SECURITY OF SHORT-RANGE WIRELESS TECHNOLOGIES AND AN AUTHENTICATION PROTOCOL FOR IOT

by

KARIM LOUNIS

A thesis submitted to the
School of Computing
in conformity with the requirements for
the degree of Doctor of Philosophy

Queen’s University
Kingston, Ontario, Canada
January 2021

Copyright © Karim Lounis, 2021
Abstract

The number of wireless IoT (Internet of Thing) infrastructures is considerably growing as thousands of new devices are being connected everyday. This has allowed the emergence of various smart applications that have changed the lifestyle of many people. However, as security has always been an afterthought to innovation, the security of these infrastructures has raised serious concerns. The research community has to conduct serious investigations before IoT grows further and it becomes too late. Thus, we devote this Ph.D. thesis to study and improve the security of IoT wireless infrastructures, in particular, those adopting Wi-Fi, Bluetooth, ZigBee, and RFID technologies. To that end, we divide this thesis into two parts. In Part 1, we review the technologies and propose an attack taxonomy to classify existing attacks on these technologies and discuss possible countermeasures. Moreover, we analyze the security of these technologies and identify new vulnerabilities and attacks. In Part 2, we analyze the security of recent PUF (Physical Unclonable Function)-based authentication protocols and propose a generic authentication protocol for IoT. We specifically design the protocol for wireless Thing-to-Thing communication scheme. We implement the protocol on resource-constrained devices and perform security as well as performance analysis. We show that the proposed protocol is secure and efficient in terms of execution time, communication overhead, and energy consumption.
Acknowledgements

All the praises and thanks be to Allah, Lord of all the worlds, the most beneficent, the most merciful. This thesis would have never happened without His will and facilitation. And Allah’s Peace and Blessings be upon His Final Messenger, Mohammed, his pure family, his noble Companions, and all those who follow them with righteousness until the Day of Judgment.

I must thank many people here. I promise I will do my best to mention as many as possible. My deepest apologies to those who I will miss.

First, I would like to thank my supervisor Prof. Mohammad Zulkernine for many reasons. The first reason is for accepting me to be his Ph.D. student when we first talked in July 2017. Secondly, I would thank him for his guidance and encouragement during these three years. I remember he told me once ”You should finish the courses and start writing papers soon”, which was within the first few months of my Ph.D. As speedy and compelling as it sounds, that was a boost of motivation and has pushed me to work harder and aim for the best. It’s been a pleasure to be his apprentice. I also like to thank him for his insights about conducting research, critical thinking, and managing publications, which have made me a much better researcher although I had some previous experience in research.

I want to deeply express my gratitude to my supervisory committee members: Prof. David B. Skillicorn and Prof. Ahmad E. Hassan, who have accepted to take this important role throughout my Ph.D. degree. Also, I would like to thank them for their encouraging feedback when acknowledging my progress reports at each stage of my Ph.D.
Thanks to Prof. Mohamed Younis (University of Maryland Baltimore County), Prof. Thomas R. Dean, Prof. Selim G. Akl, and Dr. Yuan Tian (Queen’s University), for taking the time out of their busy schedule to read this thesis and provide valuable feedback.

I want to thank the Canada Research Chairs (CRC), the Natural Sciences and Engineering Research Council of Canada (NSERC), and MITACS for the financial support.

I would like to thank the Queen’s Reliable Software Technology (QRST) group members (including those who I met but left before my defense) for their moral and physical support. Also, I would thank Adel Ibrahim from Queen’s Telecommunications Research Lab (TLR) for all the discussions we had on the field of telecommunication.

I want to thank Prof. Hossam Hassanein, the director of the School of Computing, for granting me the opportunity for teaching during my Ph.D. program. Also, I want to thank many of the faculty members in the School of Computing, who have shown their support, provided administrative guidance, and technical help. Special thanks to Debby Robertson, Tom Bradshaw, Prof. Juergen Dingel, Dr. Joshua Dunfield, and Aaron Visser.

I would like to thank my previous supervisors/instructors who encouraged me a lot and provided me with knowledge to reach this stage. Special thanks to Profs. Abdelkader Belkhir (USTHB), Mahfoud Benchailba (USTHB), Nadjib Badache (CERIST), Lynda Mokdad (UPEC), Sjuoke Mauw (UL), Jean-Michel Fourneau (UVSQ), and Gaetan Hains (UPEC), and Drs. AbdelAziz Babakhoyya (CERIST), Chafika Benzaid (USTHB), Khaled Zeraoulia (USTHB), Haroun Benkaouha (USTHB), and Youcef Zafoune (USTHB). Also, many thanks to the Sensor Network Team of CERIST.

