful EvilDirect attack against Wi-Fi Direct GOs for both Standard and Autonomous group formation mechanisms. This attack is based on the fact that two GOs with the same MAC address and SSID, that operate on the same channel are indistinguishable by the clients and treated as a single GO.

We successfully launched an EvilDirect attack against GO devices due to the fact that two GOs with the same MAC address and SSID, that operate on the same channel are indistinguishable by the clients and treated as a single GO. In order to accomplish this attack, the attacker needs a smartphone/laptop running Ubuntu. The first step for the attacker is to discover (using Apps like WiFiScanner [133]) the MAC address, SSID, and operating channel of the legitimate GO. The second step for the attacker is to set his EvilDirect GO to look like the legitimate GO. On a low end notebook, we configured the wireless interface in monitor mode. Then we used the Airbase-ng attack tool to create the EvilDirect GO (with the same MAC address, SSID, and channel as the legitimate GO (shown in Figure 5.2)). When the client's user tries to connect to the legitimate

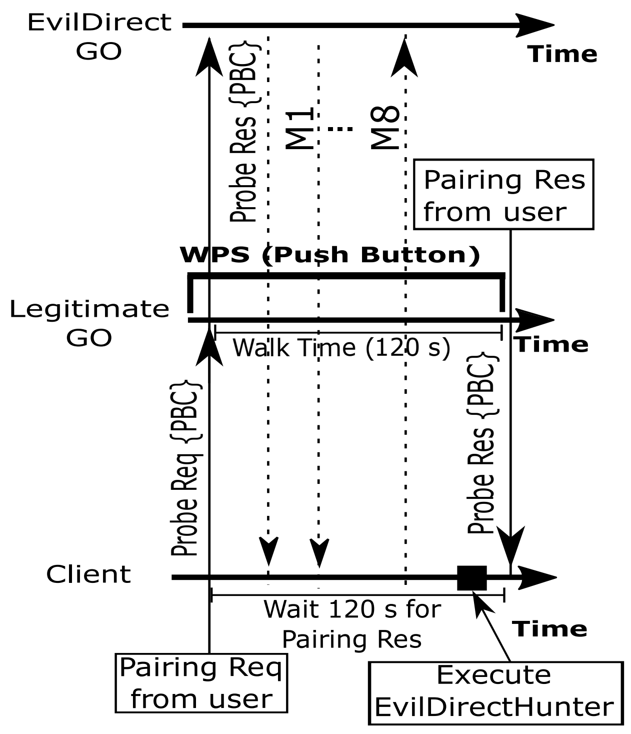


Figure 5.2: EvilDirect attack. *Reprinted with permission from [52].*

GO, his device invites the legitimate GO to start the WPS protocol and waits for its acceptance for 120 seconds, as shown in Figure 5.2. The EvilDirect GO receives this invitation too, as shown in Figure 5.2. Before the legitimate GO's user accepts this invitation by pressing a logical button, the EvilDirect GO can reply with an acceptance to the client's invitation (with the same MAC address and SSID as the legitimate GO). Then, the EvilDirect GO executes the eight registration protocol messages of the WPS protocol with the client and hijacks its wireless communication.

5.2 EvilDirectHunter Design

5.2.1 Main Idea

As shown in Figure 5.3, EvilDirectHunter is based on the idea that while the client and the legitimate GO execute the Device Discovery and Service Discovery phases in both Standard and Autonomous group formation mechanisms (i.e., before the attack), the client records an RSS profile with say $2k$ samples, $k \in \mathbb{N}$ (i.e., $[rss_1, rss_2, ...rss_k, rss_{k+1}, ...rss_{2k}]$) for the legitimate GO, which in turn records a profile for each client in the P2P group with the same number of samples.

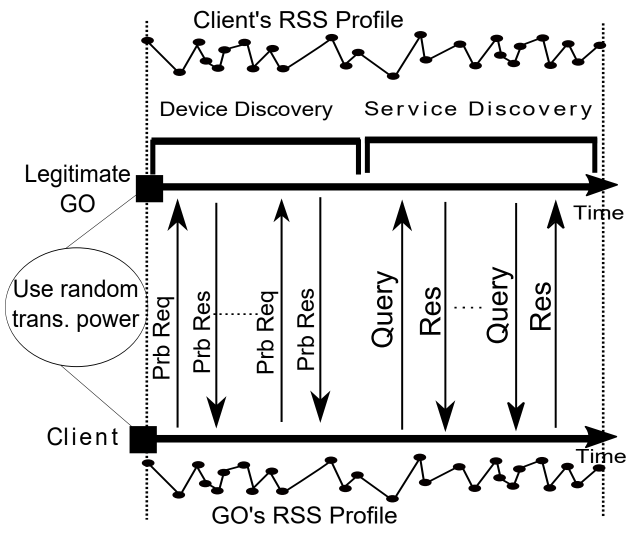


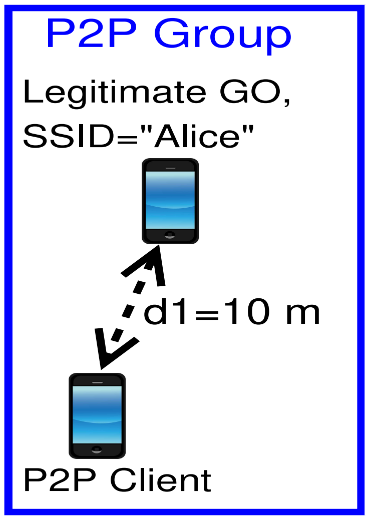
Figure 5.3: RSS Profiling at the client and legitimate GO. *Reprinted with permission from [52].*

Moreover, each client calculates the average of the RSS values of the last five packets that were overheard by it when any other client exchanged packets with the legitimate GO.

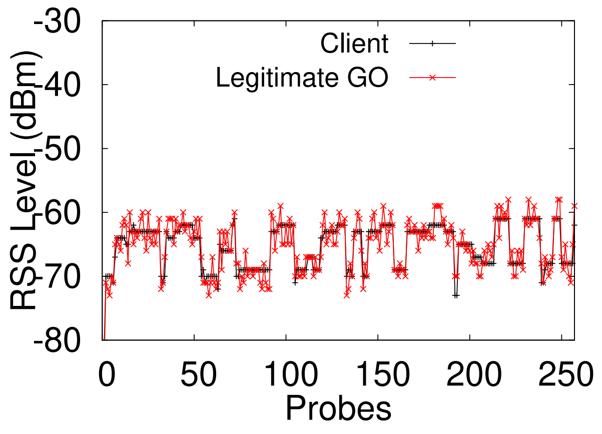
An important requirement of EvilDirectHunter as shown in Figure 5.3, is that each time before executing the Device Discovery and Service Discovery phases, the P2P devices set their transmission power randomly (i.e., tx_{pwr}= $Rand(tx_{min}, tx_{max})$). tx_{max} varies per country. We will discuss the purpose of this in Section 5.3.2.2.

In order to validate whether the RSS profile is suitable to differentiate legitimate GO and EvilDirect GO for dynamic and static environments, we performed five experiments using two Google Nexus 5 smartphones and recorded 256 RSS readings between one client and a legitimate GO, as shown in Figure 5.4a. $d1$ (i.e., the distance between the client and the legitimate GO) in these experiments is 10 m. The goals of these experiments are to examine the *randomness* (i.e., the variation window), and the *degree of reciprocity* (i.e., the Mean Squared Error (MSE) = $\frac{1}{256}\sum_{i=1}^{256}[RSSClient_i - RSSGO_i]^2$) of the two devices' RSS profiles. With MSE closer to zero indicating a higher degree of reciprocity.

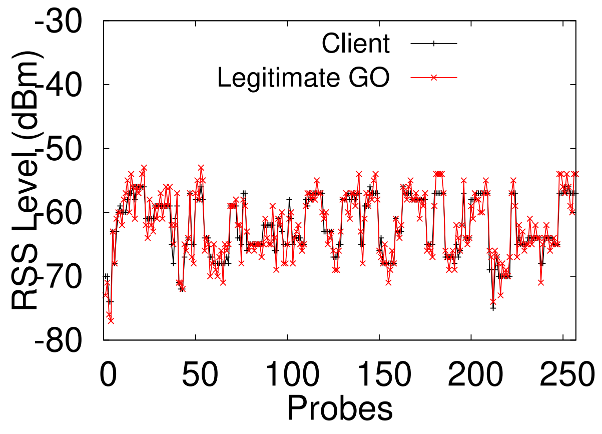
The first four experiments (with results shown in Figures 5.4b, 5.4c, 5.4d, and 5.4e) were conducted in a dynamic indoor environment (a cafeteria during and after a busy lunch hour (12:00 PM



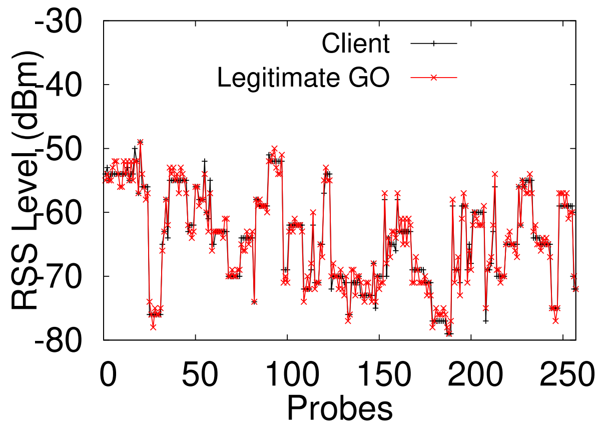
(a)



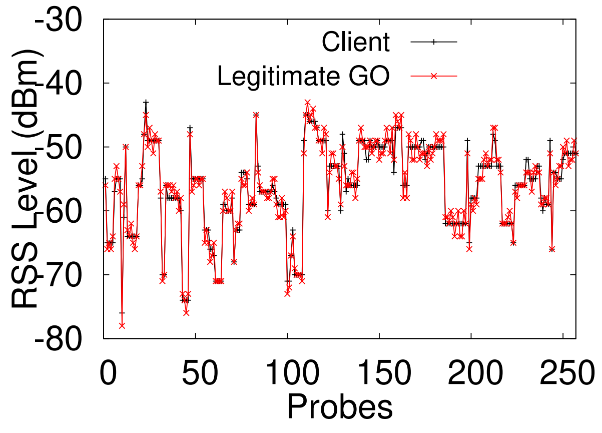
(b)



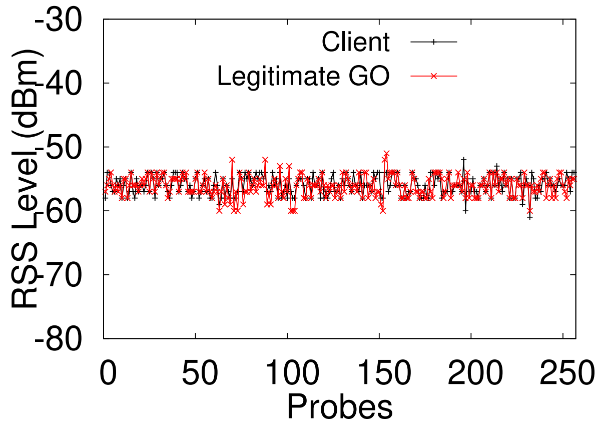
(c)



(d)



(e)



(f)

Figure 5.4: (a) RSS recording experiment. RSS profiles of one Client and GO (b) After lunch hour without shaking. (c) After lunch hour with shaking. (d) During lunch hour without shaking. (e) During lunch hour with shaking. (f) In a library. *Reprinted with permission from [52].*

- 1:00 PM)). During the experiments for Figures 5.4c and 5.4e, the two users randomly shook their smartphones 5 times for a short time duration ($\sim$3 seconds). The fifth experiment (Figure 5.4f) was conducted in a static indoor environment (a library building on a weekday evening).

As shown in Figures 5.4b, 5.4c, 5.4d, and 5.4e, the variation window of the RSS profile (i.e., the randomness of the channel) increases when we have more moving intermediate objects in the environment between the client and the legitimate GO. The shaking of the smartphones also increases the randomness of the recorded RSS profiles as shown in Figures 5.4c and 5.4e. Moreover, the mobility of either the intermediate objects or the endpoints helps in achieving a higher degree of reciprocity of the RSS profiles (the MSE of the two devices' RSS profiles for Figures 5.4b, 5.4c, 5.4d, and 5.4e are: 3.43, 3.12, 1.81, and 1.54 respectively). This mobility improves the reciprocity degree due to the fact that it dominates the RSS profile variation more than the random thermal noise and different interference sources, which affect the client and the legitimate GO differentially.

However, as shown in Figure 5.4f, there are no noticeable variations in the channel (i.e., low Shannon entropy of the RSS profile), and the MSE of the two devices' RSS profiles = 13.36 indicating a channel with low reciprocity. This is due to the fact that the variations in a static channel are primarily generated by thermal effects and hardware imperfections, which are non-reciprocal.

5.2.2 EvilDirectHunter Protocol

EvilDirectHunter is an interactive protocol that runs between each client and the GO. The client starts the execution of EvilDirectHunter after the execution of the eight messages of WPS protocol with the potential GO that accepts its invitation, as shown in Figure 5.2, (i.e., all EvilDirectHunter's messages are encrypted and authenticated using the WPS secret keys). Both devices incrementally prove the mutual knowledge of the RSS profile by exchanging challenge and response packets. Table 5.1 presents the notations we use in describing EvilDirectHunter. Essentially, the GO responds to the client's challenge and creates a new challenge to the client in each packet, and vice versa. Algorithms 5 and 6 present the pseudo code of EvilDirectHunter at the client and GO devices, respectively.

Algorithm 5 EvilDirectHunter at each Client

```
1: Read the first $2^\eta$ of $GOprfl$
2: $pass\# \leftarrow 1$
3: $\langle pass\#, Rss_{id_I}, Rss_{id_{II}} \rangle \leftarrow Pick2RSS(pass\#, GOprfl, 2^\eta)$
4: $challenge_C = Rss_{id_I} \oplus Rss_{id_{II}}$
5: send $\langle pass\#, id_I, id_{II}, null \rangle$ to $GO$
6: while $true$ do
7:    wait $\langle pass\#, id_I, id_{II}, response_{GO} \rangle$ from $GO$
8:    if $|challenge_C - response_{GO}| <= \tau$ then
9:       if $id_I \neq id_{II} \neq -1$ then
10:          $Rss_I$ =GetSubset ($pass\#, GOprfl, 2^\eta, id_I$)
11:          $Rss_{II}$ =GetSubset ($pass\#, GOprfl, 2^\eta, id_{II}$)
12:          $response_C = Rss_I \oplus Rss_{II}$
13:          $\langle pass\#, Rss_{id_I}, Rss_{id_{II}} \rangle \leftarrow Pick2RSS(pass\#,$
             $GOprfl, 2^\eta)$
14:          if $pass\# \neq null$ then
15:             $challenge_C = Rss_{id_I} \oplus Rss_{id_{II}}$
16:             send $\langle pass\#, id_I, id_{II}, response_C \rangle$ to $GO$
17:       else
18:          $GOprflEntropy \leftarrow H(GOprfl)$
19:          if $GOprflEntropy > \varepsilon$ then
20:             Legitimate GO, Stop
21:          else
22:             Broadcast $V$
23:             $D \leftarrow$ get $V$'s of other clients
24:             $Y \leftarrow$ MDS ($|D|$, 2)
25:             $\left\langle d(R_{GO}, R_{\widehat{GO}}), Avg_{i:1\rightarrow n}[d(R_{C_i}, \widehat{R_{C_i}})] \right\rangle \leftarrow Y$
26:             if $d(R_{GO}, R_{\widehat{GO}}) > Avg_{i:1\rightarrow n}[d(R_{C_i}, R_{\widehat{C_i}})]$ then
27:                EvilDirect GO, Stop
28:             else
29:                Legitimate GO, Stop
30:    else
31:       EvilDirect GO, Stop
```

Table 5.1: Notations and definitions for EvilDirectHunter. *Reprinted with permission from [52].*

C: Client device.
n: Total number of devices in the P2P group (clients and GO).
$\boldsymbol{\eta}$: max $\boldsymbol{j} \in \mathbb{N}$, s.t. $\boldsymbol{2^j <= size(RssPrfl)}$.
$\boldsymbol{H(GOprfl)}$: $\boldsymbol{-\sum_{i=1}^{l} p(GOprfl_i) \times log_2 p(GOprfl_i)}$, s.t. $\boldsymbol{l = size(GOprfl)}$.
$\boldsymbol{\varepsilon}$: Threshold of Shannon entropy for the dynamic environments.
$\boldsymbol{\beta}$: Smallest size of RSS subset (i.e., 16 RSS readings).
$\oplus$: (Euclidean distance)$^{\mathbf{2}}$ between RSS subsets.
$\boldsymbol{\tau}$: Threshold of difference between C and GO $\oplus$ calculations.
$\boldsymbol{D}$: $\boldsymbol{(n \times 2) \times (n \times 2)}$ RSS distance matrix.
$\boldsymbol{Y}$: $\boldsymbol{(n \times 2) \times 2}$ configuration matrix. (i.e., output of MDS).

Algorithm 6 EvilDirectHunter at the GO

```
1:  Read the first $2^\eta$ of $Cprfl$
2:  while $true$ do
3:      wait $\langle pass\#, id_I, id_{II}, response_C \rangle$ from $C$
4:      if $pass\# == 1$ or $|challenge_{GO} - response_C| <= \tau$ then
5:          $Rss_I$ =GetSubset ($pass\#, Cprfl, 2^\eta, id_I$)
6:          $Rss_{II}$ =GetSubset ($pass\#, Cprfl, 2^\eta, id_{II}$)
7:          $response_{GO} = Rss_I \oplus Rss_{II}$
8:          $\langle pass\#, Rss_{id_I}, Rss_{id_{II}} \rangle \leftarrow Pick2RSS(pass\#, Cprfl, 2^\eta)$
9:          if $pass\# \neq null$ then
10:             $challenge_{GO} = Rss_{id_I} \oplus Rss_{id_{II}}$
11:         else
12:             $id_I \leftarrow id_{II} \leftarrow -1$
13:         send $\langle pass\#, id_I, id_{II}, response_{GO} \rangle$ to $C$
14:     else
15:         Illegitimate Client, Stop
```

1) Phase I of EvilDirectHunter for dynamic and static environments: the client reads the first 2^η of the GO's RSS profile ($GOprfl$) (e.g., for 290 RSS readings, it reads the first 256 readings). For the first pass, $pass\# = 1$, the client creates a challenge packet by dividing the $GOprfl$ into two subsets using the Pick2RSS function presented in Algorithm 7. First, Pick2RSS divides the $GOprfl$ into $total$ subsets based on the value of the $pass\#$ (i.e., 2 subsets each of size 128 for the 256 readings). Second, it picks two random subsets, which were not selected before as challenges. These two random subsets are tagged so that they will not be picked again for the next challenges (line 8 of Algorithm 7). Pick2RSS returns $\langle 1, Rss_1, Rss_2 \rangle$ for the first pass (Rss_1 and Rss_2 contain the first and last 128 RSS readings of the $GOprfl$, respectively).

The client calculates the $\oplus$ metric (defined in Table 5.1) between Rss_1 and Rss_2 subsets, and stores the result in $challenge_C$ (line 4 of Algorithm 5). Then, it sends its challenge packet, $\langle pass\#, id_I, id_{II}, null \rangle$, which contains the indices of its challenge to the GO, and waits for its response and challenge (lines 5 & 7 of Algorithm 5). For $pass\# = 1$, there is no challenge for the client ($response_C$ is $null$).

The GO waits for the client packet as shown in line 3 of Algorithm 6 after it reads the first 2^η of the client's RSS profile ($Cprfl$). For the first pass, the GO responds to the client's challenge by getting the two RSS subsets of $Cprfl$, which correspond to id_I and id_{II} (indices of the client's challenge to the GO). The GO uses the GetSubset function (Algorithm 8) to get these two RSS subsets (lines 5 & 6 of Algorithm 6). Then, the GO calculates $\oplus$ between these two subsets (line 7 of Algorithm 6) to prepare its response. Moreover, the GO prepares its challenge using Pick2RSS function. Since the two 128 RSS subsets of $Cprfl$ have been used as challenges by the client, Pick2RSS increases $pass\#$ by one. Accordingly, $Cprfl$ is divided into four subsets (each of size 64 for the 256 readings). Pick2RSS randomly picks two subsets out of them to challenge the client. The GO sends $\langle pass\#, id_I, id_{II}, response_{GO} \rangle$ (line 13 of Algorithm 6), which contains the $pass\#$ and indices for its new challenge, and the response to the client's previous challenge.

Both devices follow the protocol by responding to other device's challenge and preparing its new challenge as long as $|challenge_{C/GO} - response_{GO/C}| <= \tau$ (lines 8 & 4 of Algorithms 5 and 6, respectively). Otherwise, either the client detects the GO as an EvilDirect GO (line 31 of Algorithm 5), or the GO stops the protocol (line 15 of Algorithm 6) to prevent the client spoofing attack (which will be discussed in Section 5.3.2.4). By increasing the $pass\#$ in Pick2RSS function, the size of the RSS subsets that are being challenged between the two devices becomes smaller. The goal of decreasing the subsets size is to ensure the similarity between the two RSS profiles even for the small portions.

If the two devices successfully execute the protocol (the Pick2RSS function returns $null$ since the $subsetSize < \beta$), the client has to check if the channel is dynamic or static (lines 18 and 19 of Algorithm 5) by calculating the Shannon entropy of $GOprfl$. If $GOprflEntropy$ is greater than ε, this indicates a dynamic channel. Accordingly, EvilDirectHunter protocol stops and establishes the P2P group (line 20 of Algorithm 5). Otherwise, an additional detection phase is executed for the static environments.