Finally, I would like to express my deepest thanks to my family, especially my parents (as well as my parents in law), brother (+in law), sister (+in law), and wife, for their support, prayers, encouragement, and understanding throughout my Ph.D. adventure. Also, I want to thank my friends, neighbors in my home country, and close colleagues: Samir Ouchani, Abderrahmane Mayouche, Omar ElHarthi, Fouad Boutiche, Merouan Rakik, Khaled Berriche, Othmane Marouani, Walid Smara, Ahmed ElBari, Billel Yadoughi, Anas...

To all and every single one of you, many thanks.

I dedicate this thesis to my parents (including my parents in law), my wife Oum Baraa, and my three children AbdelAllah, Djoumana, and Hind.

Karim Lounis,

Statement of Originality

I hereby declare that I am the sole author of this thesis. Ideas and techniques contributed by others are fully acknowledged in accordance with the standard referencing practices. Also, I declare that this is a true copy of the thesis, including any required final revisions, as accepted by my examiners. I authorize Queen’s University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research. I understand that my thesis may be made electronically available to the public.

Karim Lounis,

# Contents

Abstract i

Acknowledgements ii

Statement of Originality v

Contents vi

List of Tables x

List of Figures xi

Chapter 1: Introduction 1

1.1 Internet of Things ........................................ 1
1.2 IoT Security .................................................. 2
1.3 Motivation and Research Objectives .............................. 3
  1.3.1 IoT Wireless Infrastructures Security ....................... 3
  1.3.2 Authentication Issue in IoT ................................. 4
1.4 Contributions .................................................. 6
1.5 Thesis Organization ........................................... 7

I Security of IoT Short-Range Wireless Technologies 9

Chapter 2: IoT Short-Range Wireless Technologies 10

2.1 Introduction .................................................. 10
2.2 Wi-Fi Technology .............................................. 11
  2.2.1 Wi-Fi Overview ........................................ 11
  2.2.2 Wi-Fi Security .......................................... 11
2.3 Bluetooth Technology ......................................... 13
  2.3.1 Bluetooth Overview ..................................... 13
  2.3.2 Bluetooth Security .................................... 14
2.4 ZigBee Technology .............................................. 15
  2.4.1 ZigBee Overview ....................................... 15
  2.4.2 ZigBee Security ........................................ 15
Chapter 3: Taxonomy and Analysis of IoT Wireless Attacks

3.1 Introduction .......................................................... 20
3.2 Taxonomy of Attacks ............................................... 21
    3.2.1 IoT Security Threats ..................................... 21
    3.2.2 Fundamental Security Services .......................... 21
    3.2.3 Attack Classification Scheme ........................... 22
3.3 Attacks on Wi-Fi Technology .................................... 26
    3.3.1 Fabrication Attacks on Wi-Fi .......................... 26
    3.3.2 Interception Attacks on Wi-Fi ........................ 29
    3.3.3 Modification attacks on Wi-Fi ........................ 34
    3.3.4 Interruption Attacks on Wi-Fi ......................... 36
    3.3.5 Domination Attacks on Wi-Fi ......................... 41
3.4 Attacks on Bluetooth Technology ............................... 46
    3.4.1 Fabrication attacks on Bluetooth .................... 46
    3.4.2 Interception attacks on Bluetooth ................... 48
    3.4.3 Modification Attacks on Bluetooth .................. 53
    3.4.4 Interruption Attacks on Bluetooth ................... 54
    3.4.5 Domination attacks on Bluetooth .................... 58
3.5 Attacks on ZigBee Technology .................................. 61
    3.5.1 Fabrication attack on ZigBee ........................ 61
    3.5.2 Interception attacks on ZigBee ...................... 63
    3.5.3 Modification Attacks on ZigBee ...................... 66
    3.5.4 Interruption Attacks on ZigBee ...................... 67
    3.5.5 Domination Attacks on ZigBee ....................... 70
3.6 Attacks on RFID Technology .................................. 71
    3.6.1 Fabrication Attacks on RFID ......................... 71
    3.6.2 Interception Attacks on RFID ....................... 76
    3.6.3 Modification Attacks on RFID ....................... 79
    3.6.4 Interruption Attacks on RFID ....................... 79
    3.6.5 Domination Attacks on RFID ......................... 80
3.7 Conclusion ......................................................... 81