Algorithm 7 Pick Two Random RSS Subsets

```
1: function Pick2RSS ($pass\#, rssProfile, 2^\eta$)
2: $subsetSize \leftarrow 2^\eta / 2^{pass\#}$
3: $total \leftarrow 2^\eta / subsetSize$
4: Divide $rssProfile$ evenly into $\{rss_1, rss_2 ... rss_{total}\}$
5: $unused \leftarrow \{rss_1, rss_2 ... rss_{total}\} - rss_i$, s.t. $rss_i$ is tagged
6: if $size(unused) >= 2$ then
7:     Pick $\{Rss_I, Rss_{II}\}$ randomly from $unused$
8:     Tag $Rss_I$ and $Rss_{II}$ as used subsets
9:     Find $i$ & $j \in \mathbb{N}$, s.t. $Rss_I$ & $Rss_{II}$ are the $i_{th}$ & $j_{th}$ subsets of $\{rss_1, rss_2 ... rss_{total}\}$,
       respectively
10:    return $\langle pass\#, Rss_i, Rss_j \rangle$
11: else
12:    if $subsetSize >= \beta$ then
13:        $pass\#$++
14:        goto 2
15:    else
16:        return $null$
```

2) Phase II of EvilDirectHunter for static environments: The solution we propose for detect-

Algorithm 8 Get the RSS Subset

```
1: function GetSubset ($pass\#, rssProfile, 2^{\eta}, index$)
2: $subsetSize \leftarrow 2^{\eta}/2^{pass\#}$
3: $total \leftarrow 2^{\eta}/subsetSize$
4: Divide $rssProfile$ evenly into $\{rss_1, rss_2...rss_{total}\}$
5: Find the $rss_i$ in $\{rss_1, rss_2...rss_{total}\}$, s.t. $i = index$
6: Tag $rss_i$ as a used subset
7: return $rss_i$
```

ing EvilDirect GO in static environments makes use of Multi-Dimensional Scaling (MDS). MDS [60] is a class of statistical techniques used to discover relationships in a set of data. The basic idea is that given n objects and a numerical $n \times n$ proximity matrix representing inter-object dissimilarities, an equivalent representation of n points in m-dimensional space can be found whose inter-point distances are proportional to the similarities. MDS algorithm has been used for wormhole localization in mobile ad hoc networks [134]. The proximity matrix in [134] represents the travel distances of the mobile nodes. Due to the fact that there are low variations in the RSS levels in static environments (Figure 5.4f), and that the RSS level decreases 6.02 dBm each time the distance from the source is doubled [135], we decided to use the RSS levels between the clients and the legitimate GO to build their proximity matrix.

Each client broadcasts a V vector (line 22 of Algorithm 5), which contains the average of RSS values of the last five packets that were overheard by it when other clients exchanged packets with the GO before and after each client receives its invitation acceptance (i.e., Probe Res {PBC}) from the GO. For example, if we have two clients (A and B), client A's $V = [RSS_B^A, RSS_{\widehat{B}}^{\widehat{A}}, RSS_{GO}^A, RSS_{\widehat{GO}}^{\widehat{A}}]$ (where $\widehat{.}$ represents the device after receiving the invitation acceptance). RSS_B^A is the average of the RSS values of the last five packets that were overheard by client A when client B exchanged packets with the legitimate GO before it receives the invitation acceptance. $RSS_{\widehat{B}}^{\widehat{A}}$ is the average of the RSS values of the last five packets that were overheard by client A when client B exchanged packets with the GO after it receives the invitation acceptance. RSS_{GO}^A is the average of the RSS values of the last five packets that were received by client A from the GO

$$\begin{pmatrix} 0 & R_B^A & \text{N} & \text{N} & R_{GO}^A & \text{N} \\ & 0 & \text{N} & \text{N} & R_{GO}^B & \text{N} \\ & & 0 & R_{\widehat{B}}^{\widehat{A}} & \text{N} & R_{\widehat{GO}}^{\widehat{A}} \\ & & & 0 & \text{N} & R_{\widehat{GO}}^{\widehat{B}} \\ & & & & 0 & \text{N} \\ & & & & & 0 \end{pmatrix}$$

Figure 5.5: D matrix. *Reprinted with permission from [52].*

before it receives its invitation acceptance. $RSS_{\widehat{GO}}^{\widehat{A}}$ is the average of the RSS values of the last five packets that were received by client A after it receives its invitation acceptance. Client B's $V = [RSS_A^B, RSS_{\widehat{A}}^{\widehat{B}}, RSS_{GO}^B, RSS_{\widehat{GO}}^{\widehat{B}}]$. Each client builds a matrix D (symmetric RSS distance matrix) based on its V and the received V's from other clients. For our example above, the D matrix for client A is shown in Figure 5.5. N indicates the absence of the RSS value (e.g., $RSS_{\widehat{A}}^A$, $RSS_{\widehat{B}}^A$, $RSS_{\widehat{GO}}^A$). We use R instead of RSS, for condensed notation. The rows represent the RSS distance vectors for devices $A, B, \widehat{A}, \widehat{B}, GO, \widehat{GO}$, respectively. These vectors are to devices $A, B, \widehat{A}, \widehat{B}, GO, \widehat{GO}$, respectively ($D$'s columns).

The absolute value of D matrix, $|D|$, is passed to the MDS algorithm (line 24 of Algorithm 5). The steps of the MDS algorithm are presented in [60]. The output of MDS is Y, the set of 2-dimensional coordinates that describes the similarities between each device's RSS values before and after it receives its invitation acceptance from the GO. The EvilDirect GO can be detected due to the high difference between his RSS values (received/overheard by all clients after they receive the acceptance for their invitations), and the legitimate GO's RSS values (received/overheard by all clients before they receive the acceptance for their invitations). This high difference is due to the fact that the EvilDirect GO has a different physical location corresponding to all clients in the P2P group compared to the legitimate GO physical location. On the other hand, since the clients are static, and there are few intermediate mobile objects in static environments, the differences between clients' RSS values (before and after they receive the invitation acceptance from the GO) are low. Accordingly, any client can detect the EvilDirect GO attack by comparing the difference between the GO's RSS values (before and after receiving the invitation acceptance) to the average of differences between all clients' RSS values (before and after they receive the acceptance for their invitations).

For any client, $d(R_{GO}, R_{\widehat{GO}})$ is the Euclidean distance between the two RSS values of the GO before and after it receives its invitation acceptance (line 25 of Algorithm 5). $Avg_{i:1\to n}[d(R_{C_i}, \widehat{R_{C_i}})]$ is the average of Euclidean distances of all clients' RSS values before and after they receive the acceptance for their invitations. The client compares $d(R_{GO}, R_{\widehat{GO}})$ and $Avg_{i:1\to n}[d(R_{C_i}, \widehat{R_{C_i}})]$ (line 26 of Algorithm 5) to detect the EvilDirect attack. In the normal scenarios, $d(R_{GO}, R_{\widehat{GO}}) \leq Avg_{i:1\to n}[d(R_{C_i}, \widehat{R_{C_i}})]$, which indicates a slight change in the GO's RSS values.

5.2.3 EvilDirectHunter Parameter Tuning

This section presents the measurement and tuning of ε and τ thresholds, respectively.

1. **Measuring the threshold parameter ε:** We repeated each experiment presented in Figures 5.4b and 5.4d 267 times. We aim to experimentally investigate the suitable value of ε, which is used to distinguish between static and dynamic environments. Accordingly, we measured the average value of $GOprflEntropy$ of these 534 experiments and we set ε to 4.53. We acknowledge that the value of ε might be different for other environments, and it requires further study.

2. **Tuning the threshold parameter τ:** τ is used by the client to check if the GO is a legitimate or an EvilDirect device (line 8 of Algorithm 5). We aim to find the best value of τ for all passes when EvilDirectHunter runs on same/different smartphones.

   We repeated each experiment presented in Figure 5.4 267 times (total = 801 experiments) to generate 205,056 RSS readings using Google Nexus 5 smartphone for the client and the GO. Each experiment generates 256 RSS readings for the GO, RSS_{GO}, and the client, RSS_{Client}. We assume and validate in Section 5.3.2.3 that the distribution of any RSS reading (say m) of the client is $rss_m \sim \mathcal{N}(\mu, \sigma^2)$. We calculated [$RSS_{GO}$ - RSS_{Client}] for each experiment to create a vector of 256 RSS level differences (dBm). We plotted the histogram for the 801

Table 5.2: τ value for each pass (dBm). *Reprinted with permission from [52].*

Pass#	Nexus5 v.s. Nexus5	S2 v.s. Nexus5
1	809.6	923.8
2	404.9	462.2
3	202.5	231.1
4	101.2	115.6

[RSS_{GO} - RSS_{Client}] vectors that were generated from these 801 experiments. Figure 5.6a shows the histogram of the 205,056 (801×256) RSS level differences. It is clear that the difference between the GO's and client's RSS readings (RSS_{GO} - RSS_{Client}) is distributed $\sim \mathcal{N}(\mu, \sigma^2)$.

In order to calculate the best value of τ for all passes when EvilDirectHunter executes between two Google Nexus 5 smartphones, we run a Monte Carlo simulation with the following steps. First, we generated 256 RSS values using the normal distribution (i.e., which we assume and validate in Section 5.3.2.3) for the client device (RSS_{Client}). Second, we generated 256 RSS values for the GO, RSS_{GO} = RSS_{Client} + x. x is a random number generated from the normal distribution presented in Figure 5.6a. Third, we calculated the $d^2(Rss_I, Rss_{II})$'s for all passes for both the client and the GO. Fourth, we calculated the differences between the client d^2's and the GO d^2's for all passes. Fifth, we repeated the previous steps 100,000 times, and we calculated the average of the differences between the client d^2's and the GO d^2's. Table 5.2 presents the τ values (dBm) for all passes when EvilDirectHunter executes on Google Nexus 5 smartphones.

We repeated the same 801 experiments using Samsung Galaxy S2 for the client, and Google Nexus 5 for the GO. Figure 5.6b shows the histogram of the 205,056 RSS level differences of these experiments. Moreover, we repeated the same steps of the Monte Carlo simulation for these smartphones. However, we used the normal distribution of Galaxy S2's RSS level, and the normal distribution presented in Figure 5.6b. Table 5.2 shows the τ values (dBm)

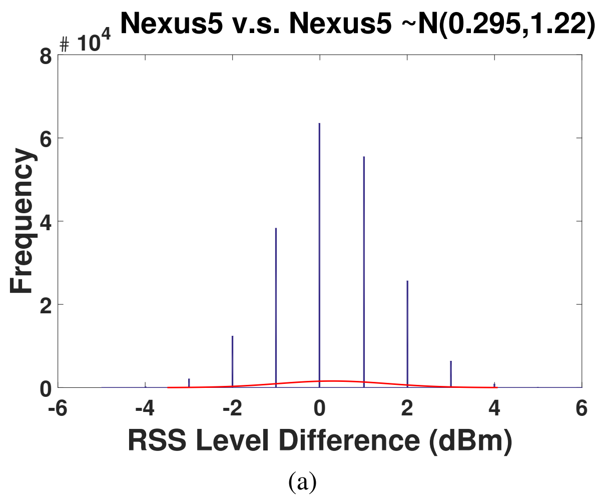


Figure 5.6: Histograms of the RSS level difference between (a) Two Google Nexus 5 smartphones. (b) Samsung Galaxy S2 and Google Nexus 5 smartphones. *Reprinted with permission from [52].*

for all passes when EvilDirectHunter executes on Samsung Galaxy S2 and Google Nexus 5 smartphones. We use the τ values in Table 5.2 in our experiments that we will present in Section 6.3.3.

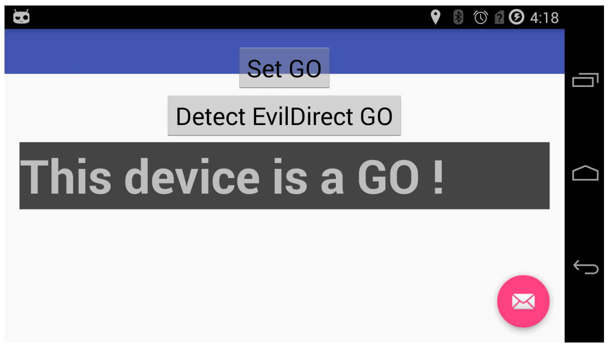
5.3 Evaluation

5.3.1 Implementation and Experimental Setup

This section presents the implementation of our EvilDirectHunter and our experimental setup.

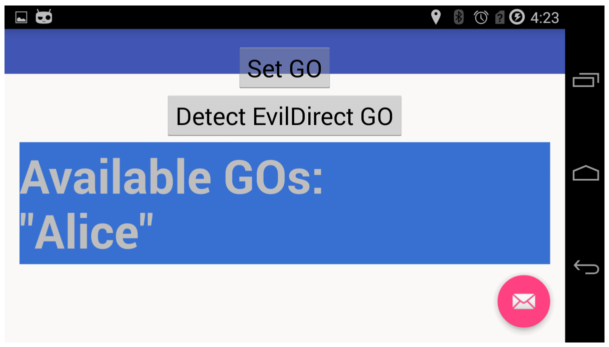
In order to record the RSS profile for the client and the GO, we downloaded, modified and built the LineageOS [124] Android kernel code for Google Nexus 5 and Samsung Galaxy S2 s-martphones. The wireless card drivers of these smartphones report the RSS values as integers from -25 to -100 dBm. The Android kernel code that is responsible for establishing the Wi-Fi Direct connection for these smartphones implements the “discovery algorithm” (illustrated in Section 5.1).

We modified the Android kernel code such that each device stores the RSS value of the received probe request/response frames during its listen state. In the original Android kernel code, the amount of time for the search and listen states is randomly distributed between 100 and 300 msec.



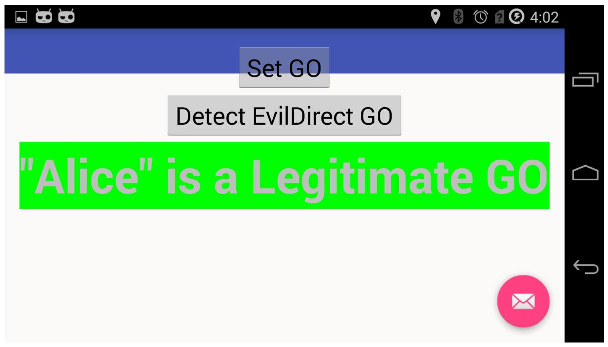
This device is a GO !

(a)



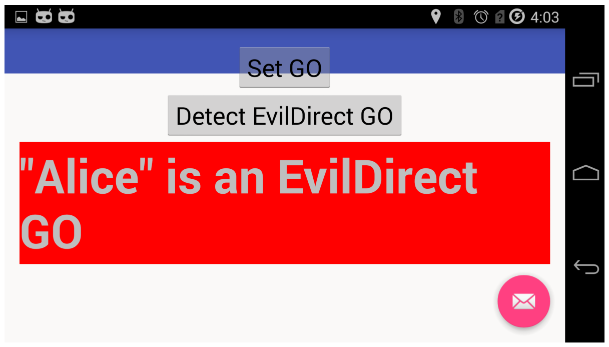
Available GOs:
"Alice"

(b)



"Alice" is a Legitimate GO

(c)



"Alice" is an EvilDirect GO

(d)

Figure 5.7: "EvilDirectHunter" (a) on GO. (b) on client. (c) Finding legitimate GO. (d) Finding EvilDirect GO. *Reprinted with permission from [52].*

Since the client and the GO respond with probe responses only during their listen states, we set the duration of the listen and search states to 100 msec to get 10 RSS readings/sec. Each device might be the initiator of the probe request frames (during its search state), or the responder with the probe response frames (during its listen state). The initiator injects the header of the probe request frame with a specific sequence number in order to handle the frame losses and retransmissions. If the probe response frame has the same sequence number as the request frame, the initiator records the RSS value of that frame. Moreover, If the initiator receives two probe response frames with the same sequence number, it records both of them. Also, If the responder receives two probe request frames with the same sequence number, it records both of them.

Moreover, we implemented the "EvilDirectHunter" Android app, which helps the smartphone user in detecting the EvilDirect GOs within his wireless communication range. Figure 5.7a shows the "EvilDirectHunter" app when it runs on the GO smartphone. In Figure 5.7b, our app lists the active GO devices (i.e., Alice) for the client. The user has to click the "Detect EvilDirect GO" button to run our EvilDirectHunter, and determine if Alice is a legitimate or an EvilDirect GO (Figures 5.7c and 5.7d, respectively).

We performed three experiments using four smartphones in different environments as shown in Figure 5.8a. $d1$ (the distance between each client and the legitimate GO) = 10 m. $d2$ is the distance between the legitimate GO and the attacker. In each experiment, both clients execute the Device Discovery and Service Discovery phases with the legitimate GO for 30 seconds. The clients and GO's smartphones record the RSS readings of their last 256 packets. The attacker's smartphone overhears the packets that were sent from each client to the legitimate GO, and records the RSS readings of the last 256 packets to launch his attack. In the following, we describe our experiments:

1. **Experiment A**: This experiment was conducted in a crowded cafeteria during a busy lunch hour (12:00 PM - 1:00 PM). There were around 70 customers inside the cafeteria.

2. **Experiment B**: This experiment was conducted in a cafeteria after the lunch hour (1:30 PM - 2:30 PM). There were around 25 customers inside the cafeteria.

3. **Experiment C**: This experiment was conducted in the study area of a library on a weekday evening (10:00 PM - 11:00 PM). There were 5 students on the study seats.

We evaluated EvilDirectHunter using the following metrics:

1. **Detection Rate**: the ability of EvilDirectHunter on detecting the attempts of the EvilDirect attacker (i.e., true positive rate), ideally 100%.

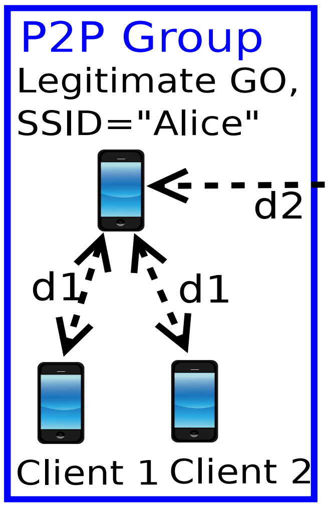
2. **False Positive Rate**: the rate of claiming a legitimate GO as an EvilDirect GO, ideally 0%.

3. **Execution Time (sec)**: the time delay resulted from running EvilDirectHunter in smartphones.

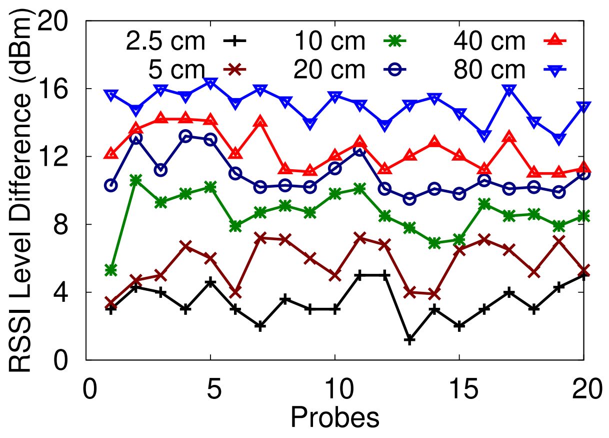
5.3.2 Security Analysis

5.3.2.1 Physical proximity attack defense

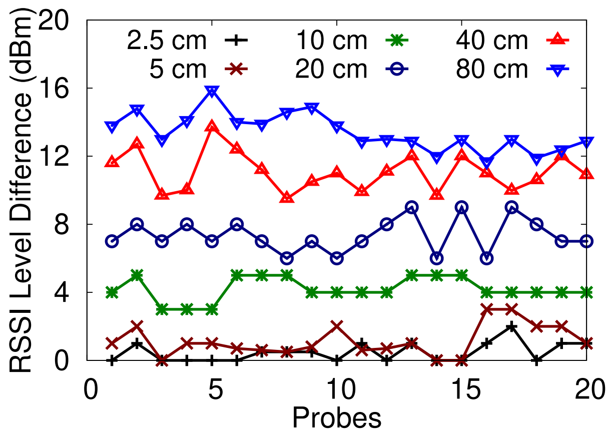
[50] proved that Eve, who is more than half a wavelength away from the client and the legitimate GO, experiences fading channels to them that are statistically independent from the fading between the client and the legitimate GO. In order to investigate the value of $d2$ distance that enables Eve to obtain the same RSS readings as the legitimate GO, we performed experiments A, B, and C while we varied $d2$ distance between 2.5 cm to 80 cm. Eve's smartphone can overhear the packets that were sent from Client 1 (Figure 5.8a) to the legitimate GO, and records the RSS readings of the last 256 packets. We plotted the absolute difference between the RSS readings of the first 20 packets (for illustration purposes) that were received by the legitimate GO from Client 1, and the RSS readings that were recorded by Eve for the same packets. As shown in Figures 5.8b, 5.8c, and 5.8d, in order to make the RSS readings for Eve similar to the RSS readings at the legitimate GO's location, $d2$ distance has to be at most 5 cm ($d2 \leq 5$ cm). For $d2 > 5$ cm, the difference in the RSS readings increases, and this makes Eve's task of obtaining the legitimate GO RSS readings more difficult.



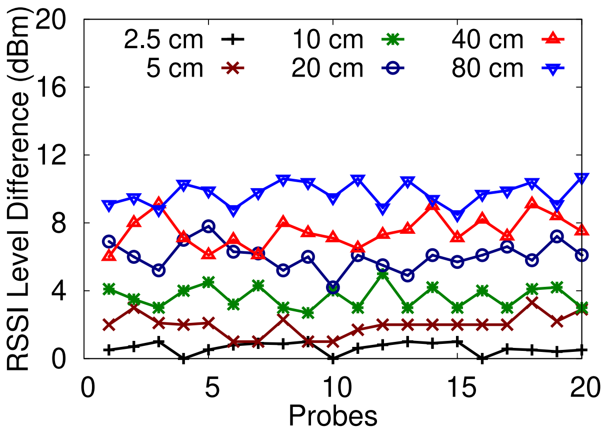
(a)



(b)



(c)



(d)

Figure 5.8: (a) Experiments setup. RSS Differences between Eve and Legitimate GO for experiments (b) A. (c) B. (d) C. *Reprinted with permission from [52].*

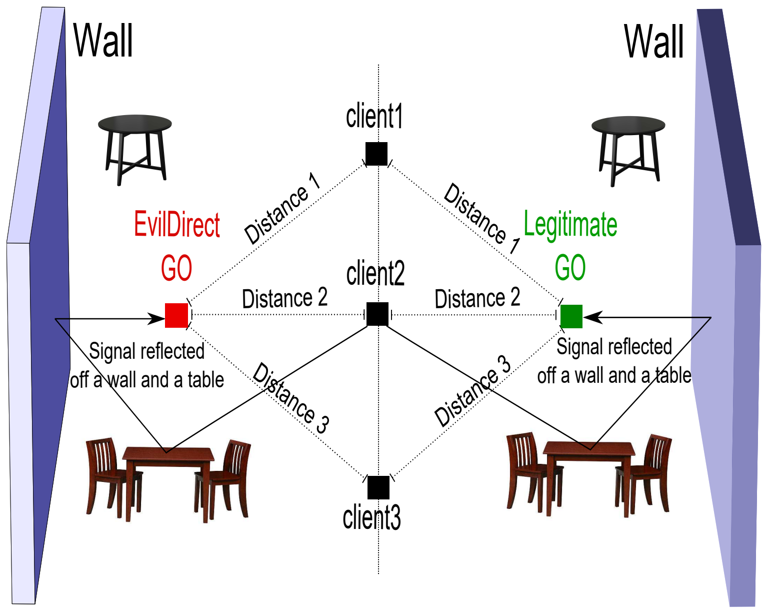


Figure 5.9: Successful Predictable Channel Attack. *Reprinted with permission from [52].*

5.3.2.2 Predictable channel attack defense

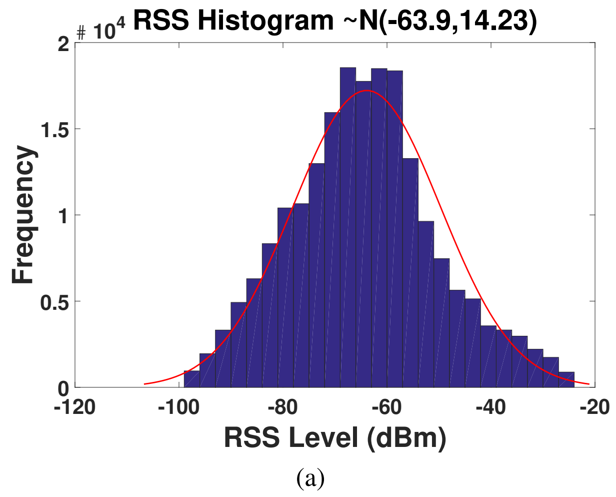
The RSS level in free space environments (the open air or the anechoic chamber) follows the inverse square law (the RSS level decreases 6.02 dBm each time the distance from the source is doubled) [135]. It is easy for Eve to predict the RSS level at either the client's or the legitimate GO's locations in such environments. In the real-world, Eve launches her attack in public, indoor areas. In such areas, there are many static and moving objects (e.g., tables, walls, doors, passengers, customers, and students). As a result, the radio waves are reflected, diffracted, and scattered many times. Accordingly, the inverse square law is not strictly applicable, and the RSS level is very random.