Chapter 4: Vulnerability Identification and Mitigation in Wireless IoT 82

4.1 Introduction .......................................................... 82
4.2 Wi-Fi Compromise .................................................. 83
    4.2.1 Race-Condition Vulnerability ...................... 83
    4.2.2 WPA3-SAE Authentication Phases ..................... 84
    4.2.3 Attacks on WPA3-SAE through Race-Condition .... 88
List of Tables

4.1 Self-connection attack experimental results. ............................ 118
4.2 Connection dumping attack experimental results ....................... 130

5.1 Multi-features comparison of PUF-based authentication protocols .... 187

6.1 Notation adopted for T2T-MAP ............................................. 225
6.2 ARDUINO boards used to implement T2T-MAP ......................... 245
6.3 Size of T2T-MAP codes (Configuration 1) ............................... 250
6.4 Size of T2T-MAP codes (Configuration 2) ............................... 250
6.5 Size of messages in T2T-MAP .............................................. 259
6.6 T2T-MAP Execution time measured using oscilloscope (Configuration 1) . 269
6.7 T2T-MAP Execution time measured using oscilloscope (Configuration 2) . 269
6.8 Average current-draw measured using oscilloscope (Configuration 1) . . 271
6.9 Average current-draw measured using oscilloscope (Configuration 2) . . 271
6.10 Amount of energy consumed by T2T-MAP (Configuration 1) ......... 271
6.11 Amount of energy consumed by T2T-MAP (Configuration 2) ......... 272
6.12 Amount of energy consumed by T2T-MAP (Configuration 2) ......... 272
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Wireless IoT attack classification</td>
<td>26</td>
</tr>
<tr>
<td>3.2</td>
<td>ADTree for Wi-Fi fabrication attacks</td>
<td>27</td>
</tr>
<tr>
<td>3.3</td>
<td>ADTree for Wi-Fi interception attacks</td>
<td>30</td>
</tr>
<tr>
<td>3.4</td>
<td>ADTree for Wi-Fi modification attacks</td>
<td>35</td>
</tr>
<tr>
<td>3.5</td>
<td>ADTree for Wi-Fi interruption attacks</td>
<td>36</td>
</tr>
<tr>
<td>3.6</td>
<td>ADTree for Wi-Fi domination attacks</td>
<td>41</td>
</tr>
<tr>
<td>3.7</td>
<td>ADTree for Bluetooth fabrication attacks</td>
<td>47</td>
</tr>
<tr>
<td>3.8</td>
<td>ADTree for Bluetooth interception attacks</td>
<td>49</td>
</tr>
<tr>
<td>3.9</td>
<td>ADTree for Bluetooth interruption attacks</td>
<td>55</td>
</tr>
<tr>
<td>3.10</td>
<td>ADTree for Bluetooth domination attacks</td>
<td>58</td>
</tr>
<tr>
<td>3.11</td>
<td>ADTree for ZigBee fabrication attacks</td>
<td>61</td>
</tr>
<tr>
<td>3.12</td>
<td>ADTree for ZigBee interception attacks</td>
<td>63</td>
</tr>
<tr>
<td>3.13</td>
<td>ADTree for ZigBee interruption attacks</td>
<td>67</td>
</tr>
<tr>
<td>3.14</td>
<td>ADTree for RFID fabrication attacks</td>
<td>72</td>
</tr>
<tr>
<td>3.15</td>
<td>ADTree for RFID interception attacks</td>
<td>76</td>
</tr>
<tr>
<td>3.16</td>
<td>ADTree for RFID interruption attacks</td>
<td>80</td>
</tr>
<tr>
<td>4.1</td>
<td>WPA3-SAE phases</td>
<td>86</td>
</tr>
<tr>
<td>4.2</td>
<td>Simultaneous authentication of equals handshake phases</td>
<td>87</td>
</tr>
<tr>
<td>4.3</td>
<td>Attack environment 1 for Wi-Fi DoS attacks</td>
<td>89</td>
</tr>
<tr>
<td>4.4</td>
<td>MSC: Man-in-the-Middle attack on WPA3-SAE downgrade protection</td>
<td>91</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>4.32</td>
<td>Hardware for PKES’s experiments</td>
<td>167</td>
</tr>
<tr>
<td>4.33</td>
<td>Harmonics of an LF-RFID reader’s carrier</td>
<td>169</td>
</tr>
<tr>
<td>4.34</td>
<td>Carrier of a UHF-RFID reader while applying FHSS</td>
<td>171</td>
</tr>
<tr>
<td>4.35</td>
<td>MSC: FHSS-capable PKES authentication model</td>
<td>174</td>
</tr>
<tr>
<td>5.1</td>
<td>A typical PUF-based authentication protocol</td>
<td>184</td>
</tr>
<tr>
<td>5.2</td>
<td>MSC: Load modification-resistant smart meter authentication protocol</td>
<td>190</td>
</tr>
<tr>
<td>5.3</td>
<td>MSC: Attack on protocol by Boyapally et al.,</td>
<td>192</td>
</tr>
<tr>
<td>5.4</td>
<td>MSC: Vehicle-to-Grid authentication protocol</td>
<td>208</td>
</tr>
<tr>
<td>5.5</td>
<td>MSC: Attack on protocol by Barbareschi et al.,</td>
<td>210</td>
</tr>
<tr>
<td>5.6</td>
<td>MSC: Secure PUF-based Wi-Fi authentication protocol</td>
<td>213</td>
</tr>
<tr>
<td>5.7</td>
<td>Attack on protocol by Mahalat et al.,</td>
<td>214</td>
</tr>
<tr>
<td>5.8</td>
<td>Attack on protocol by Clupek et al.,</td>
<td>217</td>
</tr>
<tr>
<td>6.1</td>
<td>MSC: Enrolment phase of T2T-MAP</td>
<td>227</td>
</tr>
<tr>
<td>6.2</td>
<td>MSC: T2T-MAP authentication (Stage 1 of Configuration 1)</td>
<td>229</td>
</tr>
<tr>
<td>6.3</td>
<td>MSC: T2T-MAP authentication (Stage 2 of Configuration 1)</td>
<td>234</td>
</tr>
<tr>
<td>6.4</td>
<td>MSC: T2T-MAP authentication (Configuration 2)</td>
<td>236</td>
</tr>
<tr>
<td>6.5</td>
<td>Secret key disclosure verification using Tamarin</td>
<td>244</td>
</tr>
<tr>
<td>6.6</td>
<td>ARDUINO-boards wiring for T2T-MAP</td>
<td>249</td>
</tr>
<tr>
<td>6.7</td>
<td>Execution time for T2T-MAP on ARDUINO MEGA (Configuration 1)</td>
<td>253</td>
</tr>
<tr>
<td>6.8</td>
<td>Execution time for T2T-MAP on ARDUINO DUE (Configuration 1)</td>
<td>254</td>
</tr>
<tr>
<td>6.9</td>
<td>Execution time for T2T-MAP on ARDUINO MEGA (Configuration 2)</td>
<td>256</td>
</tr>
<tr>
<td>6.10</td>
<td>Execution time for T2T-MAP on ARDUINO DUE (Configuration 2)</td>
<td>256</td>
</tr>
<tr>
<td>6.11</td>
<td>Total amount of bytes exchanged in T2T-MAP (Configuration 1)</td>
<td>260</td>
</tr>
<tr>
<td>6.12</td>
<td>Total amount of bytes exchanged in T2T-MAP (Configuration 2)</td>
<td>261</td>
</tr>
<tr>
<td>6.13</td>
<td>Oscilloscope for measuring the current-draw per board (Configuration 1)</td>
<td>263</td>
</tr>
<tr>
<td>6.14</td>
<td>Oscilloscope for measuring the current-draw per board (Configuration 2)</td>
<td>264</td>
</tr>
</tbody>
</table>
6.15 Electrical current drawn during T2T-MAP execution on ARDUINO MEGA
        boards (8-byte Configuration 1) ........................................ 265
6.16 Electrical current drawn during T2T-MAP execution on ARDUINO MEGA
        boards (16-byte Configuration 1) ........................................ 265
6.17 Electrical current drawn during T2T-MAP execution on ARDUINO MEGA
        boards (32-byte Configuration 1) ........................................ 266
6.18 Electrical current drawn during T2T-MAP execution on ARDUINO DUE
        boards (8-byte Configuration 1) ........................................ 266
6.19 Electrical current drawn during T2T-MAP execution on ARDUINO DUE
        boards (16-byte Configuration 1) ........................................ 266
6.20 Electrical current drawn during T2T-MAP execution on ARDUINO DUE
        boards (32-byte Configuration 1) ........................................ 267
6.21 Electrical current drawn during T2T-MAP execution on ARDUINO MEGA
        boards (8-byte Configuration 2) ........................................ 267
6.22 Electrical current drawn during T2T-MAP execution on ARDUINO MEGA
        boards (16-byte Configuration 2) ........................................ 267
6.23 Electrical current drawn during T2T-MAP execution on ARDUINO MEGA
        boards (32-byte Configuration 2) ........................................ 268
6.24 Electrical current drawn during T2T-MAP execution on ARDUINO DUE
        boards (8-byte Configuration 2) ........................................ 268
6.25 Electrical current drawn during T2T-MAP execution on ARDUINO DUE
        boards (16-byte Configuration 2) ........................................ 268
6.26 Electrical current drawn during T2T-MAP execution on ARDUINO DUE
        boards (32-byte Configuration 2) ........................................ 269
Chapter 1