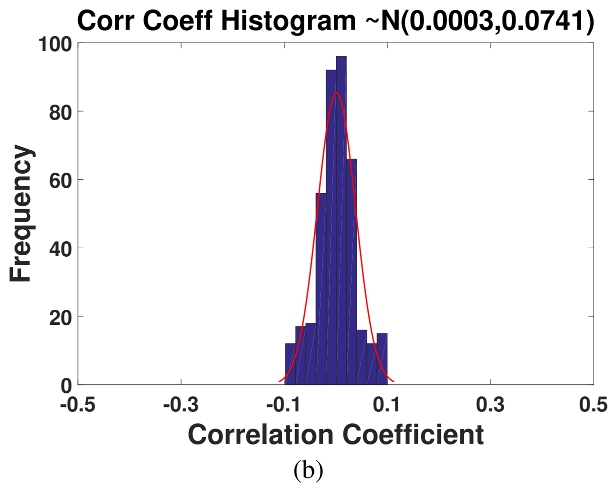
In static areas, due to the lack of randomness in the wireless channel, there are few vatiations in the RSS profiles. In such areas, Eve has two approaches to obtain the same RSS profile of the client or the legitimate GO. First, is to collect many RSS profiles that simulate the client and legitimate GO communications at different physical locations. Eve uses these RSS profiles during her attack based on the physical locations of the clients and legitimate GO. However, the clients use a random transmission power each time they execute the Device Discovery and Service Discovery phases with the legitimate GO (shown in Figure 5.3. As a result, it is hard for Eve to predict the transmission power that the client uses. Second, is to pass the MDS detection by locating herself in a physical location in which her RSS values (received/overheard by all clients after they receive the acceptance for their invitations from her) equal to the legitimate GO's RSS values (received/overheard by all clients before they receive the acceptance for their invitations from Eve). This physical location has to be symmetric to the legitimate GO location corresponding to all clients in terms of distances and intermediate objects (to ensure that Eve has the same multi-path channels with all clients as the legitimate GO). In order to make the distances between Eve and any client equal to the distances between the client and the legitimate GO, all clients have to be in the same geometric plane/line, as shown in Figure 5.9. However, even if the clients exist in the same geometric plane (rare when the number of clients ≥ 4), the intermediate objects between Eve and all clients have to be similar to the corresponding intermediate objects between the legitimate GO and all clients, as shown in Figure 5.9 (also rare in public areas).

5.3.2.3 $\oplus$ *result guessing attack defense*

In order to validate whether $\oplus$ is a suitable statistical metric that can securely measure the similarity between the clients' and the legitimate GO's RSS profiles, we repeated each experiment presented in Figure 5.4 267 times to generate 205,056 RSS readings. First, we want to find the statistical distribution of the RSS readings. Figure 5.10a shows the histogram of the client RSS readings for the 801 experiments. The distribution of any RSS reading, say m, of the client, $rss_m \sim \mathcal{N}(\mu, \sigma^2)$. The histogram of the GO RSS readings is similar to Figure 5.10a. Second, we validate $\oplus$ for EvilDirectHunter.

The Pearson correlation coefficient between X and Y sets of data, each of size q with $\overline{X}$ and $\overline{Y}$ as mean values, respectively, is defined as: $\rho(X,Y) = \frac{\sum_{i=1}^{q}\left[(X_i-\overline{X})\times(Y_i-\overline{Y})\right]}{\sqrt{\sum_{i=1}^{q}(X_i-\overline{X})^2}\times\sqrt{\sum_{i=1}^{q}(Y_i-\overline{Y})^2}}$. At each pass of the EvilDirectHunter protocol, the client divides its $rssProfile$ into two subsets (i.e., $rssProfile_{1st}$ = $[RSS_1,\ RSS_2\ ...RSS_k]$ and $rssProfile_{2nd}$ = $[RSS_{k+1},\ RSS_{k+2}\ ...RSS_{2k}]$). For $rssProfile$ with 256 readings, $k = 256$ for the first pass. In order to validate whether the Pearson correlation coefficient is a suitable statistical metric for our protocol, we calculate the





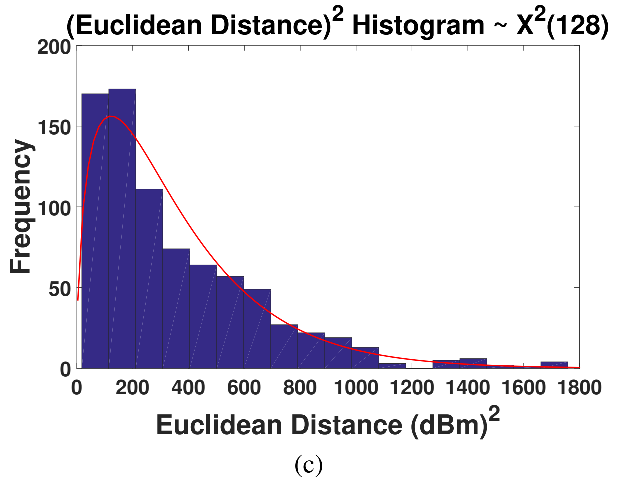


Figure 5.10: Google Nexus 5's histograms of the (a) RSS readings. (b) Correlation Coefficients between two RSS Subsets. (c) $\oplus$ between two RSS subsets. *Reprinted with permission from [52].*

$\rho(rssProfile_{1st}, rssProfile_{2nd})$ during the first pass of the client RSS readings for the 800 experiments. We want to find the statistical distribution of the ρ calculations.

Figure 5.10b shows the histogram of the ρ calculations for the 801 experiments during the first pass. As shown in this figure, the value of ρ is statistically distributed according to $\mathcal{N}(0.0003, 0.0741)$. Even though the typical value of ρ has to be in $[-1.0, 1.0]$, the value of ρ for these experiments is in $[-0.1, 0.1]$. This is due to the fact that RSS_i's in both $rssProfile_{1st}$ or $rssProfile_{2nd}$ subsets are statistically independent (i.e., the events that affect the RSS readings such that the mobility of either the intermediate objects or the endpoints are uncorrelated). As a result, we find that the correlation coefficient metric is unsuitable for our EvilDirectHunter protocol due to the easiness of predicting its value by the attacker, Eve. We will discuss this in more detail in Section 5.3.2.3.

The Euclidean distance, $d(X, Y)$, between X and Y sets of data is $\sqrt{\sum_{i=1}^{q}(X_i - Y_i)^2}$, s.t. q is the size of each set. At each pass of EvilDirectHunter, the client divides the $GOprfl$ into two subsets (Rss_I = $[rss_1, rss_2 ... rss_k]$ and Rss_{II} = $[rss_{k+1}, rss_{k+2} ... rss_{2k}]$). For $GOprfl$ with 256 readings, $k = 128$ for the first pass. We want to find the statistical distribution of $d^2(Rss_I, Rss_{II})$, which is the square of the Euclidean distance. Since each $rss_m \sim \mathcal{N}(\mu, \sigma^2)$, then $(rss_m - rss_r) \sim \mathcal{N}(0, 2\sigma^2)$ (m - r = k). If we assume that $Z_1, Z_2 ... Z_k$ are k independent normal random variables that represent $[(rss_1 - rss_{k+1}), (rss_2 - rss_{k+2}) ... (rss_k - rss_{2k})]$ (i.e., the square of the Euclidean distance between Rss_I and Rss_{II}), then: $d^2(Rss_I, Rss_{II})$ = $\sum_{i=1}^{k} Z_i^2$ = $\sum_{i=1}^{k}[\mathcal{N}(0, 2\sigma^2)]^2$ = $\sum_{i=1}^{k}[\sqrt{2}\sigma\mathcal{N}(0,1)]^2$ = $2\sigma^2 \sum_{i=1}^{k}\mathcal{N}(0,1)^2$.

The sum of the squares of k independent standard normal random variables ($\sum_{i=1}^{k}\mathcal{N}(0,1)^2$) is distributed according to chi-squared distribution [136] with k degrees of freedom, $\chi^2(k)$. Due to the multiplication by $2\sigma^2$, the value of $d^2(Rss_I, Rss_{II})$ is distributed according to the scaled version of chi-squared distribution with k degrees of freedom. Figure 5.10c shows the histogram of $d^2(Rss_I, Rss_{II})$ for the 801 experiments with 128 (i.e., size of Rss_I & Rss_{II} during the first pass) degrees of freedom.

The hardness of guessing the result of $\oplus$ depends on its statistical distribution. We use Shannon entropy to measure the randomness of the statistical distributions. The hardness of guessing the

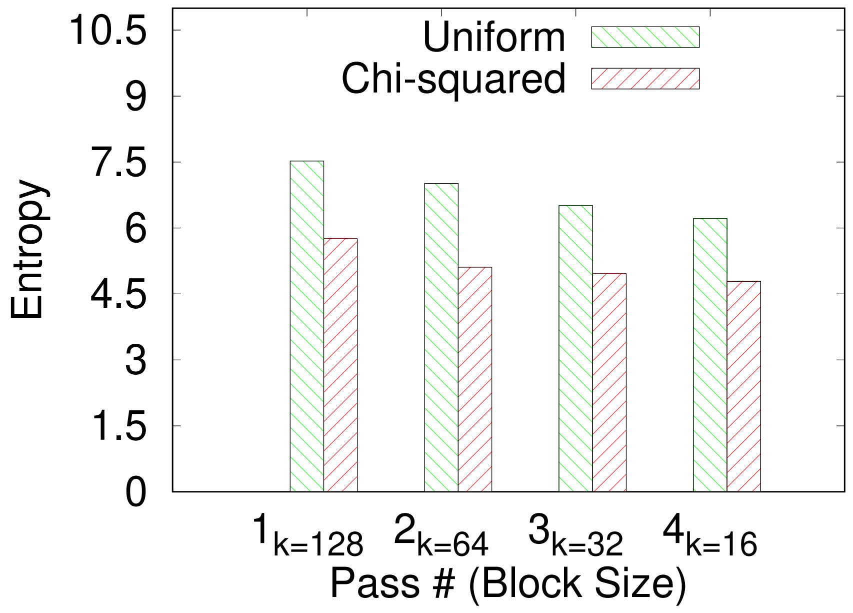


Figure 5.11: Entropy of the Uniform and $\chi^2(k)$ Distributions. *Reprinted with permission from [52].*

result of $\oplus$ increases when the entropy of its statistical distribution increases. A valid question follows: what is the maximum entropy of $\oplus$'s statistical distribution? In probability theory and statistics, the uniform distribution is the maximum entropy distribution on any interval $[a, b]$ [137]. In order to investigate the hardness of guessing the result of $\oplus$, we compared the entropy of $\oplus$'s statistical distribution with the entropy of the uniform distribution for all passes.

For the first pass, the statistical distribution of $\oplus$ (Figure 5.10c) is distributed on the interval $[0, 1800]$ according to the scaled version of chi-squared distribution with 128 degrees of freedom, $\chi^2(128)$. The entropy of $\chi^2(k) = \frac{k}{2} + ln(2\Gamma(\frac{k}{2})) + (1 - \frac{k}{2})\psi(\frac{k}{2})$ [136]. The entropy of the uniform distribution on the interval $[0, 1800]$ is $ln(1800) = 7.5$. We found the distributions of $\oplus$ for other passes (when $pass\# = 2, 3, 4$) in the same way we did for the first pass. Then, we compared the entropies of these distributions with the entropies of the uniform distributions on the same intervals. As shown in Figure 5.11, the entropy of the distribution of $\oplus$ is at least 75% of the entropy of the uniform distribution.

5.3.2.4 Client spoofing attack defense

EvilDirectHunter enables the legitimate GO to discover and stop Eve while she launches the client spoofing attack. As we described in Section 5.2.2, the client and the legitimate GO incrementally prove the mutual knowledge of the RSS profile by exchanging challenge and response packets. Since it is difficult for Eve to create an RSS profile that is similar to the client profile and to guess the result of $\oplus$ (as shown in Sections 5.3.2.1 and 5.3.2.3, respectively), Eve cannot pursue EvilDirectHunter and learn the results of the $\oplus$ metric.

5.3.2.5 Replay attack defense

If Eve replays some/all of the client's probe-requests/GAS-requests or the legitimate GO's probe-responses/GAS-responses, the RSS profile that the client builds will be a mixture of the legitimate GO's messages and Eve's replayed messages. Indeed, the RSS profile at the client is created based on the multi-path of the channels between the client and the legitimate GO, and between the client and Eve. On the other hand, the RSS profile at Eve is created based on the multi-path of the channels between Eve and the client, and between Eve and the legitimate GO. Accordingly, the RSS profile that is created at Eve is different from the RSS profile at the client (unless Eve is very close to the client or the legitimate GO, as shown in Section 5.3.2.1). As a result, if Eve launches her attack, she will be detected by EvilDirectHunter. On the other hand, if Eve does not launch her attack and the client tries to connect to the legitimate GO, our algorithm will mistakenly claim the legitimate GO as an EvilDirect GO because the RSS profile that the legitimate GO builds will be a mixture of the client's messages and Eve's replayed messages. Essentially, if Eve causes a replay attack, she will not be able to successfully launch an EvilDirect GO attack. However, her replay attack will cause a denial-of-service attack (DoS) against the legitimate GO, which is not the interest of Eve.

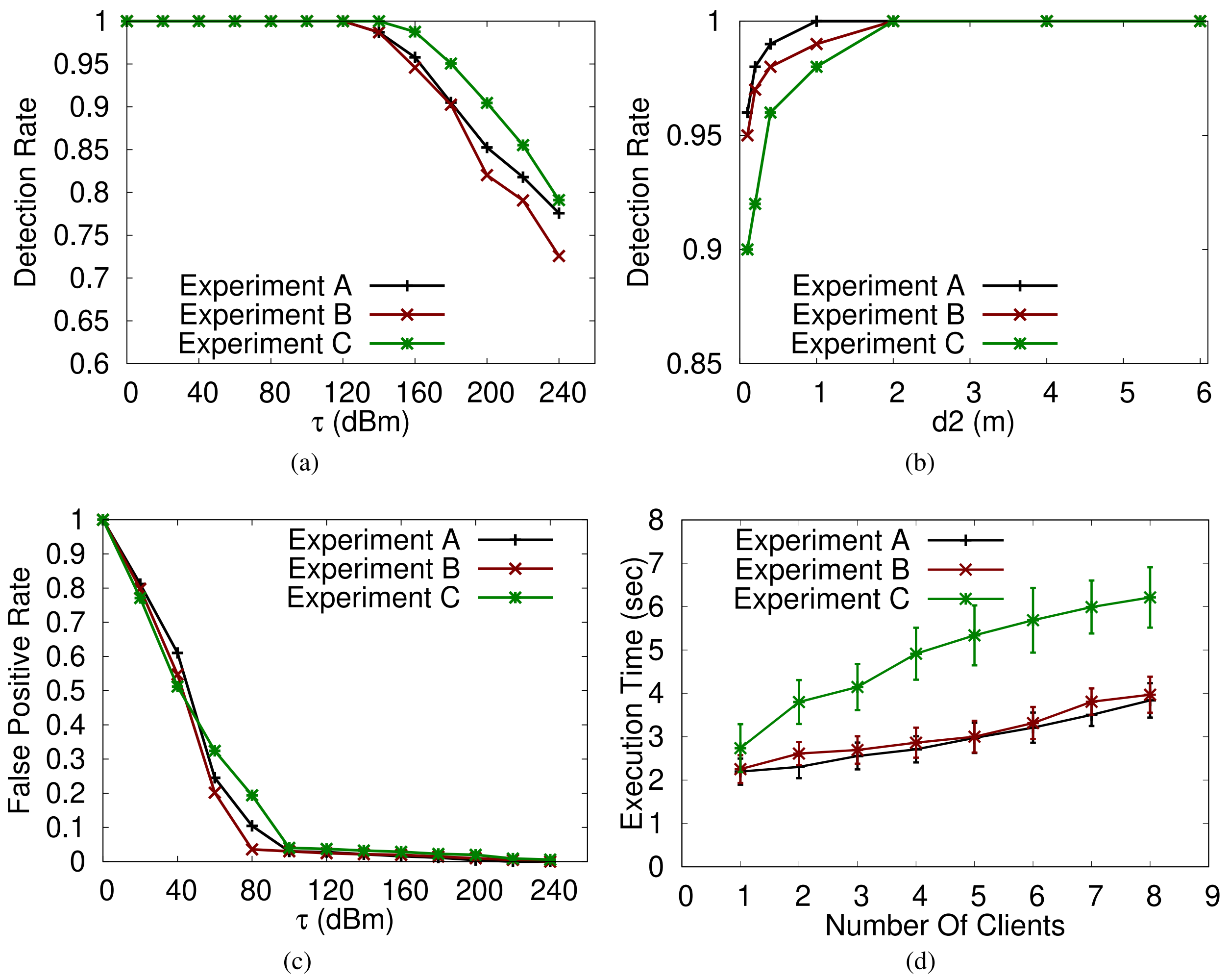


Figure 5.12: Experiments A, B, and C on same hardware. (a) Detection Rate for different τ's. (b) Detection Rate for different $d2$'s. (c) False Positive Rate for different τ's. (d) Execution Time. *Reprinted with permission from [52].*

5.3.3 Performance Evaluation

5.3.3.1 EvilDirectHunter on same smartphones

We performed the experiments with Google Nexus 5 smartphones for the clients, the legitimate GO, and the attacker.

1. **Detection Rate**: in Section 5.2.3, we tuned the values of τ when EvilDirectHunter runs on Google Nexus 5 smartphones. The value of τ impacts the detection rate of our protocol. In

order to investigate the best values of τ from the detection rate perspective, we performed experiments A, B, and C with $d2$ = 2 m, and ε = 4.53. We varied the value of τ for the fourth pass to demonstrate its influence on the detection rate. In each experiment, the attacker used the RSS readings of the packets that he overheard while each client exchanged packets with the legitimate GO. Then, both clients executed EvilDirectHunter with the attacker's smartphone. We counted the number of times that EvilDirectHunter was able to detect the attacker as EvilDirect GO for both clients. We repeated each experiment 40 times, and we averaged these 40 runs for each experiment.

Figure 5.12a shows the detection rate of EvilDirectHunter for both clients for experiments A, B, and C. As shown in this figure, the detection rate of EvilDirectHunter decreases when τ increases. Indeed, the values of τ that we tuned from the Monte Carlo simulation for Google Nexus 5 smartphones in Table 5.2 achieve a 100% detection rate for all experiments. The additional phase for static environments (Section 5.2.2) executed for all runs of experiment C.

Moreover, we investigated the effect of $d2$ distance on the detection rate. We performed experiments A, B, and C while we varied $d2$ distance between 10 cm to 6 m to investigate its effect on the detection rate. For each experiment, we repeated the same steps above. However, we used the τ values for the four passes presented in the second column of Table 5.2. We repeated each experiment 40 times, and we averaged these 40 runs for each experiment. Figure 5.12b shows the detection rate of EvilDirectHunter for both clients for different $d2$ values. As is clear in this figure, the detection rate is 100% for A, B, and C experiments when $d2 > 1$ m.

2. **False Positive Rate**: in order to investigate the best values of τ from the false positive rate perspective, we performed experiments A, B, and C with ε = 4.53. We varied the value of τ for the fourth pass to demonstrate its influence on the false positive rate. In each experiment, both clients executed EvilDirectHunter with the legitimate GO. We counted the number of

times that EvilDirectHunter mistakenly detected the legitimate GO as EvilDirect GO for both clients. We repeated each experiment 40 times, and we averaged these 40 runs for each experiment. Figure 5.12c shows the false positive rate when we vary the value of τ for the fourth pass. As is clear in this figure, the false positive rate decreases when τ increases. Our goal is to achieve a 0% false positive rate for our protocol. Even though increasing τ values achieves a very low false positive rate, we are unable to use high τ values because they will lower the detection rate as is presented in Figure 5.12a. Thus, there is a tradeoff between the detection rate and the false positive rate. We found that the highest false positive rate for all experiments using the τ values for Google Nexus 5 smartphones in Table 5.2 is $\sim$4.0%.

3. **Execution Time**: in order to evaluate the execution time of EvilDirectHunter, we repeated experiments A, B, and C while we increased the number of clients between one and eight devices. These clients simultaneously run EvilDirectHunter with the legitimate GO. We measured the execution time needed to run EvilDirectHunter on all clients. We repeated each experiment 40 times, and we averaged these 40 runs for each experiment. Figure 5.12d shows the execution time for all experiments, with error bars showing standard deviation. As is clear in this figure, by increasing the number of clients, the execution time increases linearly. Accordingly, the legitimate GO is able to run EvilDirectHunter, and reply to all clients' challenges in a short time. Due to the additional phase for static environments, the execution time for experiment C is higher than the execution times for experiments A and B.

5.3.3.2 EvilDirectHunter on different smartphones

We investigated the robustness of EvilDirectHunter on smartphones with different hardware components by repeating the same steps of all experiments presented in Section 5.3.3.1. However, we used Samsung Galaxy S2 smartphones for the two clients, and Google Nexus 5 smartphones for both the legitimate GO and the attacker. Figure 5.13a shows that the detection rate of EvilDirectHunter for both clients decreases when the τ value for the fourth pass increases. The values of τ that we calculated from the Monte Carlo simulation for Google Nexus 5 and Samsung Galaxy

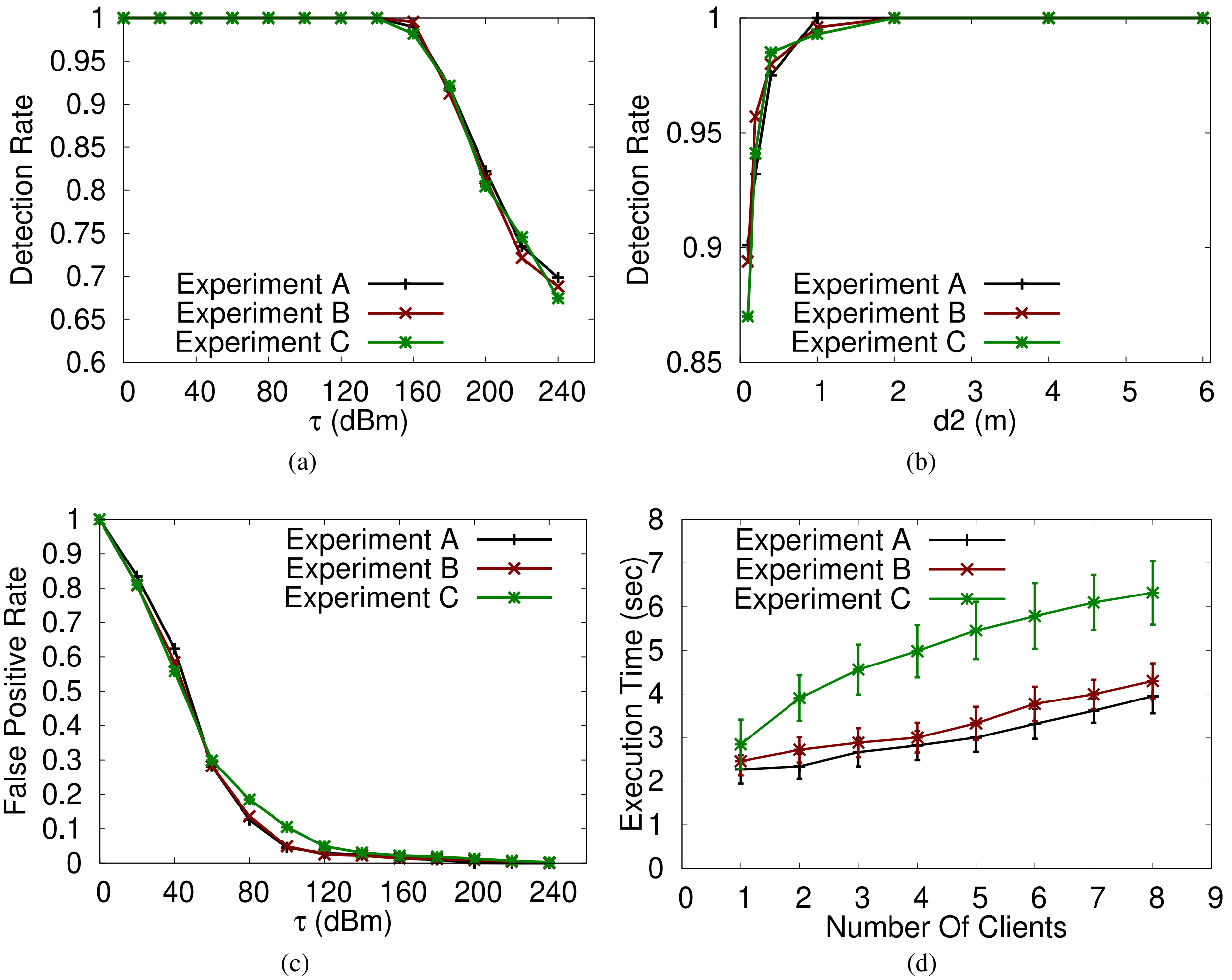


Figure 5.13: Experiments A, B, and C on different hardware. (a) Detection Rate for different τ's. (b) Detection Rate for different $d2$'s. (c) False Positive Rate for different τ's. (d) Execution Time. *Reprinted with permission from [52].*

S2 smartphones (third column of Table 2) achieve a 100% detection rate for experiments A, B, and C. As shown in Figure 5.13b, the detection rate is 100% for A, B, and C experiments when $d2 > 1$ m. Figure 5.13c shows the false positive rate when we vary the value of τ for the fourth pass. The highest false positive rate for all experiments using the τ values for Google Nexus 5 and Samsung Galaxy S2 smartphones (third column of Table 2) is $\sim$4.5%. The execution time (shown in Figure 5.13d) of EvilDirectHunter on different smartphones is similar to the execution time on similar smartphones (i.e., Figure 5.12d).

6. DETECTING ROUTE HIJACKING ATTACK IN WDONs[*]

In this section, we first show that the Hybrid Routing Protocol (HRP) [26] and the Prophet protocol [27] [28] in WDONs are vulnerable to the *CollusiveHijack* attack. In the CollusiveHijack attack, we assume that Eve has successfully launched either the brute-force/dictionary attack or the EvilDirect attack and connect her devices to the WDON. Or, she has control of some compromised devices that is carried by benign users and infected by a malware a priori. Eve lies about their Inter Contact Times (ICTs) of her compromised/malicious devices. Eve's nodes claim that they meet more frequently than what they really do. Accordingly, Eve hijacks the routes of the legitimate nodes in the network.

Next, in order to address the CollusiveHijack attack in WDONs, we propose to detect the CollusiveHijack attack by employing the Kolmogorov-Smirnov two-sample test (KS2ST). The K-S2ST [54] is a well known test for comparing two statistical distributions. That is, it measures the distance between the empirical distribution functions of two samples to determine whether they have been drawn from the same distribution or not. The KS2ST has been used by previous works for detecting covert channels, detecting selfish wireless nodes, and in intrusion detection systems [56–59]. We design two techniques to detect the CollusiveHijack attack: the Path Detection Technique (PDT) and the Hop Detection Technique (HDT). PDT and HDT offer a trade-off between the *compatibility* with the Bundle Security Protocol (BSP) [55] (if additional steps are required at the intermediate nodes) and the *detection rate* against the CollusiveHijack attack. In PDT, the destination performs a path-wise detection by collecting the packets' delays along the path. The intermediate nodes in the path are authenticated by leveraging the sequential authenticators capability of BSP. The destination uses the KS2ST to test whether the statistical distribution of the packets' delays follows the statistical distribution that is derived from the ICTs of the in-

[*]Reprinted with permission from "CollusiveHijack: A New Route Hijacking Attack and Countermeasures in Opportunistic Networks" by Ala Altaweel, Radu Stoleru, Guofei Gu, and Arnab Kumar Maity, In Proceedings of 2019 IEEE Conference on Communications and Network Security (CNS), pages 1-9, Washington, D.C., USA, 2019, Copyright 2019 by IEEE.

termediate nodes. In HDT, the destination performs a hop-wise detection by requiring additional information from the intermediate nodes (the packets' receiving times). The destination leverages the KS2ST to detect whether each hop in the path is compromised by Eve or not. HDT can achieve a higher detection rate against the CollusiveHijack attack than PDT.

6.1 Background and Motivation

In this section, we show the real-world attack results of the CollusiveHijack attack against HRP and Prophet protocols.

We deploy the HRP and Prophet's daemon services [96] [97] on 11 Asus Eee notebooks that run Ubuntu 14.04 LTS. The wireless cards of the notebooks are set to operate in 2.4 GHz (channel 3) and in ad-hoc mode. Due to space limitation and for the ease of testing, we place all notebooks together and emulate a fully connected multi-hop mesh network by manipulating firewall configurations. To this end, we connect all notebooks to a server through Ethernet cable and issue *iptables* and *ip6tables* commands from the server to create different typologies, as shown in Figure 6.1a. In order to create contact events between the notebooks as it is the case in WDONs, we conducted a dynamic on-off network experiment. First, we created the topology shown in Figure 6.1b. Second, we turned each node on and off randomly (according to an exponential distribution) by issuing *iptables* and *ip6tables* commands with different on-proportion. The expected total duration for one on-off cycle is set to 60 seconds. We generate a data flow from v_1 to v_{11} with 1 KB and set its deadline to 300 seconds. We leveraged HRP's *hrptclient* and IBR-DTN's *dtnsend* and *dtnrecv* for the data flow. We used the default values of $\beta = 0.9$ and $\gamma = 0.999$ for the Prophet protocol. The duration of each experiment is 30 minutes including a warm-up period for 5 minutes. The warm-up period is used by HRP and Prophet to learn about the contact events and the ICTs before starting the data flow.

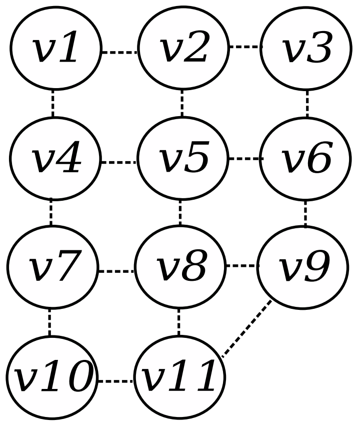
For each experiment, we considered a normal scenario and an attack scenario. During the normal scenario, all nodes are honest about their ICTs. Differently, in the attack scenario, we assigned the same ICTs that were used during the normal scenario among the honest nodes, however, the ICTs of the compromised nodes are multiplied by 0.5. We intend to show that the CollusiveHijack can be successfully launched against HRP and Prophet under diverse network conditions. That is, by varying the on-proportion $\in$ [0.4, 1.0] to emulate the well connected mesh network and the sparsely connected WDON. We tracked the routing paths via *tcpdump* to find the number of replicated packets to the compromised nodes during each experiment. We repeated each experiment 30 times (with different ICTs) and we averaged these 30 runs per each experiment.

Figures 6.1c and 6.1d show the total number of routed packets when v_2 and v_3 are compromised for HRP and Prophet, respectively. During the attack scenario of these experiments, the ICT between v_2 and v_3 is claimed to be 0.5 of the ICT between v_2 and v_3 during the normal scenario. For example, if in the normal scenario the average ICT between v_2 and v_3 during the warm-up period is 60 seconds, it is claimed to be 30 seconds during the attack scenario. As shown in Figure 6.1c, HRP replicates more packets towards v_2 and v_3 during the attack scenario. The total number of hijacked packets by v_2 and v_3 is decreased while increasing the on-proportion because HRP decreases the replication factor when the network becomes more connected [26] [96]. For Prophet, the total number of routed packets via v_2 and v_3 increases for the attack scenario compared to the normal scenario (Figure 6.1d).

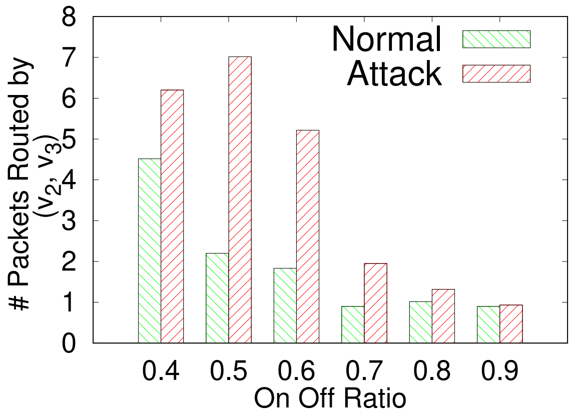
We repeated the same experiments above when v_2, v_3, and v_6 are compromised. During the attack scenario, the ICTs among (v_2-v_3) and (v_3-v_6) are claimed to be 0.5 of the corresponding ICTs during the normal scenario. As shown in Figures 6.2a and 6.2b, the total number of hijacked packets by v_2, v_3, v_6 are higher compared to the experiments when only v_2 and v_3 are compromised for HRP and Prophet. We also repeated the experiments when v_2, v_3, v_6, and v_9 are compromised (they fake the ICTs of (v_2-v_3), (v_3-v_6), and (v_3-v_9) to be 0.5 of the corresponding ICTs during the normal scenario). Similar to the above conclusion, when Eve compromises more nodes, she can hijack more packets. The total number of hijacked packets in Figures 6.2c and 6.2d increases compared to the total number of hijacked packets in Figures 6.2a and 6.2b, respectively.

We also examined the impact of CollusiveHijack through extensive simulations in the ONE simulator [61]. We conducted trace-driven simulations using real world mobility trace, Reality [138], that consists of 97 users from MIT students, faculty and staff members. The mobility

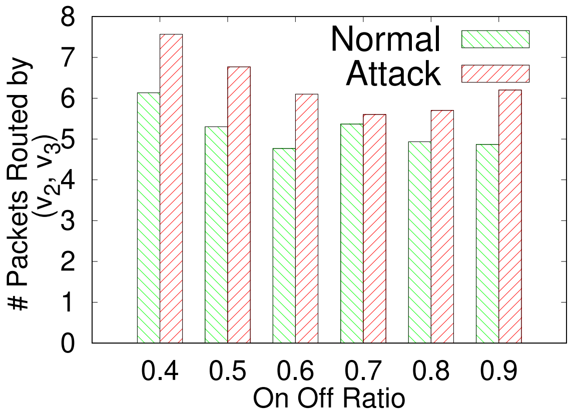
(a)



(b)



(c)



(d)

Figure 6.1: (a) Attack implementation testbed. (b) Network topology. Routed packets for HRP (c) and Prophet (d). *Reprinted with permission from [53].*

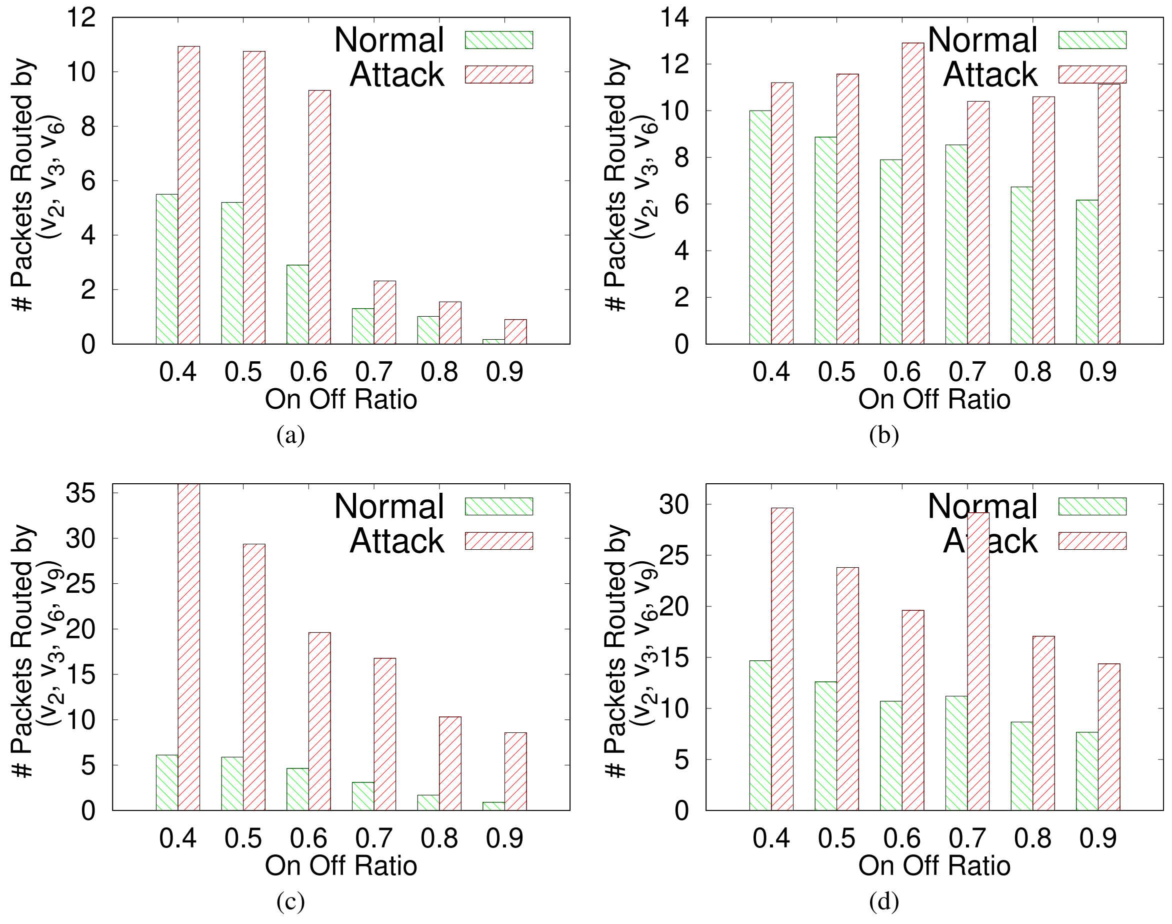


Figure 6.2: Routed packets for HRP (a, c) and Prophet (b, d). *Reprinted with permission from [53].*

trace contains Bluetooth connection events over a time period of 9 months. We set the duration of each experiment to 5 weeks including a warm-up period for 1 week. The 97 nodes run Prophet protocol [28] with $\beta = 0.9$ and $\gamma = 0.999$. After the warm-up period (during the second week), we randomly generated 100 messages, each with size = 1KB and deadline = 10 days, between the nodes with id $\in$ [1, 50]. For each experiment, we considered two scenarios, a normal scenario and an attack scenario. During the normal scenario, the nodes do not fake their ICTs. However, during the attack scenario, we randomly choose nodes from the set of nodes with id $\in$ $[51, 97]$ and fake their ICTs during the warm-up period. Then, we compared the total number of routed packets during the normal and attack scenarios while varying the number of compromised nodes from 2 to 4 and varying the *lying factor* from 2 to 8. The lying factor reflects the ratio that the compromised nodes fake their ICTs by (ICT_{faked} = $\frac{ICT_{original}}{lying\ factor}$). We repeated each experiment 100 times and we averaged these 100 runs per each experiment. Figure 6.3(a) shows the total number of hijacked packets for two compromised nodes, which increases when the compromised nodes increase their lying factor. Also, increasing the number of the compromised nodes increases the total number of hijacked packets, as shown in Figures 6.3(b) and 6.3(c) for three and four compromised nodes, respectively.

6.2 CollusiveHijack Detection

In this section, we present the KS2ST and the design of PDT and HDT.

6.2.1 The Kolmogorov-Smirnov two-sample test (KS2ST)

We represent the WDON by an undirected graph $G = (V, E)$, where V is a set of nodes and E is a set of links, as shown in Figure 6.4. The link between any two nodes, v_i and v_j in V, is denoted by $e_{i,j}$. If $1/\lambda_{i,j}$ is the ICT between v_i and v_j, then, the link weight of $e_{i,j}$ is defined as the contact frequency (i.e., $\lambda_{i,j}$), as shown in Figure 6.4. If v_i never meets v_j, then $e_{i,j}$ will not be in E. Accordingly, any two connected nodes in G should have at least one contact. Message forwarding from v_i to v_j can be accomplished during the contact event. If $D_{i,j}$, $ICT_{i,j}$ represent the link delay and the ICT between two uncorrelated nodes v_i and v_j in V, respectively, then $P[D_{i,j} \leq d]$ =

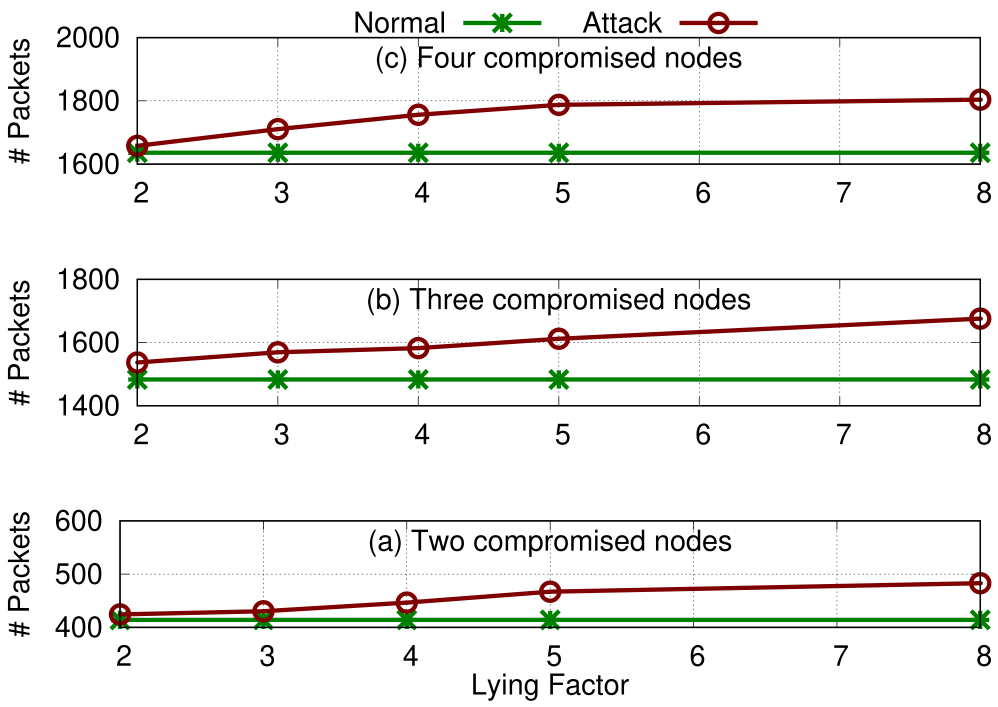


Figure 6.3: Routed packets for Prophet. *Reprinted with permission from [53].*

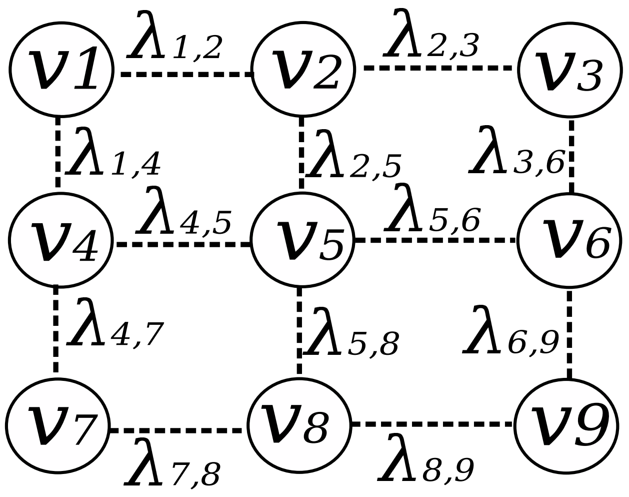


Figure 6.4: WDON contact graph. *Reprinted with permission from [53].*

$\frac{1}{\mathbb{E}[ICT_{i,j}]} \int_0^d (1 - P[ICT_{i,j} \leq z])dz$ and $\mathbb{E}[D_{i,j}]$ = $\frac{\mathbb{E}[ICT_{i,j}]}{2} + \frac{\sigma^2(ICT_{i,j})}{2\mathbb{E}[ICT_{i,j}]}$ [139]. Where $\sigma^2(ICT_{i,j})$ is the variance of $ICT_{i,j}$. Previous research [26] [139] [140] show that the probability density function (pdf) of the $ICT_{i,j}$ between v_i and v_j follows the exponential distribution (i.e., $\lambda_{i,j}e^{-\lambda_{i,j}t}$). We also validated that, using the Reality trace [138], and found that the pdfs of all ICTs among the users in this trace follow the exponential distribution. Accordingly, $\mathbb{E}[D_{i,j}]$ = $\mathbb{E}[ICT_{i,j}]$. Let P_{v_i,v_j} = $\{v_i, v_1, v_2, .., v_{\eta-1}, v_j\}$ represents a path between v_i and v_j in G, where v_x is the x_{th} relay node and η is the number of hops. Then, the path delay consists of links delays, which are independently and exponentially distributed. The pdf of P_{v_i,v_j} delay = $\lambda_{i,1}e^{-\lambda_{i,1}t}$ + $\lambda_{1,2}e^{-\lambda_{1,2}t}$ + .. + $\lambda_{\eta-1,j}e^{-\lambda_{\eta-1,j}t}$ is hypo-exponentially distributed $\sim Hypo(\lambda_{i,1}, \lambda_{1,2}, .., \lambda_{\eta-1,j})$ with mean = $1/[\lambda_{i,1} + \lambda_{1,2} + .. + \lambda_{\eta-1,j}]$ and variance = $1/[\lambda_{i,1}^2 + \lambda_{1,2}^2 + .. + \lambda_{\eta-1,j}^2]$ [141].