Introduction

1.1 Internet of Things

IoT (Internet of Things) is a networking paradigm that aims to extend the Internet by connecting an exponentially increasing number of heterogeneous devices, called Things\(^1\), to the existing infrastructure of Internet. According to IoT-Analytics, the number of connected Things is supposed to reach 11.6 billion in 2021, 15.8 billion in 2023, and 21.5 billion by 2025 [Iot18]. These Things can have multiple functionalities in addition to communication, such as sensing, collecting and aggregating data, processing data, making-decisions, and performing actions through actuators. They are primarily used to implement various private and public smart applications in different domains, such as smart healthcare [Git20c], smart farming [Git20b], smart homes [Git20d], smart cities [Git20a], smart wearable [Git20e], and smart grids [BS20]. These applications have remarkably increased the comfort, productivity, and quality of live for many individuals and organizations worldwide.

As per the physical structure of IoT, it is perceived as a networking infrastructure that is built by interconnecting geographically distributed wireless and wired telecommunication networks to the Internet. Each of these networks, connects and controls a certain number of different devices, i.e., Things, that can be accessed either through the Internet from another

\(^1\)Note that this term (Thing) is a very general term. However, in the context of IoT, it refers to physical entities that did not use to be computers in the past and have recently been computerized (e.g., thermostats, lightbulbs, and fish tanks). It also includes classical computers that were used in the Internet.
network that is located miles away, e.g., via a cloud service, or locally via a short-range wireless technology, such as Wi-Fi and Bluetooth. Furthermore, being a diversified evolution of the classical Internet infrastructure, IoT expands the Internet from a Machine-to-Machine (M2M) communication system to a Things-to-Machine (T2M) and Things-to-Things (T2T) communication system. In other words, IoT aims to transform everything into a computer that can perform computation and communication over a network [Sch18].

1.2 IoT Security

IoT brings to our ecosystem a considerable improvement in terms of social development, economic benefits, and governmental activities. However, on the downside, it invites cybercriminals to exploit the infrastructure of IoT to conduct large scale, distributed, and devastating cyberattacks that may have serious consequences. In fact, IoT connects an exponentially increasing number of heterogeneous devices (i.e., Things) for which security configurations, as well as, security design principles, are unknown. Also, the security mechanisms that IoT employs are similar to those employed by Internet. This makes IoT inherit from Internet all known security issues. Hence, if those devices (Things) are not secure, they can be compromised and hijacked by cyberattackers to make them their cybersoldiers, also known as bots or zombie devices. Attackers may use the compromised devices to build large botnets (e.g., Mirai, Hajime, and Brickerbot) and exploit them to generate attacks that affect the security of information systems at a large scale, generally in the form of a DDoS (Distributed Denial of Service) attack. Examples of such attacks are the Dyn attack in 2016 [The16] and the Brickerbot attack in 2017 [Ble17]. Moreover, attackers can conduct local scale attacks on individual critical devices that may involve human life, such as the Stuxnet attack in 2011 [New11], the Ukraine’s power-grid blackout in late 2015 [Zet16], the Jeep Cherokee attack in 2015 [The15], the finns chilling attack in 2016 [The18], and
the Philips lightbulbs attack demonstration in 2018 [RSW+18]. These attacks have demonstrated how catastrophic and diversified cybercrimes could be in a world where everything is a computer.

Currently, the security of the IoT strongly relies on the security mechanisms that are provided by the existing network infrastructures, protocols, and communication technologies. In fact, as mentioned earlier, IoT is just an extension of an existing “insecure” infrastructure (Internet) with more heterogeneous devices and various communication protocols where security has been left as a question mark. What is more frustrating is that the number of these connected devices is exponentially increasing every year [Iot18], creating more and more security holes in the entire infrastructure.

In this thesis, we limit our research to study the security of wireless infrastructures of IoT, in particular, those adopting short-range technologies and operating resource-constrained devices, with a focus on improving its security and designing a new authentication protocol that conforms to IoT security and performance requirements.