The KS2ST is a well known non-parametric goodness-of-fit test [54]. This test measures the distance between the empirical cumulative distribution functions (CDFs) of two samples **a** $= \{a\}_{i=1}^n$ and **b** $= \{b\}_{i=1}^m$ to determine whether they have been drawn from the same distribution or not. The KS test, $KS.test(a, b)$, for $\{a\}_{i=1}^n$ and $\{b\}_{i=1}^m$ samples is defined as $D_{a,b} = \underset{x \in a \cup b}{sup} |F_a(x) - F_b(x)|$, where sup is the supremum function and F_a, F_b are the empirical CDFs. $F_a(x) = \frac{1}{n}\sum_{i=1}^n I_{\{a_i \leq x\}}$ and $F_b(x) = \frac{1}{m}\sum_{i=1}^m I_{\{b_i \leq x\}}$, where $I_{\{a_i \leq x\}}$ is the indicator function that has the value 1 if $a_i \leq x$, and 0 otherwise (same for $I_{\{b_i \leq x\}}$). The null hypothesis of the two sample KS test (i.e., the samples are drawn from the same distribution) is rejected at significance level of $\alpha \in (0, 1]$ if $D_{a,b} > c(\alpha) \times \sqrt{\frac{n+m}{nm}}$, where $c(\alpha) = \sqrt{-\frac{1}{2}\ln(\frac{\alpha}{2})}$ and n, m are the sizes of **a** and **b** samples, respectively. In our design we use α = 0.05. Accordingly, we reject the null hypothesis if the $p-$ $value$ [142] of the KS2ST is < 0.05.

6.2.2 Path Detection Technique (PDT)

Before presenting the details of PDT, we present the steps at each node when it routes a packet, as shown in Algorithm 9. *Note: the steps shown in oval-boxes are for HDT that we present in Section 6.2.3.* When a packet is available at the sender's buffer, the sender inserts the packet's creation time (PCT) and $Sender\_Id$ into the packet header. Moreover, the sender signs the inserted PCT (i.e., using the $DestPrivK$) and $Sender\_Id$ (i.e., using both $DestPrivK$ and $NextHopPrivK$), as shown in lines 2-3 of Algorithm 9. Then, the sender waits until the next hop is available to forward the packet (lines 4 and 7 of Algorithm 9). In case a node receives a packet to be routed, it verifies the signed Id of the previous hop using its private key, as shown in line 10 of Algorithm 9. When the next hop is available for the carrier node, it inserts $Carrier\_Id$ into the packet header. The carrier node also signs the inserted $Carrier\_Id$ (i.e., using $NextHopPrivK$) before forwarding the packet (lines 12 and 15 of Algorithm 9). The aforementioned steps can be accomplished by including the nodes' payload integrity blocks in the packets (pages 24-25 of BSP [55]).

Algorithm 9 Steps at each node for PDT and HDT

```
 1: if Sender then
 2:     Packet_Creation_Time ($PCT$) = current_time()
 3:     Sign$_{DestPrivK}$($PCT$, $Sender\_Id$) || Sign$_{NextHopPrivK}$($Sender\_Id$) $\rightarrow$ Packet_Header
 4:     Wait until Next hop is available
 5:         $\boxed{\text{Packet\_Receiving\_Time } (PRT) = \text{current\_time()}}$
 6:         $\boxed{\text{Sign}_{NextHopPrivK}(PRT) \rightarrow \text{Packet\_Header}}$
 7:         Forward the Packet
 8: else if Carrier then
 9:     if Packet received then
10:         Verify$_{PrevHop_{PrivK}}$($PrevHop\_Id$, $\boxed{PRT}$)
11:     Wait until Next hop is available
12:         Sign$_{NextHopPrivK}$($Carrier\_Id$) $\rightarrow$ Packet_Header
13:         $\boxed{\text{Packet\_Receiving\_Time } (PRT) = \text{current\_time()}}$
14:         $\boxed{\text{Sign}_{NextHopPrivK}(PRT) \rightarrow \text{Packet\_Header}}$
15:         Forward the Packet
```

PDT is a detection-based protocol that runs by the destination nodes in WDONs. The pseudo code of PDT is presented in Algorithm 10. Once the destination receives a packet, it calculates the packet's delay and finds its path. Then, the destination verifies the inserted Ids into the packet header, using the sender and carriers' private keys (lines 2-6 of Algorithm 10). For all received packets via the same path, e.g., $P_{v_i,v_j} = \{v_i, v_1, v_2, .., v_{\eta-1}, v_j\}$, the destination, v_j, stores the packets' delays (line 7 of Algorithm 10) and finds the contact frequencies of P_{v_i,v_j} (i.e., $\lambda_{i,1}, \lambda_{1,2}, .., \lambda_{\eta-1,j}$). Notice that these contact frequencies are used by HRP and Prophet to make replication decisions. HRP

Algorithm 10 PDT at the destination

```
1: for each received $Packet$ via a $Path$ with $\eta$ hops do
2:     Packet_delay ($Pd$) = current_time() - $PCT$
3:     $Path$ = get_path(Packet_Header) = $\{v_i, v_1, v_2, .., v_{\eta-1}, v_j\}$
4:     for the $Sender$ and each $Carrier$ in $Path$ do
5:         Verify$_{Sender_{PrivK}}$($Sender\_Id$)
6:         Verify$_{Carrier_{PrivK}}$($Carrier\_Id$)
7:     Save $Pd$ for $Path$
8: for k received $Packets$ via a $Path$ with $\eta$ hops do
9:     $< Pd_1, Pd_2, .., Pd_k >$ = get_packet_delays($Path$)
10:    $< \lambda_1, \lambda_2, .., \lambda_\eta >$ = get_$\lambda$'s($Path$)
11:    $reject_{null\_hypoehsis}$ = 0
12:    for $j$ = 1 to $num_{tests} = 10000$ do
13:        $< \hat{Pd}_1, \hat{Pd}_2, .., \hat{Pd}_{10k} >$ = $get_{rand}(Hypo(\lambda_1, \lambda_2, .., \lambda_\eta))$
14:        $p\text{-}value = KS.test(< Pd_1, Pd_2, .., Pd_k >, < \hat{Pd}_1, \hat{Pd}_2, .., \hat{Pd}_{10k} >)$
15:        if $p\text{-}value < 0.05$ then
16:            $reject_{null\_hypoehsis} = reject_{null\_hypoehsis} + 1$
17:    if $(reject_{null\_hypoehsis}/num_{tests}) > 0.05$ then
18:        $Path$ is compromised
19:    else
20:        $Path$ is honest
```

and Prophet require these contact frequencies to be announced to all nodes in the WDON. Then, v_j employs the KS2ST to determine whether the delays of the received packets match the summation of the announced links' delays of P_{v_i,v_j} (i.e., the path delay of $P_{v_i,v_j} \sim Hypo(\lambda_{i,1}, \lambda_{1,2}, .., \lambda_{\eta-1,j})$. In the following paragraph, we present the steps of PDT through an example.

As an example, we assume that v_1 in Figure 6.4 sent k packets with delays = $< Pd_1$, Pd_2, .., $Pd_k >$ to v_9 via P_{v_1,v_9} = $\{v_1, v_2, v_3, v_6, v_9\}$ path that has the following contact frequencies: $< \lambda_{1,2}, \lambda_{2,3}, \lambda_{3,6}, \lambda_{6,9} >$. Once v_9 collects the aforementioned packets' delays and contact frequencies (lines 9-10 in Algorithm 10), it repeats the following process 10,000 times. v_9 draws a random sample, with size = 10×k, $< \hat{Pd}_1$, $\hat{Pd}_2$, .., $\hat{Pd}_{10k} >$ from the hypo-exponential distribution, $Hypo(\lambda_{1,2}$, $\lambda_{2,3}$, $\lambda_{3,6}$, $\lambda_{6,9})$, and performs a KS2ST between the packets delays and the drawn sample (lines 13-14 of Algorithm 10). If the resulted $p\text{-}value$ of the KS2ST is less than α = 0.05, v_9 increases the $reject_{null\_hypoehsis}$ counter, which keeps track of the number of times the null hypothesis is rejected. After repeating the KS2ST 10,000 times (with different random samples), if the ratio of the number of times that the null hypothesis is rejected is $> 5\%$, then v_9 labels P_{v_1,v_9} as a compromised path. Otherwise, P_{v_1,v_9} is labeled as an honest path (lines 17-20 of Algorithm 10).

6.2.3 Hop Detection Technique (HDT)

HDT is based on the idea that collecting the packets' receiving times (PRTs) at the intermediate nodes enhances the destination node detection capability against the CollusiveHijack attack. That is, instead of labeling the whole path as compromised, as the case of PDT, HDT aims to pinpoint the lying hops and accordingly the compromised nodes in the WDON. However, HDT requires additional steps to be performed by the intermediate nodes apart from the steps accomplished by BSP [55], as shown in oval-boxes of Algorithm 9. When the next hop is available for the sender or the intermediate nodes, they have to insert the PRT into the packet header and sign the inserted PRT (i.e., using $NextHopPrivK$) before delivering the packet to the next hop, as shown in lines 5-6 and 13-14 of Algorithm 9. Moreover, when an intermediate node, e.g., v_i, receives a packet, it verifies whether the PRT (that has been signed by the previous hop) matches its time. Also, v_i verifies the signature of the previous hop using the $PrevHop_{PrivK}$ (line 10 of Algorithm 9). Notice that if we ignore the links' propagation and transmission delays (relatively small compared to ICTs in WDONs), then the signed time by the previous hop should equal the PRT at v_i.

The pseudo code of HDT is shown in Algorithm 11. To illustrate how HDT works, we present its steps using the same example we used in Section 6.2.2 (node v_1, in Figure 6.4, sent k packets to node v_9 via $P_{v_1,v_9} = \{v_1, v_2, v_3, v_6, v_9\}$). Once v_9 receives a packet from v_6, it verifies the PRT and the Id of v_6 using v_6's private key, as shown in line 2 of Algorithm 11. Afterwards, v_9 gets the path of the received packet and verifies the inserted Ids of v_1, v_2, v_3, and v_6 using their private keys, respectively, as shown in lines 3-6 of Algorithm 11.

Once v_9 collects the packets' delays for all intermediate hops, it builds the packet receiving times matrix, PRT_{nodes}, as shown in line 8 of Algorithm 11 . PRT_{nodes} is a $[k \times (\eta + 1)]$ matrix that contains the creation and receiving times for the k packets that have been routed via a path with η hops. The $(k \times 5)$ PRT_{nodes} that is built by v_9 is shown in Figure 6.5. Notice that $PCT_{x,y}$ represents the creation time of the x_{th} packet at node y and $PRT_{x,y}$ represents the receiving time of

Algorithm 11 HDT at the destination

```
1: for each received $Packet$ via a $Path$ with $\eta$ hops do
2:     Verify$_{PrevHop_{PrivK}}(PrevHop\_Id,\ PRT)$
3:     $Path$ = get_path(Packet_Header) = $\{v_i, v_1, v_2, .., v_{\eta-1}, v_j\}$
4:     for the $Sender$ and each $Carrier$ in $Path$ do
5:         Verify$_{Sender_{PrivK}}(Sender\_Id,\ PCT_{Sender\_Id})$
6:         Verify$_{Carrier_{PrivK}}(Carrier\_Id,\ PRT_{Carrier\_Id})$
7: for k received $Packets$ via a $Path$ with $\eta$ hops do
8:     $PRT_{nodes} \leftarrow \underbrace{\left.\begin{array}{|c|c|c|c|}\hline PCT_{1,v_i} & PRT_{1,v_1} & .. & PRT_{1,v_j} \\ \hline PCT_{2,v_i} & PRT_{2,v_1} & .. & PRT_{1,v_j} \\ \hline .. & .. & .. & .. \\ \hline PCT_{k,v_i} & PRT_{k,v_1} & .. & PRT_{1,v_j} \\ \hline \end{array}\right\}k}_{\eta+1}$
9:     for each $col_i$ in $PRT_{nodes}$ do
10:        $D_{hops}[col_i]$ = $PRT_{nodes}[col_{i+1}]$ - $PRT_{nodes}[col_i]$
11:    $D_{hops} = \underbrace{\left.\begin{array}{|c|c|c|c|}\hline D_{1,(v_i,v_1)} & D_{1,(v_1,v_2)} & .. & D_{1,(v_{\eta-1},v_j)} \\ \hline D_{2,(v_i,v_1)} & D_{2,(v_1,v_2)} & .. & D_{2,(v_{\eta-1},v_j)} \\ \hline .. & .. & .. & .. \\ \hline D_{k,(v_i,v_1)} & D_{k,(v_1,v_2)} & .. & D_{k,(v_{\eta-1},v_j)} \\ \hline \end{array}\right\}k}_{\eta}$
12:    for each $col_i$ in $D_{hops}$ do
13:        $< D_1, D_2, .., D_k >$ = transpose($col_i$)
14:        $< \lambda_i >$ = get_$\lambda(Link_i)$
15:        $reject_{null\_hypoehsis}$ = 0
16:        for $j$ = 1 to $num_{tests} = 10000$ do
17:            $< \hat{D_1}, \hat{D_2}, .., \hat{D_{10k}} >$ = $get_{rand}(Hypo(\lambda_i))$
18:            $p\text{-}value = KS.test(< D_1, D_2, .., D_k >, < \hat{D_1}, \hat{D_2}, .., \hat{D_{10k}} >)$
19:            if $p\text{-}value < 0.05$ then
20:                $reject_{null\_hypoehsis} = reject_{null\_hypoehsis} + 1$
21:        if $(reject_{null\_hypoehsis}/num_{tests}) > 0.05$ then
22:            Source Node in Hop $i$ is compromised
23:        else
24:            Hop $i$ is not compromised
```

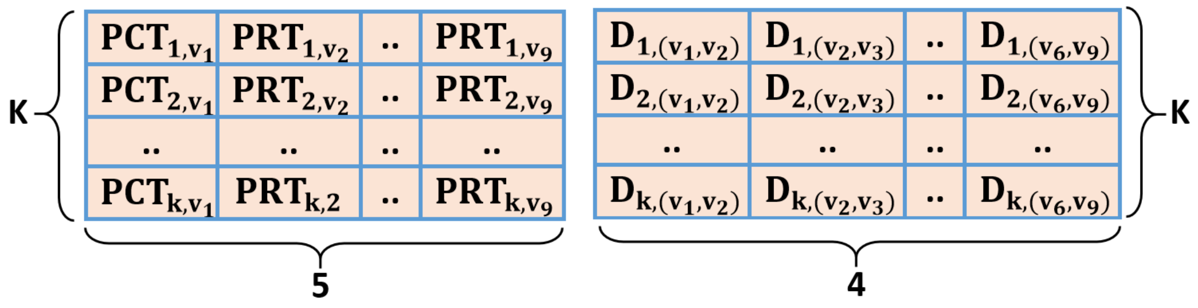


Figure 6.5: PRT_{nodes} and D_{hops} built by v_9. *Reprinted with permission from [53].*

the x_{th} packet at node y. By decrementing each i_{th} column from the $i_{th} + 1$ column in PRT_{nodes}, v_9 builds the D_{hops}, an $(k \times \eta)$ = $(k \times 4)$ matrix that contains the hops' delays of P_{v_1,v_9} (lines 9-11 of Algorithm 11). Notice that $D_{x,(y,w)}$ represents the delay of the x_{th} packet at $\textcircled{y} \longleftrightarrow \textcircled{w}$ hop. The $(k \times 4)$ D_{hops} that is built by v_9 is shown in Figure 6.5. Then, v_9 performs the same detection steps as the PDT algorithm by leveraging the KS2ST. However, since the hops' delays are calculated by v_9 (i.e., the columns of D_{hops}), HDT performs a hop-wise detection for P_{v_1,v_9}. Consequently, v_9 can label each of the intermediate hops, $[(v_1, v_2), (v_2, v_3), (v_3, v_6), (v_6, v_9)]$, as a compromised or honest hop (lines 12-24 of Algorithm 11).

6.3 Evaluation

6.3.1 Implementation and Experimental Setup

We implemented PDT and HDT using R statistical language [143] on Asus Eee notebooks, that are shown in Figure 6.1a. The notebooks run Ubuntu 14.04 LTS and have Intel(R) Atom(TM) CPU operating at 1.6 GHz and 1GB RAM. Our R code is executed at the destination nodes by the C++ user-space daemon services of HRP and Prophet protocols [96] [97] using Rcpp package [144]. We also enabled BSP [55] for HRP and Prophet daemon services. For the cryptographic sign and verify operations, we leveraged the Pairing-Based Cryptography (PBC) library [145] that implements the Hess identity-based signatures [146]. The PBC implementation uses a 160-bit elliptic curve group with 512-bit keys. Notice that we decided to use the PBC library in our experiments since it is executable in our notebooks (i.e., our SekGens algorithm was only implemented for Android).

Apart from the time that is needed for the sign/verify operations in PBC, which is expected to be higher than the time that is needed in case we use the symmetric keys of the Diffie-Hellman algorithm, we expect that the results of our experiments below should be similar. In the following paragraphs, we describe our experiments.

1. **Experiments in the ONE simulator:** we extracted the ICTs and the number of hops from the experiments that we conducted in Section 6.1 in the ONE simulator [61]. We aim to investigate the performance of PDT and HDT for various scenarios. Hence, we performed different experiments by varying the ratio of the compromised links in different paths (with 4, 5, and 6 hops). The ratio of the compromised links in our experiments are: (1) $16.6\%(\frac{1}{6})$. (2) $20\%(\frac{1}{5})$. (3) $25\%(\frac{1}{4})$. (4) $33.3\%(\frac{2}{6})$. (5) $40\%(\frac{2}{5})$. (6) $50\%(\frac{3}{6})$. (7) $60\%(\frac{3}{5})$. We also varied the *lying factor* (illustrated in Section 6.1) of the compromised links (from 2 to 8) and the number of the received packets, k, at the destination from 20 to 100. We also repeated the same experiments that we described in Section 6.1 and the steps mentioned above for another real world mobility trace, UCSD [147]. UCSD trace consists of 275 PDA users from University of California - San Diego students. The mobility trace contains Wi-Fi connection events over a time period of 11 weeks.

2. **Experiments on Asus notebooks testbed:** we repeated two experiments against HRP on the testbed shown in Figure 6.1a for 0.4 and 0.9 on-off ratios. We generated 100 data packets (each with a deadline = 300 seconds) from v_1 to v_{11} and we varied the number of compromised nodes from 2 to 4. The lying factor for these experiments is 2. We leveraged the HRP implementation [96] to collect the ICTs among the nodes. The HRP implementation leverages Optimized Link State Routing (OLSR) [148] for topology maintenance by invoking *olsrd* [149], an OLSR implementation, and fetches topology information from it. The *hrptclient* [96] measures the packet's delay at the destination. We evaluated our PDT and HDT in two aspects:

1. **Quantifying the overhead** of PDT and HDT using the following metrics: **a) Packet Size Overhead:** $\frac{additonal\ data\ size}{packet\ size}$, which measures the overhead of adding the node's Id, PCT, and PRT into the packets; **b) Execution Time Overhead** of the sign and verify operations.

2. **Evaluating the performance** of PDT and HDT using the following metrics: **a) Detection Rate:**

the ability of PDT and HDT to detect the attempts of the CollusiveHijack attacker (true positive rate); **b) False Positive Rate:** the rate of claiming a legitimate path/link as a compromised one; **c) Detection Latency (days/hours):** the required time to collect 20, 40, 60, 80, and 100 packets at the receiver for both PDT and HDT.

6.3.2 Security Analysis

6.3.2.1 Fake Ids attack defense

Eve might pretend that two or more of her nodes have the same Id to not be detected by PDT. Without loss of generality, if v_2 and v_3 in $P_{s,d} = \textcircled{s} \xleftrightarrow{\lambda_{s,v_1}} \textcircled{v_1} \xleftrightarrow{\lambda_{v_1,v_2}} \textcircled{v_2} \xleftrightarrow{\lambda_{v_2,v_3}} \textcircled{v_3} \xleftrightarrow{\lambda_{v_3,v_4}} \textcircled{v_4} \xleftrightarrow{\lambda_{v_4,d}} \textcircled{d}$ are compromised, then v_3 pretends to be v_2 (i.e., Eve aims to hide v_3 identity). In this case, the packets' delays at d follow $Hypo(\lambda_{s,v_1}, \lambda_{v_1,v_2}, \lambda_{v_2,v_3}, \lambda_{v_3,v_4}, \lambda_{v_4,d})$. However, when d runs PDT, the random samples (line 13 of Algorithm 10) are drawn from $Hypo(\lambda_{s,v_1}, \lambda_{v_1,v_2}, \lambda_{v_2,v_4}, \lambda_{v_4,d})$. In order to check whether PDT is able to detect fake Ids attacks, we repeated the first and second steps of the experiments in the ONE simulator (in Section 6.3.3.2) for η = 5, 6, and 7 hops. For the third step, we randomly picked two adjacent nodes and launched a fake Ids attack by pretending they have the same Id. Then, we averaged the number of times that the destination, which runs PDT, was able to detect the fake Ids attack for different number of received packets, k. The PDT detection rate is $>$ 98.0% when $k >$ 60, as shown in Figure 6.8b. Hence, if Eve launches the fake Ids attack, one of her nodes will be detected (v_2 in $P_{s,d}$ above). We also calculated the detection rates against fake Ids attack for 3 compromised nodes (all have same Id) and found that it is $>$ 99.0% when $k >$ 40.