1.3 Motivation and Research Objectives

In this section, we discuss our motivation for focusing on the security of wireless IoT infrastructures and IoT authentication. We also highlight important research objectives.

1.3.1 IoT Wireless Infrastructures Security

The wireless infrastructure of IoT is considerably growing every year and is currently overwhelming the wired part of IoT. At the same time, this wireless infrastructure is thought to be the most vulnerable part of IoT. In fact, the security-level of the wireless devices that compose the wireless infrastructure remains obscure to end users. There is no guarantee that a user’s device will not be hijacked to conduct a cyberattack (e.g., [The16]) or will not be attacked to cause harm to the user (e.g., [The15]). Most of these devices are believed to
be not secure at all [Sch18; LZ20a].

*Hence, the security of billions of IoT devices is left as a question mark and needs to be investigated before further devices are manufactured and connected to IoT.*

Although the wireless infrastructure of IoT involves different network standards and technologies, studies have shown that the largest category of IoT devices adopts personal and local area networks technologies, such as Wi-Fi, Bluetooth, ZigBee, Z-Wave, and RFID [Emb19; Iot18]. Understanding the security issues of these technologies would considerably help security-engineers to better enforce security in IoT, as well as, design future security solutions. Attackers are well informed of probable attacks and existing vulnerabilities on these technologies. Thus, the same attacks are likely to reoccur with an IoT flavor. Also, new attacks on new IoT technologies generally follow the same attack methodology of the attacks that occurred on the old ones.

*Hence, having a complete and thorough understanding of these wireless communication technologies, their security mechanisms, and their related attacks would help in designing and managing a better security for future IoT technologies.*

In the first part of this thesis, we focus on the security of IoT wireless infrastructures, in general, and the security of the most used short-range IoT wireless technologies – Wi-Fi, Bluetooth, ZigBee, and RFID, in particular. Also, we study the security of some IoT devices (with an intention of discovering vulnerabilities) that adopt these technologies.

### 1.3.2 Authentication Issue in IoT

If we carefully analyze the most recent IoT cyberattacks, we can easily realize that these cyberattacks are due to vulnerabilities in the adopted authentication protocols. Indeed, as authentication constitutes the spine of all security properties, any vulnerability residing on the adopted authentication protocol will be exploited to generate security attacks that may breach system’s security services. *Thus, authentication is considered the first step toward the prevention of any cyberattack* [LAY20].
Nevertheless, due to the heterogeneity and the particular security requirements of IoT, developing secure, low-cost, and lightweight authentication protocols has become a serious challenge. This has excited the research community to design and develop new authentication protocols that meet IoT requirements. In fact, ten years after the emergence of IoT (i.e., early 2010), a large number of authentication protocols, known as lightweight protocols (or ultra-lightweight), were designed and proposed for various IoT wired or wireless infrastructures and IoT applications. These protocols relied on the application of classical cryptographic mechanisms, where some of them adopted ECC (Elliptic Curve Cryptography), whereas others employed lightweight cryptographic operations, such as simple hash functions and bitwise logical operators. However, these protocols suffered from known security and/or performance issues that are related to classical cryptographic mechanisms. This has pushed the research community to seek for new mechanisms that better conform to IoT requirements. Also, due to the incorrect usage of certain security tools in classical cryptography (e.g., Xor operator and nonces), many researchers have demonstration known attacks on these authentication protocols but with an IoT flavor. Moreover, only a very small number of these authentication protocols can be used in Thing-to-Thing (T2T) architectures, where Things authenticate each other without involving any human intervention.

Therefore, the need to design secure T2T authentication protocols that conform to IoT security and performance requirements has become one of the top priorities.

Interestingly, during the past five years, there has been a remarkable attraction and convergence from the research community and the industry to adopt PUFs (Physical Unclonable Functions) as a prominent physical security technology. Important industrial cores, such as NXP, Microsemi, Intel, and Xilinx, have already implemented the technology to develop secure integrated circuits (ICs) [NXP18; Mic16; Int15; Cis16]. In the meantime, researchers have turned their attention to PUF technology to develop lightweight and secure-by-design authentication protocols for IoT.
1.4. CONTRIBUTIONS

Hence, PUFs are considered as a novel, reliable, and prominent physical security technology to develop secure ICs and lightweight authentication protocols for IoT.

We devote the second part of this thesis to study PUF-based authentication protocols for IoT, in general, and for wireless IoT, in particular. We aim to investigate the security of exiting PUF-based authentication protocols for IoT to design and implement a better authentication protocol using PUFs for IoT. The developed protocol would be lightweight, secure, and efficient, and can be used for T2T architectures.