6.3.2.2 Fake PRTs attack defense

Eve's nodes might fake their PRTs to not be detected by HDT. For example, in $P_{s,d} = \textcircled{s} \xleftrightarrow[\tau_1]{\lambda_{s,v_1}} \textcircled{v_1} \xleftrightarrow[\tau_2]{\lambda_{v_1,v_2}} \textcircled{v_2} \xleftrightarrow[\tau_3]{\lambda_{v_2,v_3}} \textcircled{v_3} \xleftrightarrow[\tau_4]{\lambda_{v_3,v_4}} \textcircled{v_4} \xleftrightarrow[\tau_5]{\lambda_{v_4,d}} \textcircled{d}$, where τ's are the links delays, if s sends a packet at τ_0, the PCT and PRTs for this packet = $[\tau_0, \sum_{i=0}^{1} \tau_i, \sum_{i=0}^{2} \tau_i, \sum_{i=0}^{3} \tau_i, \sum_{i=0}^{4} \tau_i, \sum_{i=0}^{5} \tau_i]$. These PCT and PRTs are respectively signed by $[(s), (s, v_1), (v_1, v_2), (v_2, v_3), (v_3, v_4), (v_4, d)]$.

Without loss of generality, if Eve compromises v_2 and v_3 and fakes their contact frequencies to

$2 \times \lambda_{v_2,v_3}$, she also can fake the PRT at v_3 to $\frac{\tau_3}{2}$ to not be detected by HDT on the $\textcircled{$v_2$}\xleftrightarrow[\frac{\tau_3}{2}]{2\times\lambda_{v_2,v_3}}\textcircled{v_3}$ hop. That is, Eve fakes the PRT at v_3 to match her claimed contact frequency, $2 \times \lambda_{v_2,v_3}$. However, in this case, our HDT is able to detect the source node of $\textcircled{$v_3$}\xleftrightarrow[\frac{\tau_3}{2}+\tau_4]{\lambda_{v_3,v_4}}\textcircled{v_4}$, as shown in line 22 of Algorithm 11 (i.e., since the packets' delays at this hop, $\frac{\tau_3}{2} + \tau_4$, do not match its announced contact frequency, λ_{v_3,v_4}). Accordingly, if Eve compromises n adjacent nodes, $v_1, v_2, .., v_n$ in a path, she can launch fake PRTs attack by faking the PRTs among $v_1, v_2, .., v_{n-1}$ nodes, however, the last compromised node, v_n, will be detected by HDT.

6.3.3 Performance Evaluation

In this section, we present the overhead of PDT and HDT and their performance evaluations.

6.3.3.1 Quantifying the Overhead of PDT and HDT

In PDT and HDT, the sender adds its Id and PCT (each is 4B) into the packet's header and each intermediate node adds its Id (4B). Also, in HDT, each intermediate node adds its PRTs (4B). The signature field of the Hess identity-based scheme is 64B. Hence, the PDT and HDT's packet size overhead in bytes for $P_{v_i,v_j} = \{v_i, v_1, v_2, .., v_{\eta-1}, v_j\}$ path are $[68 \times \eta] + 4$ and $72 \times \eta$, respectively. For 10-hop path with an IPv4 packet size (64KB), the packet size overheads for PDT and HDT are $\sim$1.04% and $\sim$1.09%, respectively, which are relatively small. In order to calculate the execution time of the sign and verify operations of the Hess identity-based scheme [146], we averaged 10,000 measurements of these operations on one Asus Eee notebooks. The execution time of the sign and verify operations are 118 msec and 42 msec, respectively.

6.3.3.2 Performance Evaluation of PDT and HDT

Evaluation in the ONE simulator using Reality trace. We conducted the first experiment on a path with η = 6 hops including one compromised hop. We performed the following steps. First, we extracted 1,000 different paths with 6 hops from the trace-driven 100 experiments that we conducted in Section 6.1. Second, we extracted the ICTs among the nodes in these paths and collected the packets' delays at the destination nodes. Third, we randomly picked one hop and varied its lying factor from 2 to 8. Fourth, we counted the number of times the destination, which runs the PDT algorithm, was able to detect the CollusiveHijack attack after receiving 20, 40, 60, 80, and 100 packets. Fifth, we averaged the 1,000 runs for each lying factor and k value.

The PDT's detection rate for the first experiment is shown in Figure 6.6a. The detection rate increases when the two compromised nodes increase their lying factor and when the total number of the received packets at the destination increases. We illustrated in Figure 6.3(a), that it is attractive for the compromised nodes to increase their lying factor to hijack more packets (the total hijacked packets are 10 and 80 for 2 and 8 lying factors, respectively). However, increasing the number of hijacked packets enhances the PDT's detection capability. The PDT's detection rates for the first experiment when the destination receives 100 packets are: 85.4%, 96.5%, 98.8%, 99.5%, 99.7%, 99.7%, 99.8% for 2, 3, 4, 5, 6, 7, 8 lying factors, respectively, as shown in Figure 6.6a.

We repeated the five steps mentioned above for a higher ratio of compromised links in the path. In the second experiment, we had a path with η = 5 hops including one compromised hop. As it is clear in Figure 6.6b, we can draw the same conclusions as from Figure 6.6a. The PDT's detection rate increases when the compromised nodes increase their lying factor and when the destination receives more packets. The aforementioned conclusions can be also observed for the remaining five experiments that are shown in Figures 6.6c, 6.6d, 6.7a, 6.7b, 6.7c. Moreover, if we compare the PDT's detection rate of all figures, we observe that while Eve increases the ratio of the compromised links, the PDT's detection rate increases. Figure 6.8a presents the false positive rate for PDT when η = 4, 5, and 6 hops. The false positive rate for all experiments is $<$ 3.6%, which is relatively small.

Since HDT performs a hop-wise detection against the CollusiveHijack attack, we present its detection rate on 1-hop for different k values in Figure 6.7d. As it is clear in Figure 6.7d, HDT's detection rates are $> 98\%$ for all lying factors when $k > 40$. Hence, we recommend to leverage HDT to effectively detect the CollusiveHijack attack despite it's overhead and incompatibility with BSP [55]. The false positive rates of HDT (i.e., on 1-hop) are relatively small, as shown in Figure 6.8a.

For the detection latency, we calculated the average time needed to receive 20, 40, 60, 80, and

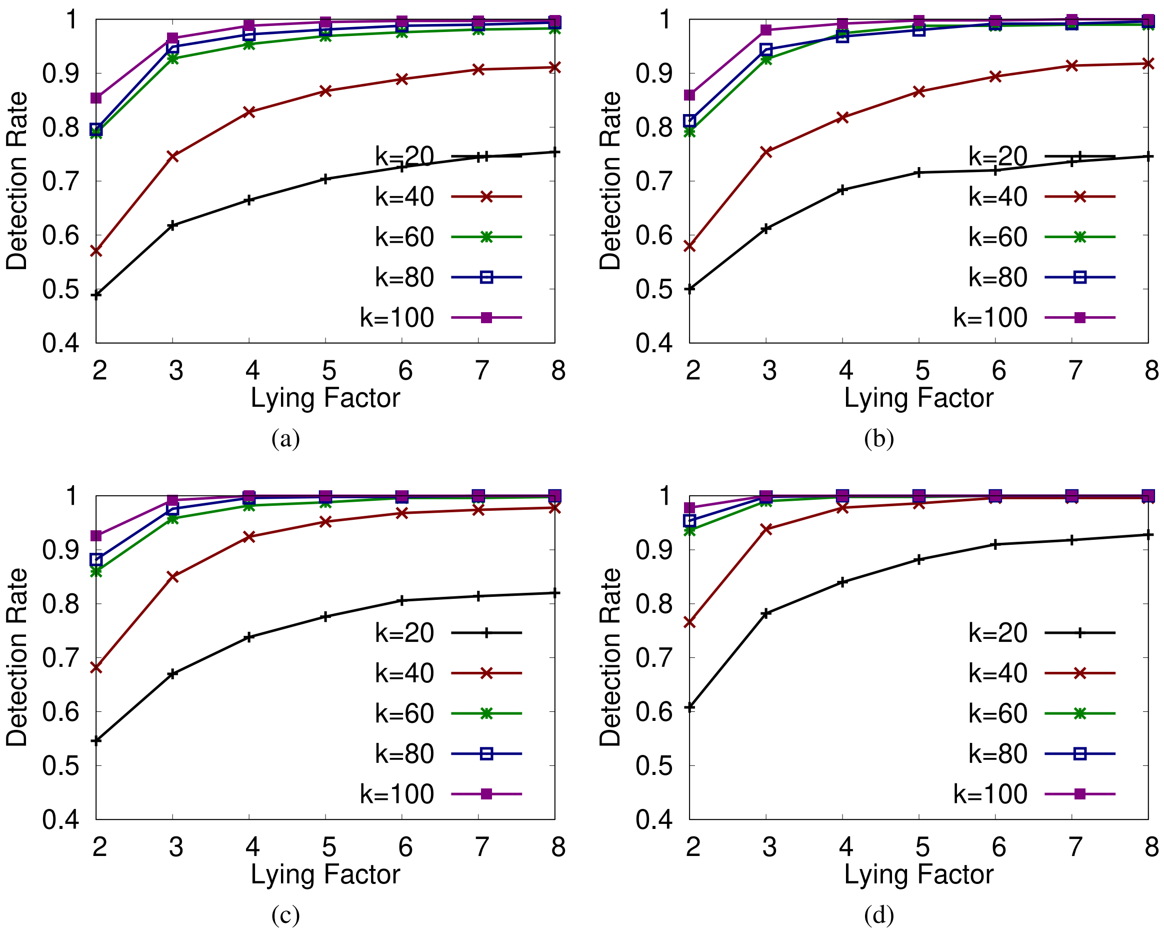


Figure 6.6: PDT's Detection Rate for Reality trace when the ratio of the compromised links: (a) $16.6\%(\frac{1}{6})$. (b) $20\%(\frac{1}{5})$. (c) $25\%(\frac{1}{4})$. (d) $33\%(\frac{2}{6})$. *Reprinted with permission from [53].*

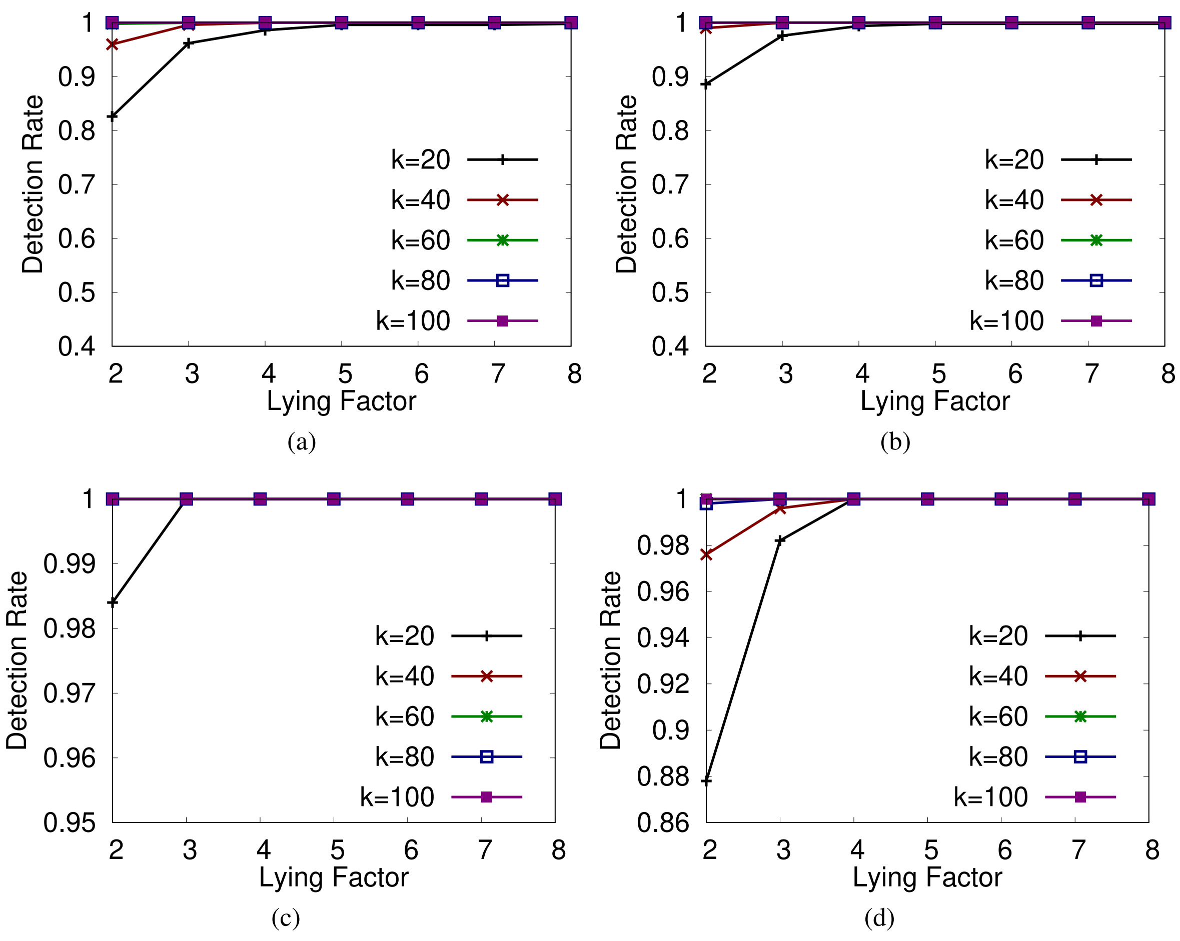


Figure 6.7: PDT's Detection Rate for Reality trace when the ratio of the compromised links: (a) $40\%(\frac{2}{5})$. (b) $50\%(\frac{3}{6})$. (c) $60\%(\frac{3}{5})$. (d) HDT's Detection Rate. *Reprinted with permission from [53].*

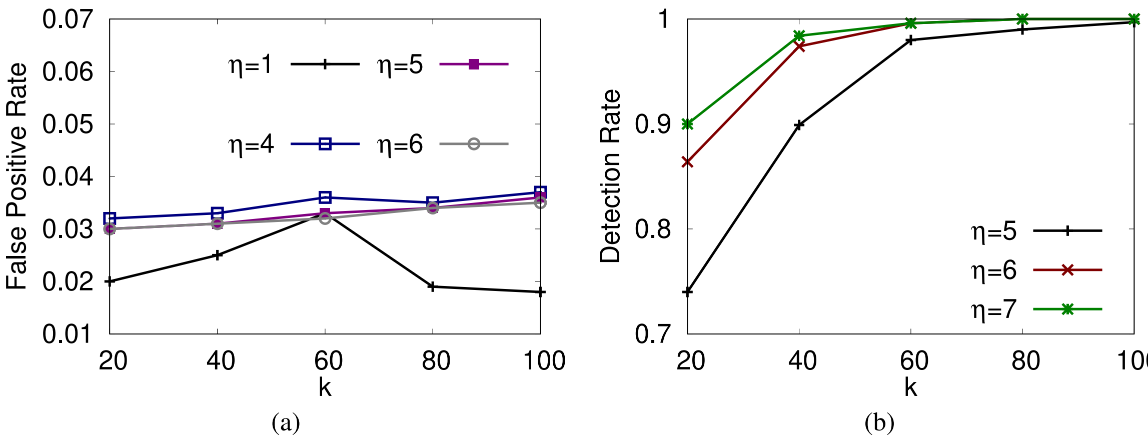
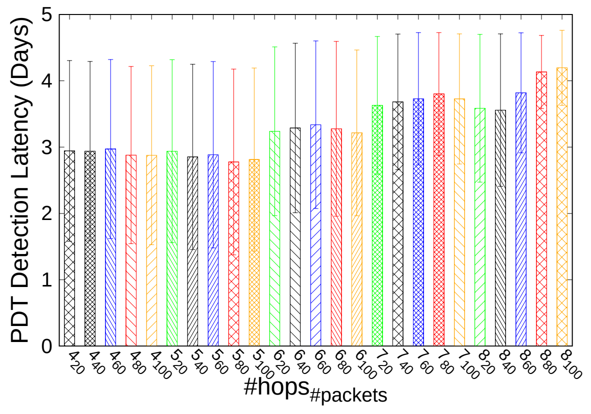


Figure 6.8: (a) False Positive Rate for Reality trace of PDT (η = 4, 5, 6 hops) and HDT (η = 1 hop). (b) PDT Detection Rate Against Fake Ids Attack. *Reprinted with permission from [53].*


(a)

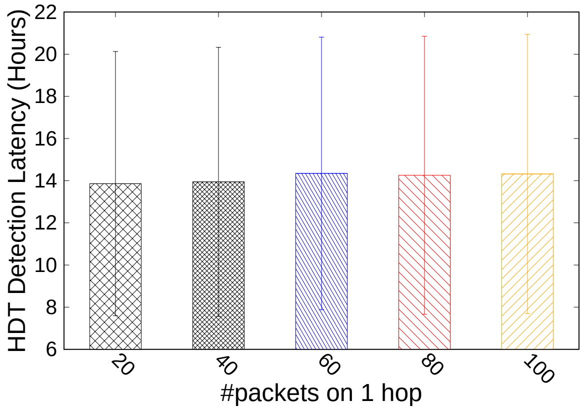
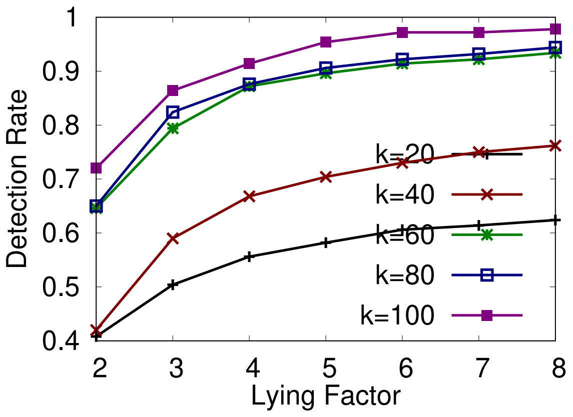

(b)

Figure 6.9: Detection Latency for Reality trace for (a) PDT. (b) HDT.

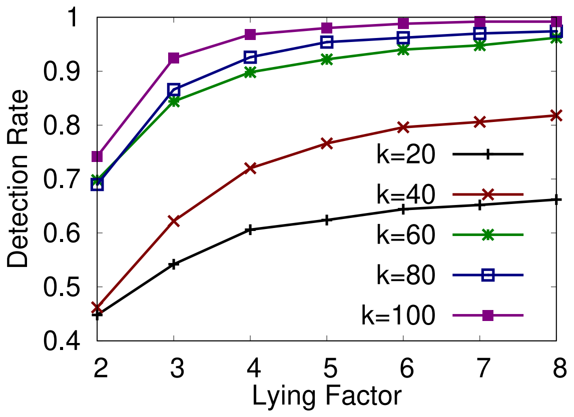
100 packets in PDT and HDT. As presented in Figure 6.9a, the detection latencies for PDT for all packets for 4, 5, 6, 7, and 8 hops are 3 to 4 days, which are relatively short. For HDT, since the destination performs a hop-wise detection, we calculated the detection latencies for different packets on 1-hop. The HDT detection latencies are ∼14 hours (i.e., HDT detection is faster than PDT), as presented in Figure 6.9b.

Evaluation in the ONE simulator using UCSD trace. We calculated the PDT and HDT's detection rates for UCSD trace with the same ratio of compromised hops, lying factors, and received packets at the destination node as we did for the Reality trace above. The detection rates for PDT and HDT are presented in Figures 6.10a, 6.10b, 6.10c, 6.10d, 6.11a, 6.11b, 6.11c, and 6.11d. We obtain the same conclusions as from the experiments with Reality trace above. That is, the PDT's detection rate increases when Eve increases the ratio of the compromised links and/or the lying factors, as well as when the destination receives more packets. Similarly, the HDT's detection rate increases when Eve increases the lying factors and when the destination receives more packets. We also calculated the false positive rates for PDT and HDT using UCSD trace, as presented in Figure 6.12a (i.e., the false positive rates for UCSD trace experiments are relatively small).

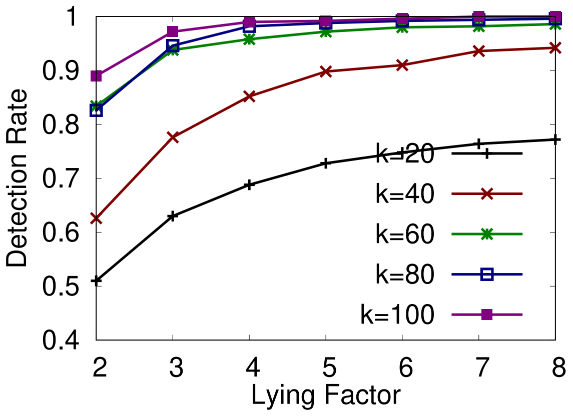
Moreover, we calculated the PDT and HDT's detection latencies for UCSD trace. As shown in Figure 6.13a, the detection latencies for PDT for all packets for 4, 5, 6, 7, and 8 hops are 1.5 to 2.6



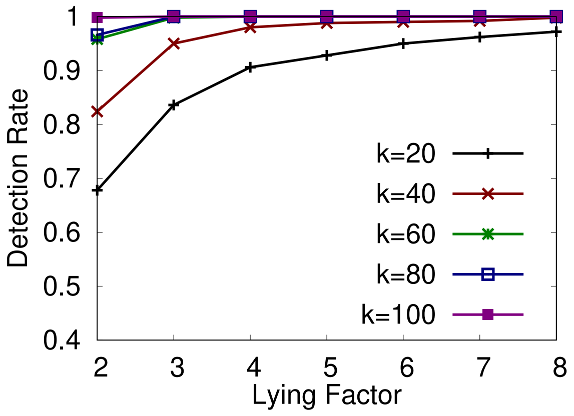
(a)



(b)

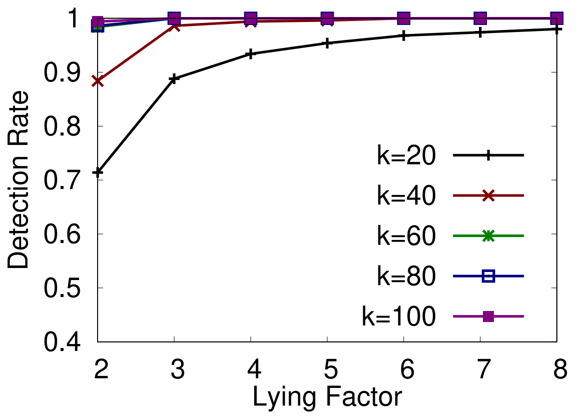


(c)

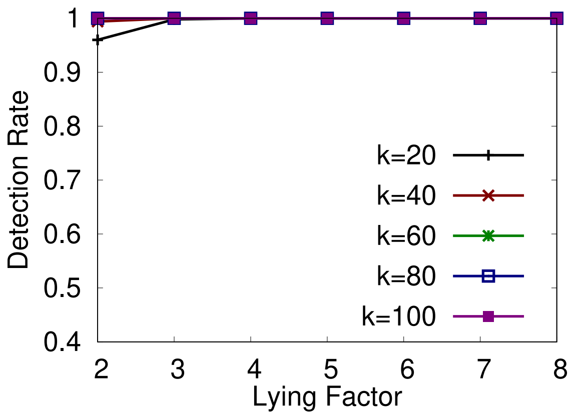


(d)

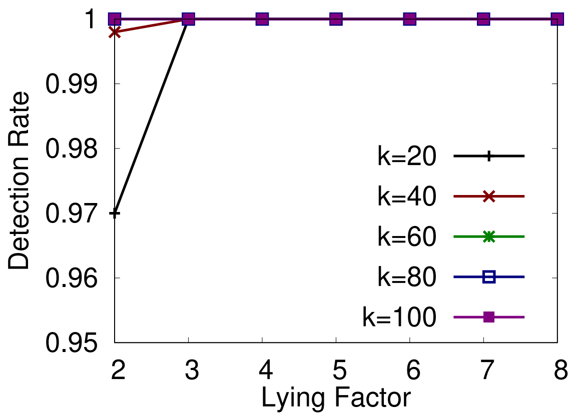
Figure 6.10: PDT's Detection Rate for UCSD trace when the ratio of the compromised links: (a) $16.6\%(\frac{1}{6})$. (b) $20\%(\frac{1}{5})$. (c) $25\%(\frac{1}{4})$. (d) $33\%(\frac{2}{6})$.



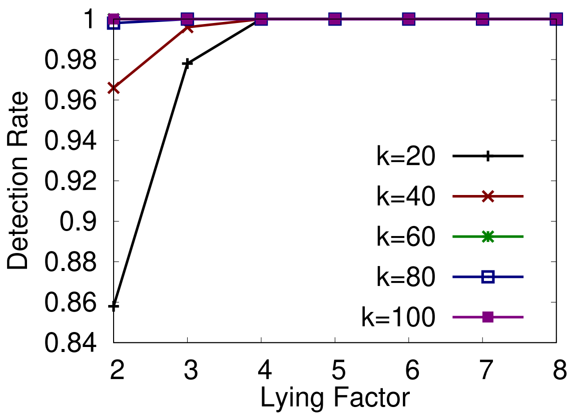
(a)



(b)

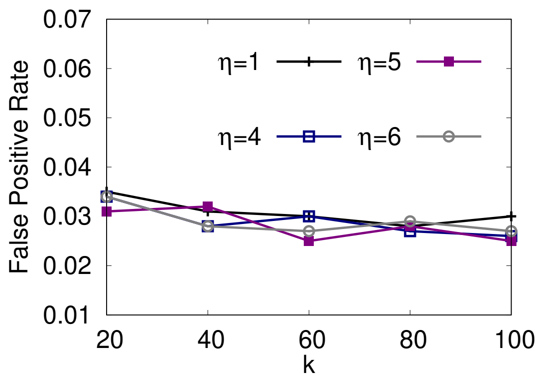


(c)



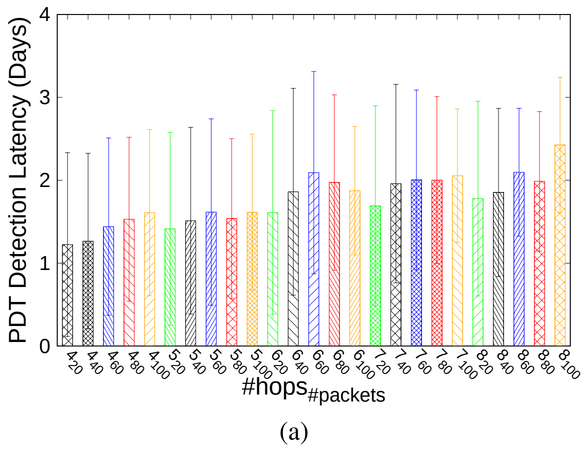
(d)

Figure 6.11: PDT's Detection Rate for UCSD trace when the ratio of the compromised links: (a) $40\%(\frac{2}{5})$. (b) $50\%(\frac{3}{6})$. (c) $60\%(\frac{3}{5})$. (d) HDT's Detection Rate.



(a)

Figure 6.12: (a) False Positive Rate for UCSD trace of PDT (η = 4, 5, 6 hops) and HDT (η = 1 hop).



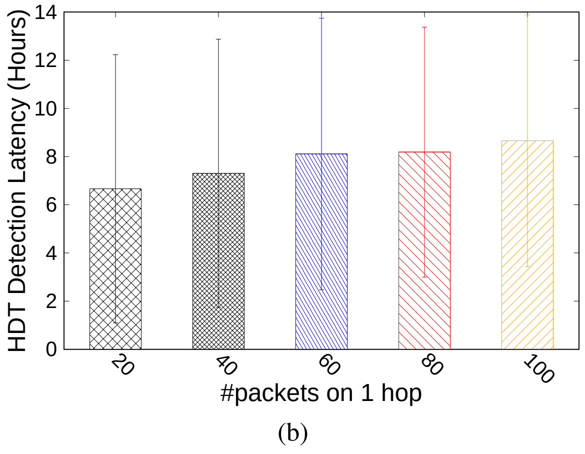
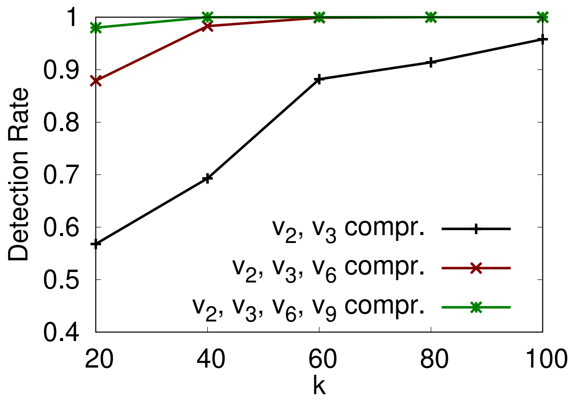


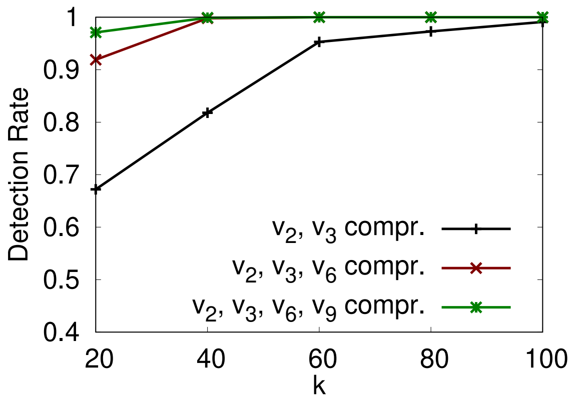
Figure 6.13: Detection Latency for UCSD trace for (a) PDT. (b) HDT.

days, which are relatively short. For HDT, the detection latencies are 6 to 8.5 hours, as shown in Figure 6.13b.

System evaluation on Asus notebooks testbed. Figures 6.14a and 6.14b show the detection rate of PDT for 0.4 and 0.9 on-off ratios. The detection rate increases when the total number of received packets at v_{11} increases and when the number of compromised nodes increases from 2 to 4. When v_{11} receives 60 packets, the PDT detection rate is $>$ 88.0%. We also investigated the HDT detection capability against the CollusiveHijack attack for these experiments and found that its detection rate is $>$ 98.0% when k $>$ 40. The false positive rates of PDT and HDT for these experiments are $<$ 3.5% (close to the rates shown in Figure 6.8a).



(a)



(b)

Figure 6.14: PDT's Detection Rate for on-off ratio = (a) 0.4. (b) 0.9. *Reprinted with permission from [53].*

7. CONCLUSIONS AND FUTURE WORK

In this section, we conclude this dissertation and present the future work.

7.1 Conclusions

With the prevalence and ubiquity of Wi-Fi Direct devices nowadays, Wi-Fi Direct based Opportunistic Networks (WDONs) will play a critical role in future mobile networks. Although the communication performance and power saving of Wi-Fi Direct devices have received some attention from the research community [1] [150] [151], little is known regard the security of this technology. In this dissertation, we present the vulnerabilities of Wi-Fi Direct devices in WDONs to three security attacks. First, we show the vulnerability of the PIN method of the Wi-Fi Protected Setup (WPS) protocol to the brute-force/dictionary attack. Second, we show the vulnerability of the Group Owner devices that run the Push-Button method of the WPS protocol to the EvilDirect attack. Third, we show the vulnerability of the Hybrid Routing Protocol (HRP) and the Prophet protocol in WDONs to the CollusiveHijack attack. We show the severe consequences of these attacks against the devices authentication and data confidentiality. Next, we tackle these vulnerabilities in a bottom-up approach by moving along the Datalink and Network layers of the TCP/IP protocol stack of WDONs. Specifically, we propose a framework that contains a set of a secure-key-establishment algorithm and protocols to address the aforementioned vulnerabilities.

In order to secure the PIN method of the WPS protocol against the brute-force/dictionary attacks, we propose to use contextual information, obtained from on-board sensors of the WDONs' devices. Our solution, SekGens, has three phases. Quantization: in which, the draft key is generated from different sensors data. Reconciliation: during this phase, the two devices eliminate any minor differences in the bits of their draft keys by using the Cascade reconciliation mechanism. Privacy-Amplification-and-Hashing: in which, the two devices omit all bits exposed during the reconciliation phase and apply hashing to the remaining secret bits to generate the final 128 bits key. We demonstrate the feasibility and the robustness of SekGens through a proof-of-concept implementation on Google Nexus 5 and Samsung Galaxy S2 smartphones. We also prove the effectiveness of SekGens by showing that it generates keys with high agreement, at a fast rate, and with high Shannon entropy.

In order to secure the Push-Button method of the WPS protocol against the EvilDirect attacks, we propose to employ the inherent randomness in the wireless channel between the clients and the GO in WDONs. Our protocol, EvilDirectHunter, is an interactive protocol that executes between each client in the P2P group and the GO. Each client records an RSS profile for the legitimate GO, which in turn records a profile for each client in the P2P group. EvilDirectHunter checks whether the RSS profiles of both the client and potential GO devices are similar to each other by exchanging challenge and response packets. Moreover, due to the lack of randomness in the wireless channel when the WDON's devices are static or when there are few mobile intermediate objects in the environment, we propose to execute an additional detection phase for the EvilDirectHunter protocol, which is based on Multi-Dimensional Scaling algorithm and involves a cooperation among all clients. We demonstrate the feasibility of EvilDirectHunter through a proof-of-concept implementation on Google Nexus 5 and Samsung Galaxy S2 smartphones. Moreover, we prove the effectiveness of EvilDirectHunter by showing that it detects EvilDirect attacks in WDONs with a high detection rate while maintaining a low false positive rate.

In order to secure the HRP and Prophet routing protocols against the CollusiveHijack attacks, we propose to employ the Kolmogorov-Smirnov two-sample test (KS2ST) and design two techniques based on that: the Path Detection Technique (PDT) and the Hop Detection Technique (HDT). PDT and HDT offer a trade off between the compatibility with the Bundle Security Protocol in WDONs and the detection capability against the CollusiveHijack attack. In PDT, the destination node performs a path-wise detection by collecting the packets' delays of the path and then employs the KS2ST to test whether the statistical distribution of the packets' delays follows the statistical distribution that is derived from the ICTs of the intermediate nodes. On the other hand, in HDT, the destination node performs a hop-wise detection by requiring additional information from the intermediate nodes in WDONs. That is, the packets' forwarding times. The destination node leverages the KS2ST to detect whether each hop in the path is compromised or not. Hence, HDT can achieve a higher detection rate against the CollusiveHijack attack than PDT. We demonstrate the feasibility of PDT and HDT through a proof-of-concept implementation on Asus Eee notebooks and extensive simulations on the ONE simulator. Moreover, we prove the effectiveness of PDT and HDT by showing that they detect CollusiveHijack attacks with high detection rates while maintaining low false positive rates.

7.2 Future Work

In this section, we present a few ideas for future work.

7.2.1 Including Additional Contextual Information

In our proposed framework, we used the acceleration, the sound, and the Received Signal Strength (RSS) contextual information that are obtained from smartphones' sensors and wireless cards to design our SekGens algorithm and EvilDirectHunter protocol. However, smartphones nowadays are increasingly equipped with a variety of sensors including GPS, cellular radios, gyroscopes, light and proximity. One interesting topic is to investigate whether more sensor(s) can be involved in the design of our framework's algorithm and protocols. Additionally, we currently assume the existence of a collaboration phase, which has to be executed before SekGens algorithm, such that the two Wi-Fi Direct devices agree on the sensors that are involved in the calculation and their sampling rates. It would be interesting to design a real-time algorithm that dynamically decides the type of sensors and the amount of their data to be included based on the current environment of WDONs.

7.2.2 Robustness of SekGens and EvilDirectHunter on Heterogeneous Devices

In this dissertation, we evaluated the performance of SekGens algorithm and EvilDirectHunter protocol on two different smartphones (i.e., Google Nexus 5 and Samsung Galaxy S2) that run Android OS. However, in real-world, smartphones have different OS'es and their sensors and wireless cards have different manufacturers. Moreover, the readings of sensors and wireless cards of similar smartphones might vary due to hardware imperfections, thermal effects, different kernel versions, etc. One potential direction for future exploration is to extensively investigate and enhance the performance of our proposed framework on heterogeneous hardware and software devices. This is, however, challenging as we have already seen the performance degradation of SekGens algorithm and EvilDirectHunter protocol when they run on different smartphones compared to their performance on same smartphones. We envision that a carefully designed hardware-independent based algorithm is needed.

7.2.3 Addressing Other Types of Man-in-the-Middle Attacks

In the attacker model of this dissertation, we assume that the malicious attacker, Eve, can not launch the man-in-the-middle attack in which she is equipped with full-duplex radio transceivers or directional antennas that enable the reception of a signal from one device like the legitimate GO and jamming of the same signal at the clients, or vice versa. One challenging topic is to relax this assumption and enhance the detection capabilities of SekGens algorithm and EvilDirectHunter protocol to address Eve's attempts to relay and possibly alter the communication between two devices in WDONs.

7.2.4 Improving the Accuracy of the Contact Graph in WDONs

A potential topic that deserves further exploring is to incorporate the queuing effects when building the contact graph of WDONs. Currently, both PDT and HDT estimate the packet delay based on the contact graph (as is shown in 6.4) in which the edge weight, $e_{i,j}$, between any two nodes, say v_i and v_j, is defined as the contact frequency, $\lambda_{i,j}$, between them (i.e., the ICT between v_i and $v_j = 1/\lambda_{i,j}$). However, intuitively, if the WDON's device is already holding many messages, the estimated packet delay that is only derived from the ICTs between the nodes is no longer accurate. Hence, it is important to consider this aspect, as the bandwidth, the contact duration, or devices' buffer size might not be sufficient to transmit all the messages during each contact.

7.2.5 PDT and HDT in Real-World WDONs

For PDT and HDT, we have a proof-of-concept implementation and evaluation on wireless testbeds (i.e., Asus Eee notebooks) in which we conducted a dynamic on-off network experiment to create contact events between the nodes as is the case in WDONs. However, it is still important to implement the proposed techniques in smartphones (i.e., as Android services and daemons) and evaluate them in actual WDONs (i.e., with real-world users over a relatively long time period) in order to understand their performance and overhead in real-world scenarios.

REFERENCES

[1] D. Camps-Mur, A. Garcia-Saavedra, and P. Serrano, "Device-to-device communications with wi-fi direct: overview and experimentation," *IEEE Wireless Communications Magazine*, vol. 20, no. 3, 2013.

[2] "Wifi direct file transfer, google play store." `https://play.google.com/store`. Accessed: 2019-03-20.

[3] "Wifi direct file share, google play store." `https://play.google.com/store`. Accessed: 2019-03-20.

[4] W. Hu and G. Cao, "Quality-aware traffic offloading in wireless networks," *MobiHoc*, 2014.

[5] "Wi-fi certified miracast." `http://www.wi-fi.org/discover-wi-fi/wi-fi-certified-miracast`. Accessed: 2019-03-20.

[6] "Hp mobile printing." `http://www8.hp.com/us/en/ads/mobility/wireless-direct-printing.html`. Accessed: 2019-03-20.

[7] "Binary2study, google play store." `https://play.google.com/store`. Accessed: 2019-03-20.

[8] "Abi research." `https://www.abiresearch.com/`. Accessed: 2019-03-20.

[9] "Distressnet-ng: Resilient mobile broadband communication and edge computing." `http://distressnet.net/`. Accessed: 2019-03-20.

[10] "Firechat, google play store." `https://play.google.com/store`. Accessed: 2019-03-20.

[11] "1am: Censorship-resistant microblogging." `http://ziyang.eecs.umich.edu/projects/whisper/index.html`. Accessed: 2019-03-20.

[12] "Wi-fi protected setup specification version 1.0h." `http://cfile28.uf.tistory.com/attach/16132E3C50FCFFCB3EC74E`. Accessed: 2019-03-20.

[13] S. Viehbock, "Wifi protected setup (wps) pin brute force vulnerability, cert vulnerability note vu#723755." `https://www.kb.cert.org/vuls/id/723755/`, 2011. Accessed: 2019-03-20.

[14] D. Bongard, "Wi-fi protected setup pixie dust offline pin brute force vulnerability." `https://github.com/wiire-a/pixiewps`, 2014. Accessed: 2019-03-20.

[15] A. Sanatinia, S. Narain, and G. Noubir, "Wireless spreading of wifi aps infections using wps flaws: An epidemiological and experimental study," *CNS*, 2013.

[16] "802.11i-2004, ieee standard for information technology, telecommunications and information exchange between systems, local and metropolitan area networks, amendment 6: Medium access control (mac) security enhancements.," *IEEE Std 802.11i-2004*, 2004.

[17] W. Wei, S. Jaiswal, J. Kurose, D. Towsley, K. Suh, and B. Wang, "Identifying 802.11 traffic from passive measurements using iterative bayesian inference," *IEEE/ACM Transactions on Networking*, vol. 20, no. 2, 2012.

[18] W. Wei, K. Suh, B. Wang, Y. Gu, J. Kurose, D. Towsley, and S. Jaiswal, "Passive online detection of 802.11 traffic using sequential hypothesis testing with tcp ack-pairs," *IEEE Transactions on Mobile Computing*, vol. 8, no. 3, 2009.

[19] A. Venkataraman and R. Beyah, "Rogue access point detection using innate characteristics of the 802.11 mac," *SecureComm*, 2009.

[20] S. Jana and S. K. Kasera, "On fast and accurate detection of unauthorized wireless access points using clock skews," *IEEE Transactions on Mobile Computing*, vol. 9, no. 3, 2010.

[21] "Tired of rogues? solutions for detecting and eliminating rogue wireless networks, white paper." `https://catalog.m4dconnect.com/docs/Tired_of_Rogues.pdf`, 2011. Accessed: 2019-03-20.

[22] P. Bahl, R. Chandra, J. Padhye, L. Ravindranath, M. Singh, A. Wolman, and B. Zill, "Enhancing the security of corporate wi-fi networks using dair," *MobiSys*, 2006.

[23] H. Alipour, Y. B. Al-Nashif, P. Satam, and S. Hariri, "Wireless anomaly detection based on ieee 802.11 behavior analysis," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 10, 2015.

[24] R. Gill, J. Smith, and A. Clark, "Experiences in passively detecting session hijacking attacks in ieee 802.11 networks," *ACSW*, 2006.

[25] X. Long and B. Sikdar, "A mechanism for detecting session hijacks in wireless networks," *IEEE Transactions on Wireless Communications*, vol. 9, no. 4, 2010.

[26] C. Yang and R. Stoleru, "Hybrid routing in wireless networks with diverse connectivity," *MobiHoc*, 2016.

[27] A. Lindgren, A. Doria, E. Davies, and O. Schelén, "Probabilistic routing in intermittently connected networks," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 7, no. 3, 2003.

[28] A. Lindgren, A. Doria, E. Davies, and S. Grasic, "Probabilistic routing protocol for intermittently connected networks. irtf rfc 6693." `https://tools.ietf.org/html/rfc6693`, 2012. Accessed: 2019-03-20.

[29] A. Vahdat and D. Becker, "Epidemic routing for partially-connected ad hoc networks," *Technical report, Duke University*, 2000.

[30] R. Patil and M. P. Tahiliani, "Detecting packet modification attack by misbehaving router," *ICNSC*, 2014.

[31] J. Burgess, G. D. Bissias, M. Corner, and B. N. Levine, "Surviving attacks on disruption-tolerant networks without authentication," *MobiHoc*, 2007.

[32] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Robust smartphone app identification via encrypted network traffic analysis," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 1, 2018.

[33] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, “Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic,” *EuroS&P*, 2016.

[34] R. Lu, X. Lin, H. Zhu, X. Shen, and B. Preiss, “Pi: A practical incentive protocol for delay tolerant networks,” *IEEE Transactions on Wireless Communications*, vol. 9, no. 4, 2010.

[35] Q. Li, W. Gao, S. Zhu, and G. Cao, “To lie or to comply: Defending against flood attacks in disruption tolerant networks,” *IEEE Transactions on Dependable and Secure Computing*, vol. 10, no. 3, 2013.

[36] Y. Ren, M. C. Chuah, J. Yang, and Y. Chen, “Detecting wormhole attacks in delay-tolerant networks,” *IEEE Wireless Communications*, vol. 17, no. 5, 2010.

[37] F. Li, J. Wu, and A. Srinivasan, “Thwarting blackhole attacks in disruption-tolerant networks using encounter tickets,” *INFOCOM*, 2009.

[38] Q. Li and G. Cao, “Mitigating routing misbehavior in disruption tolerant networks,” *IEEE Transactions on Information Forensics and Security*, vol. 7, no. 2, 2012.

[39] M. Alajeely, R. Doss, A. Ahmad, and V. Mak-Hau, “Defense against packet collusion attacks in opportunistic networks,” *Computers and Security*, vol. 65, no. C, 2017.

[40] “Bgpmon.” `http://www.bgpmon.net`. Accessed: 2019-03-20.

[41] “Ripe myasn system.” `https://www.ripe.net/`. Accessed: 2019-03-20.

[42] M. Lad, D. Massey, D. Pei, Y. Wu, B. Zhang, and L. Zhang, “Phas: A prefix hijack alert system,” *USENIX*, 2006.

[43] Y. Chi, R. Oliveira, and L. Zhang, “Cyclops: The as-level connectivity observatory,” *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 5, 2008.