1.4 Contributions

In light of improving the security of wireless IoT infrastructures, we claim that the originality of this thesis lies in proposing a simple but concise attack classification, discovering new security vulnerabilities in short-range wireless IoT authentication protocols, demonstrating the feasibility of new attacks with a focus on breaching the security service of availability, proposing new countermeasures for critical attacks, and proposing a new authentication protocol for IoT using new authentication concepts. Specifically, the main contributions of this thesis are as follows:

1. We develop an attack taxonomy to classify attacks on Wi-Fi, Bluetooth, ZigBee, and RFID. We adopt the classification to review attacks on these technologies and discuss possible countermeasures to mitigate the attacks [LZ20a].

2. We perform security case studies on the considered technologies, specifically, on Wi-Fi and Bluetooth technologies. We exploit security vulnerabilities in the authentication protocols of the technologies and demonstrate the feasibility of various DoS (Denial of Service) attacks. Also, we propose countermeasures to fix the discovered vulnerabilities [LZ20b; LZ19c; LZ19b; LZ19a; LZ18].

3. We study the application of FHSS (Frequency Hopping Spread Spectrum) in RFID
1.5. THESIS ORGANIZATION

We divide this thesis into two parts. The first part concerns security of the most used IoT short-range technologies, in particular, Wi-Fi, Bluetooth, ZigBee, and RFID. It includes Chapter 2, 3, and 4. This part is based on the publications [LZ18; LZ19a; LZ19c; LZ19b; LZ20b; LZ20c; LZ20a]. The second part studies recent PUF-based authentication protocols and develops a novel lightweight authentication protocol using PUFs. The part includes Chapter 5 and 6. This second part is based on the papers [LZ20d; LZ20e]. The author of this thesis also co-wrote the papers [WLZ19a; WLZ19b; LO20] while developing this Ph.D. These papers are not particularly within the scope of this thesis, thus not included in it.

Chapter 2 reviews the most used IoT short-range wireless technologies. It presents Wi-Fi, Bluetooth, ZigBee, and RFID, along with their respective security mechanisms. This
1.5. THESIS ORGANIZATION

Chapter provides the reader with the necessary background to understand the research discussed in Chapter 3 and 4.

Chapter 3 develops a service-based attack taxonomy to classify attacks in IoT. The classification allows a simple but concise categorization of attacks that will help security-engineers to better visualize and filter attacks of ones concern. We adopt the taxonomy to review the attacks that occurred during the two decades on aforementioned technologies. It also discusses their possible attacks’ countermeasures.

Chapter 4 aims to discover new security vulnerabilities on current authentication protocols that the Wi-Fi and Bluetooth technologies adopt and demonstrate new attacks with a focus on breaching the availability of these two technologies. Also, we propose countermeasures to mitigate the attacks. Finally, the chapter proposes a fundamental countermeasure to counter relay attacks on PKESs (Passive Keyless Entry and Start Systems).

Chapter 5 reviews and analyzes the security of recent PUF-based authentication protocols for IoT. We demonstrate possible attacks on the reviewed protocols and draw security lessons to consider for future PUF-based authentication protocols.

Chapter 6 designs and implements a lightweight PUF-based mutual authentication protocol for IoT. We evaluate the security as well as the performance of the protocol and show that the developed protocol is secure to known attacks and efficient in terms of execution time, communication overhead, and energy consumption.

Finally, Chapter 7 summarizes the thesis along with a brief discussion on its limitations and future work.
Part I

Security of IoT Short-Range Wireless Technologies
Chapter 2

IoT Short-Range Wireless Technologies

In this chapter, we briefly review the most adopted short-range wireless communication technologies in IoT. We consider four wireless communication technologies, namely, Wi-Fi, Bluetooth, ZigBee, and RFID. These wireless communication technologies have proven to be most established and prominent technologies for the future of IoT.

2.1 Introduction

Although the wireless infrastructure of IoT involves different network standards and technologies, studies have shown that the largest category of IoT wireless devices (Things) adopts personal (WPAN) and local (WLAN) area networks technologies, such as Wi-Fi, Bluetooth, ZigBee, Z-Wave, and RFID [Lin15; Cog17; A R18; AAA+17; Let18; Emb19; Iot18]. Also, according to IoTAnalytics\(^1\) the number of IoT connected devices (Things) that adopt these technologies is exponentially increasing every year. Thence, we turn our attention to the short-range wireless networks of IoT. More precisely, we consider reviewing the security of the most used short-range wireless technologies in IoT. We concretely consider four wireless technologies, namely, Wi-Fi, Bluetooth, ZigBee, and RFID technology.