[44] P. Sermpezis, V. Kotronis, P. Gigis, X. Dimitropoulos, D. Cicalese, A. King, and A. Dainotti, “Artemis: neutralizing bgp hijacking within a minute,” *Technical Report*, 2018.

[45] “Bgpmon: a client server application to store and analyze large amounts of bgp data.” `https://github.com/CSUNetSec/bgpmon`. Accessed: 2019-03-20.

[46] "Routing information service (ris)." `https://www.ripe.net/`. Accessed: 2019-03-20.

[47] "Route views project." `http://www.routeviews.org/`. Accessed: 2019-03-20.

[48] Z. Zhang, Y. Zhang, Y. C. Hu, Z. M. Mao, and R. Bush, "ispy: Detecting ip prefix hijacking on my own," *IEEE/ACM Transactions on Networking*, vol. 18, no. 6, 2010.

[49] C. Zheng, L. Ji, D. Pei, J. Wang, and P. Francis, "A light-weight distributed scheme for detecting ip prefix hijacks in real-time," *SIGCOMM*, 2007.

[50] S. Mathur, W. Trappe, N. Mandayam, C. Ye, and A. Reznik, "Radio-telepathy: Extracting a secret key from an unauthenticated wireless channel," *MobiCom*, 2008.

[51] A. Altaweel, R. Stoleru, and S. Mandal, "On secure shared key establishment for mobile devices using contextual information," *IPCCC*, 2015.

[52] A. Altaweel, R. Stoleru, and G. Gu, "Evildirect: A new wi-fi direct hijacking attack and countermeasures," *ICCCN*, 2017.

[53] A. Altaweel, R. Stoleru, G. Gu, and A. K. Maity, "Collusivehijack: A new route hijacking attack and countermeasures in opportunistic networks," *CNS*, 2019.

[54] J. Frank and J. Massey, "The kolmogorov-smirnov test for goodness of fit," *Journal of the American Statistical Association*, vol. 46, no. 253, 1951.

[55] S. Symington, S. Farrell, H. Weiss, and P. Lovell, "Bundle security protocol specification. irtf rfc 6257." `https://tools.ietf.org/html/rfc6257`, 2011. Accessed: 2019-03-20.

[56] S. Gianvecchio and H. Wang, "An entropy-based approach to detecting covert timing channels," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 6, 2011.

[57] A. L. Toledo and X. Wang, "Robust detection of selfish misbehavior in wireless networks," *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 6, 2007.

[58] J. M. Estévez-Tapiador, P. García-Teodoro, and J. E. Díaz-Verdejo, "Measuring normality in http traffic for anomaly-based intrusion detection," *Computer Networks*, vol. 45, no. 2, 2004.

[59] J. B. D. Caberera, B. Ravichandran, and R. K. Mehra, "Statistical traffic modeling for network intrusion detection," *MASCOTS*, 2000.

[60] I. Borg and P. J. F. Groenen, *Modern Multidimensional Scaling*. Springer Series in Statistics, 2005.

[61] A. Keranen, J. Ott, and T. Karkkainen, "The one simulator for dtn protocol evaluation," *Simutools*, 2009.

[62] B. Haines, *Seven deadliest wireless technologies attacks*. Elsevier Science & Technology, 2010.

[63] L. E. Holmquist, F. Mattern, B. Schiele, P. Alahuhta, M. Beigl, and H.-W. Gellersen, "Smart-its friends: A technique for users to easily establish connections between smart artefacts," *UbiComp*, 2001.

[64] J. Lester, B. Hannaford, and G. Borriello, "Are you with me? using accelerometers to determine if two devices are carried by the same person," *Pervasive*, 2004.

[65] R. Mayrhofer and H. Gellersen, "Shake well before use: authentication based on accelerometer data," *Pervasive*, 2007.

[66] D. Bichler, G. Stromberg, M. Huemer, and M. Low, "Key generation based on acceleration data of shaking processes," *UbiComp*, 2007.

[67] S. Jana, S. N. Premnath, M. Clark, S. K. Kasera, N. Patwari, and S. V. Krishnamurthy, "On the effectiveness of secret key extraction from wireless signal strength in real environments," *MobiCom*, 2009.

[68] S. Ali, V. Sivaraman, and D. Ostry, "Zero reconciliation secret key generation for body-worn health monitoring devices," *WiSec*, 2012.

[69] M. Miettinen, N. Asokan, T. D. Nguyen, A.-R. Sadeghi, and M. Sobhani, “Context-based zero-interaction pairing and key evolution for advanced personal devices,” *CCS*, 2014.

[70] M. Sethi, M. Antikainen, and T. Aura, “Commitment-based device pairing with synchronized drawing,” *PerCom*, 2014.

[71] S. Reddy, K. Shilton, G. Denisov, C. Cenizal, D. Estrin, and M. Srivastava, “Biketastic: sensing and mapping for better biking,” *CHI*, 2010.

[72] S. Eisenman, E. Miluzzo, N. Lane, R. Peterson, G. Ahn, and A. Campbel, “The bikenet mobile sensing system for cyclist experience mapping,” *SenSys*, 2007.

[73] A. Thiagarajan, T. Gerlich, J. Biagioni, and J. Eriksson, “Cooperative transit tracking using GPS-enabled smartphones,” *SenSys*, 2010.

[74] E. Owusu, J. Han, S. Das, A. Perrig, and J. Zhang, “Accessory: Password inference using accelerometers on smartphones,” *HotMobile*, 2012.

[75] C. Yang, Y. Song, and G. Gu, “Active user-side evil twin access point detection using statistical techniques,” *IEEE Transactions on Information Forensics and Security*, vol. 7, no. 5, 2012.

[76] W. Wei, B. Wang, C. Zhang, J. Kurose, and D. Towsley, “Classification of access network types: Ethernet wireless lan, adsl, cable modem or dialup?,” *INFOCOM*, 2005.

[77] H. Han, B. Sheng, C.Tan, Q. Li, and S. Lu, “A measurement based rogue ap detection scheme,” *INFOCOM*, 2009.

[78] V. Baiamonte, K. Papagiannaki, and G. Iannaccone, “Detecting 802.11 wireless hosts from remote passive observations,” *IFIP-TC6 Networking*, 2007.

[79] S. Shetty, M. Song, and L. Ma, “Rogue access point detection by analyzing network traffic characteristics,” *MILCOM*, 2007.

[80] R. Beyah, S. Kangude, G. Yu, B. Strickland, and J. Copeland, “Rogue access point detection using temporal traffic characteristics,” *GLOBECOM*, 2004.

[81] H. Yin, G. Chen, and J. Wang, "Detecting protected layer-3 rogue aps," *BROADNETS*, 2007.

[82] L. Watkins, R. Beyah, and C. Corbett, "A passive approach to rogue access point detection," *GLOBECOM*, 2007.

[83] C. D. Mano, A. Blaich, Q. Liao, Y. Jiang, D. A. Cieslak, D. A. Salyers, and A. Striegel, "Ripps: Rogue identifying packet payload slicer detecting unauthorized wireless hosts through network traffic conditioning," *ACM Transactions on Information and System Security*, vol. 11, no. 2, 2008.

[84] "Wisentry - wireless access point detection system." `http://www.wimetrics.com/Products/WAPD.htm`. Accessed: 2019-03-20.

[85] "The airmagnet project." `http://www.airmagnet.com/`. Accessed: 2019-03-20.

[86] "Rogue access point detection: Automatically detect and manage wireless threats to your network, white paper." `http://www.proxim.com/`, 2004. Accessed: 2019-03-20.

[87] H. Mustafa and W. Xu, "Cetad: Detecting evil twin access point attacks in wireless hotspot-s," *CNS*, 2014.

[88] H. Ballani, P. Francis, and X. Zhang, "A study of prefix hijacking and interception in the internet," *SIGCOMM*, 2007.

[89] L. Subramanian, V. Roth, I. Stoica, S. Shenker, and R. H. Katz, "Listen and whisper: Security mechanisms for bgp," *NSDI*, 2004.

[90] X. Hu and Z. M. Mao, "Accurate real-time identification of ip prefix hijacking," *S&P*, 2007.

[91] X. Shi, Y. Xiang, Z. Wang, X. Yin, and J. Wu, "Detecting prefix hijackings in the internet with argus," *IMC*, 2012.

[92] J. Schlamp, R. Holz, Q. Jacquemart, G. Carle, and E. W. Biersack, "Heap: Reliable assessment of bgp hijacking attacks," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 6, 2016.

[93] M. Lepinski and K. Sriram, “Bgpsec protocol specification. ietf rfc 8205.” `https://tools.ietf.org/html/rfc8205`, 2017. Accessed: 2019-03-20.

[94] M. Lepinski and S. Kent, “An infrastructure to support secure internet routing. ietf rfc 6480.” `https://tools.ietf.org/html/rfc6480`, 2012. Accessed: 2019-03-20.

[95] S. Kent, C. Lynn, and K. Seo, “Secure border gateway protocol (s-bgp),” *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 4, 2000.

[96] C. Yang and R. Stoleru, “Hybrid routing in wireless networks with diverse connectivity,” *Technical Report, Texas A&M University*, 2018.

[97] “Ibr-dtn - a modular and lightweight implementation of the bundle protocol.” `https://github.com/ibrdtn/`. Accessed: 2019-03-20.

[98] M. Doering, S. Lahde, J. Morgenroth, and L. Wolf, “Ibr-dtn: An efficient implementation for embedded systems,” *CHANTS*, 2008.

[99] S. Mandal, C. Yang, A. Altaweel, and R. Stoleru, “An efficient pairwise key establishment scheme for ad-hoc mobile clouds,” *WiMob*, 2015.

[100] L. Davi, A. Dmitrienko, A.-R. Sadeghi, and M. Winandy, “Privilege escalation attacks on android,” *ISC*, 2010.

[101] B. Hassanshahi and R. H. Yap, “Android database attacks revisited,” *AsiaCCS*, 2017.

[102] “Reaver open source.” `https://code.google.com/p/reaver-wps/`. Accessed: 2019-03-20.

[103] “Aircrack-ng.” `https://www.aircrack-ng.org/`. Accessed: 2019-03-20.

[104] D. Zisiadis, S. Kopsidas, A. Varalis, and L. Tassiulas, “Enhancing wps security,” *WD*, 2012.

[105] “Waidps, wireless auditing intrusion detection and prevention system.” `https://github.com/SYWorks/waidps`. Accessed: 2019-03-20.

[106] G. Hancke, “A practical relay attack on iso 14443 proximity cards,” *Technical report, University of Cambridge Computer Laboratory*, 2005.

[107] S. A. PANDA, "Preventing man-in-the-middle attacks in near field communication by out-of-band key exchange," *Master's thesis, Texas A&M University*, 2016.

[108] T. S. Heydt-Benjamin, D. V. Bailey, K. Fu, A. Juels, and T. O'Hare, "Vulnerabilities in first-generation rfid-enabled credit cards," *FC/USEC*, 2007.

[109] E. Haselsteiner and K. Breitfuß, "Security in near field communication (nfc) strengths and weaknesses," *RFIDSec*, 2006.

[110] "Androidpay, google play store." `https://www.android.com/pay/`. Accessed: 2019-03-20.

[111] "Apple pay, apple." `https://www.apple.com/apple-pay/`. Accessed: 2019-03-20.

[112] A. Juels, D. Molnar, and D. Wagner, "Security and privacy issues in e-passports," *SecureComm*, 2005.

[113] Z. Kfir and A. Wool, "Picking virtual pockets using relay attacks on contactless smartcard," *SecureComm*, 2005.

[114] G. P. Hancke, "Practical eavesdropping and skimming attacks on high-frequency rfid tokens," *Journal of Computer Security*, vol. 19, no. 2, 2011.

[115] L. Francis, G. Hancke, K. Mayes, and K. Markantonakis, "Practical nfc peer-to-peer relay attack using mobile phones," *RFIDSec*, 2011.

[116] C. Mulliner, "Vulnerability analysis and attacks on nfc-enabled mobile phones," *ARES*, 2009.

[117] "Signal amplification relay attack (sara)." `https://hackernoon.com/signal-amplification-relay-attack-sara-609ce6c20d4f`. Accessed: 2019-03-20.

[118] "Ecma-385: Nfc-sec: Nfcip-1 security services and protocol." `http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-385.pdf`, 2015. Accessed: 2019-03-20.

[119] "Ecma-386: Nfc-sec-01: Nfc-sec cryptography standard using ecdh and aes." `http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-386.pdf`, 2015. Accessed: 2019-03-20.

[120] "Ecma-409: Nfc-sec-02: Nfc-sec cryptography standard using ecdh-256 and aes-gcm." `https://www.ecma-international.org/publications/files/ECMA-ST/ECMA-409.pdf`, 2015. Accessed: 2019-03-20.

[121] "Ecma-410: Nfc-sec-03: Nfc-sec entity authentication and key agreement using asymmetric cryptography." `https://www.ecma-international.org/publications/files/ECMA-ST/ECMA-410.pdf`, 2017. Accessed: 2019-03-20.

[122] "Ecma-411: Nfc-sec-04: Nfc-sec entity authentication and key agreement using symmetric cryptography." `https://www.ecma-international.org/publications/files/ECMA-ST/ECMA-411.pdf`, 2017. Accessed: 2019-03-20.

[123] G. Brassard and L. Salvail, "Secret-key reconciliation by public discussion," *EUROCRYPT*, 1994.

[124] "Building lineageos android code." `https://wiki.lineageos.org/`. Accessed: 2019-03-20.

[125] "Accelerometer monitor, google play store." `https://play.google.com/store`. Accessed: 2019-03-20.

[126] "Sound monitor, google play store." `https://play.google.com/store`. Accessed: 2019-03-20.

[127] "Android debug bridge." `http://developer.android.com/tools/help/adb.html`. Accessed: 2019-03-20.

[128] G. Ballou, *Handbook for Sound Engineers*. Elsevier, 2008.

[129] G. Muller and M. Moser, *Handbook of Engineering Acoustics*. Springer, 2013.

[130] L. E. Bassham, III, A. L. Rukhin, J. Soto, J. R. Nechvatal, M. E. Smid, E. B. Barker, S. D. Leigh, M. Levenson, M. Vangel, D. L. Banks, N. A. Heckert, J. F. Dray, and S. Vo, "A statistical test suite for random and pseudorandom number generators for cryptographic applications," *Technical Report*, 2010.

[131] W. K. Edwards, "Discovery systems in ubiquitous computing," *IEEE Pervasive Computing*, vol. 5, no. 2, 2006.

[132] "802.11u-2011, wireless lan medium access control (mac) and physical layer (phy) specifications amendment 9: Interworking with external networks." `https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5721908&tag=1`, 2011. Accessed: 2019-03-20.

[133] "Wifiscanner, google play store." `https://play.google.com/store`. Accessed: 2019-03-20.

[134] R. Stoleru, H. Wu, and H. Chenji, "Secure neighbor discovery in mobile ad hoc networks," *MASS*, 2011.

[135] W. Debus, "Rf path loss and transmission distance calculations, technical memorandum," *Technical report, AXONN*, 2006.

[136] H. Pham, *Handbook of Engineering statistics*. Springer, 2003.

[137] C. Marsh, "Introduction to continuous entropy," *Department of Computer Science, Princeton University*, 2013.

[138] N. Eagle and A. Pentland, "Reality mining: Sensing complex social systems," *Personal and Ubiquitous Computing*, vol. 10, no. 4, 2006.

[139] X. Tie, A. Venkataramani, and A. Balasubramanian, "R3: Robust replication routing in wireless networks with diverse connectivity characteristics," *MobiCom*, 2011.

[140] A. Balasubramanian, B. Levine, and A. Venkataramani, "Dtn routing as a resource allocation problem," *SIGCOMM*, 2007.

[141] K. Smaili, T. Kadri, and S. Kadry, "Hypoexponential distribution with different parameters," *Applied Mathematics*, vol. 4, no. 4, 2013.

[142] J. C. Ferreira and C. M. Patino, "What does the p value really mean?," *Jornal Brasileiro de Pneumologia*, vol. 41, no. 5, 2015.

[143] "The r project for statistical computing." `https://www.r-project.org/`. Accessed: 2019-03-20.

[144] "Rcpp for seamless r and c++ integration." `http://www.rcpp.org/`. Accessed: 2019-03-20.

[145] B. Lynn, "The pairing-based cryptography (pbc) library." `https://crypto.stanford.edu/pbc/`. Accessed: 2019-03-20.

[146] F. Hess, "Efficient identity based signature schemes based on pairings," *SAC*, 2002.

[147] M. McNett and G. M. Voelker, "Access and mobility of wireless pda users," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 9, no. 2, 2005.

[148] P. Jacquet, P. Muhlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot, "Optimized link state routing protocol for ad hoc networks," *INMIC*, 2001.

[149] A. Tonnesen, "Implementing and extending the optimized link state routing protocol," *Master's thesis, University of Oslo*, 2004.

[150] M. Conti, F. Delmastro, G. Minutiello, and R. Paris, "Experimenting opportunistic networks with wifi direct," *IFIP*, 2013.

[151] M. Usman, M. R. Asghar, I. S. Ansari, F. Granelli, and K. Qaraqe, "Towards energy efficient multi-hop d2d networks using wifi direct," *GLOBECOM*, 2017.

APPENDIX A

WI-FI DIRECT REGISTRATION PROTOCOL

The WPS provisioning messages between the Registrar (R) and the Enrollee (E) for the in-band configuration mode are as follow (page 33 of [12]):

1. $E \rightarrow R$: M_1=Version$\|N_1\|$Description$\|PK_E$

2. $E \leftarrow R$: M_2 = Version $\|$ N_1 $\|$ N_2 $\|$ Description $\|$ PK_R [$\|$ ConfigData] $\|$ $HMAC_{AuthKey}(M_1 \| M_2^*)$

3. $E \rightarrow R$: M_3 = Version $\|$ N_2 $\|$ E-Hash1 $\|$ E-Hash2 $\|$ $HMAC_{AuthKey}(M_2 \| M_3^*)$

4. $E \leftarrow R$: M_4 = Version $\|$ N_1 $\|$ R-Hash1 $\|$ R-Hash2 $\|$ $ENC_{KeyWrapKey}$(R-S1) $\|$ $HMAC_{AuthKey}(M_3 \| M_4^*)$

5. $E \rightarrow R$: M_5 = Version $\|$ N_2 $\|$$ENC_{KeyWrapKey}$(E-S1) $\|$ $HMAC_{AuthKey}(M_4 \| M_5^*)$

6. $E \leftarrow R$: M_6 = Version $\|$ N_1 $\|$ $ENC_{KeyWrapKey}$(R-S2) $\|$ $HMAC_{AuthKey}(M_5 \| M_6^*)$

7. $E \rightarrow R$: M_7 = Version $\|$ N_2 $\|$ $ENC_{KeyWrapKey}$(E-S2 [$\|$ ConfigData]) $\|$ $HMAC_{AuthKey}(M_6 \| M_7^*)$

8. $E \leftarrow R$: M_8 = Version $\|$ N_1 $\|$ [$ENC_{KeyWrapKey}$ (ConfigData)] $\|$ $HMAC_{AuthKey}(M_7 \| M_8^*)$

Such that:

- $\|$ means concatenation of parameters to form a message.
- **Subscripts** when used in the context of a cryptographic function such as $HMAC_{Key}$ refer to the key used by that function.

- $\mathbf{M}_n^*$ is message M_n excluding the HMAC-SHA-256 value.
- **Version** identifies the type of Registration Protocol message.
- $\mathbf{N}_1$ is a 128-bit random number specified by the Enrollee.
- $\mathbf{N}_2$ is a 128-bit random number specified by the Registrar.
- **Description** contains a human-readable description of the sending device (model number, MAC address, UUID, manufacturer, etc.) and device capabilities like: I/O channels, supported algorithms, Registration Protocol role, etc.
- $\mathbf{PK}_R$ and $\mathbf{PK}_E$ are Diffie-Hellman public keys of the Registrar and Enrollee, respectively.
- **AuthKey** is an authentication key derived from the Diffie-Hellman secret g^{AB} mod p, MAC address of Enrollee, and the nonces N_1 and N_2.
- **R-Hash1**, **R-Hash2** are pre-commitments, which made by the Registrar to prove knowledge of the two halves of the Enrollee's device password.
- **E-Hash1**, **E-Hash2** are pre-commitments, which made by the Enrollee to prove knowledge of the two halves of its own device password.
- $\mathbf{ENC}_{KeyWrapKey}$**(...)** This notation represents symmetric encryption of the values in parentheses via the key KeyWrapKey. The encryption algorithm is AES-CBC per FIPS 197, with PKCS#5 v2.0 padding.
- **R-S1**, **R-S2** are secret 128-bit nonces, which are used with R-Hash1 and R-Hash2 by the Enrollee to confirm the Registrar's knowledge of the two halves of the Enrollee's device password.
- **E-S1**, **E-S2** are secret 128-bit nonces, which are used with E-Hash1 and E-Hash2 by the Registrar to confirm the Enrollee's knowledge of the two halves of the Enrollee's device password.

- **HMAC$_{AuthKey}$(...)** This notation represents an Authenticator attribute, which has a HMAC keyed hash over the values in parentheses and using the key AuthKey. The keyed hash function is HMAC-SHA-256 per FIPS 180-2 and RFC-2104.

- **ConfigData** has WLAN settings and credentials for the Enrollee.

- **M2D Registrar Discovery Message** Registrars may respond to Enrollees in-band through M2D instead of M2 if they do not know the Enrollee's Device Password. M2D is used to provide the Enrollee with information about the Registrar. The Enrollee sends an M3 message and proceeds with the Registration Protocol exchange only if it receives an M2 message from the Registrar.

E-Hash1 and E-Hash2 in M_3 are computed by the Enrollee after creating two secret nonces E-S1 and E-S:

E-Hash1 = HMAC$_{AuthKey}$(E-S1 || PSK1 || PK$_E$ || PK$_R$).

E-Hash2 = HMAC$_{AuthKey}$(E-S2 || PSK2 || PK$_E$ || PK$_R$).

Also, R-Hash1 and R-Hash2 in M_4 are computed by the Registrar after creating two secret nonces R-S1 and R-S2:

R-Hash1 = HMAC$_{AuthKey}$(R-S1 || PSK1 || PK$_E$ || PK$_R$).

R-Hash2 = HMAC$_{AuthKey}$(R-S2 || PSK2 || PK$_E$ || PK$_R$).

PSK1 and PSK2 are calculated by converting the device password into two 128-bit values as follows:

PSK1 = first 128 bits of HMAC$_{AuthKey}$(1st half of Device Password). **This is the first part of the key that SekGens will create from sensor data.**

PSK2 = first 128 bits of HMAC$_{AuthKey}$(2nd half of Device Password). **This is the second part of the key that SekGens will create from sensor data.**