2.2 Wi-Fi Technology

2.2.1 Wi-Fi Overview

Wi-Fi (Wireless Fidelity) is a wireless technology based on the IEEE 802.11 standard. It allows the construction of WLANs (Wireless Local Area Networks) over both unlicensed radio bands, the 2.4 GHz ISM (Industrial Scientific and Medical) band and the 5 GHz UNII (Unlicensed National Information Infrastructure) band. It was first introduced in 1999 allowing the implementation of WLANs over a short-range with a basic transmission rate of 2Mbps. Later, Wi-Fi has significantly evolved in many aspects, such as power management, quality of service, data rate, infrastructure modes, and security. Nowadays, a Wi-Fi network can send data at 6.75Gbps [IEE13] and reach a range up to 382km [Pie07]. It is commonly used in domestic places, such as houses, hospitals, universities, and enterprises.

2.2.2 Wi-Fi Security

Wi-Fi technology provides a number of security mechanisms. In the following paragraphs, we briefly present these mechanisms. Interested readers are referred to the IEEE 802.11 specification documents [Wi-18a] for more details.

**WEP (Wired Equivalent Privacy).** WEP provides authentication through the use of a shared password in a challenge-response scheme. It applies the RC4 (Ron’s Code 4) symmetric stream-cipher along with an encryption key to provide confidentiality. Data integrity is provided by generating an integrity code using CRC-32 (Cyclic Redundancy Check, 32 bits) algorithm.

**WPA (Wi-Fi Protected Access).** WPA is based on a draft version of the IEEE 802.11i standard [IEE04]. It provides two operational modes, WPA-PSK (Pre-Shared Key) and WPA-Enterprise 3. It applies TKIP (Temporal Key Integrity Protocol) to provide encryption, which includes the use of RC4. Data integrity is guaranteed by the generation of a message integrity code (MIC) using the Michael algorithm. The authentication of
the users (a.k.a., supplicants) occurs during a key-establishment protocol, known as the 4-way-handshake.

**WPA2 (Wi-Fi Protected Access 2).** WPA2 is an implementation of the IEEE 802.11i standard [IEE04]. It provides the same operational modes as WPA and the authentication of the supplicants occurs in the same way. However, it applies CCMP (CTR with CBC-MAC Protocol) to provide encryption using AES\(^2\). Also, data integrity is provided by the generation of a message integrity code using the CBC-MAC (Cipher Block Chaining-Message Authentication Code) algorithm.

**WPS (Wi-Fi Protected Setup).** This security mechanism was introduced by the Wi-Fi Alliance in 2006 to provide an easy and secure procedure to join a Wi-Fi network through the use of a PIN code (to be inserted on Wi-Fi devices), simultaneous button pushing (on authenticating devices), near field communication (NFC), or by transferring authentication information using a USB pendrive.

**Opportunistic Wireless Encryption.** This mechanism is defined in the RFC810. It allows a Wi-Fi client and an access point to establish a shared secret key without having shared any credentials a priori. It aims to add a security layer for Wi-Fi networks that adopt the open system authentication such as public and guest networks. It uses the Diffie-Hellman key establishment protocol [DH76] to establish keys to guarantee message authentication, confidentiality, and integrity.

**MFP (Management Frame Protection).** This mechanism is part of the IEEE 802.11w amendment (2009). It aims to increase the security of Wi-Fi management frames by providing data confidentiality, integrity, authenticity, and freshness.

**WPA3 (Wi-Fi Protected Access 3).** WPA3 is a certification that augments WPA2 with additional security measures which include: the resistance against dictionary attacks

\(^2\)AES (Advanced Encryption Standard), also known as Rijndael, is a symmetric cipher established by the U.S. National Institute of Standards and Technology (NIST) in 2001 [NIS01].
through the use of a new key-establishment scheme, called SAE (Simultaneous Authentication of Equals) [Har08], which uses ECC (Elliptic Curve Cryptography); management frame anti-spoofing through MFP (Management Frame Protection); countermeasures against side-channel attacks; and forward secrecy. The encryption and data integrity occur in the same way as in WPA2. It is more secure than WPA2 but it is still under implementation [Wi-18b].

2.3 Bluetooth Technology

2.3.1 Bluetooth Overview

Bluetooth is a wireless technology based on the IEEE 802.15.1 standard. It is used for exchanging data between fixed and mobile wireless devices within a short-range and building WPANs (Wireless Personal Area Networks). It was originally conceived by the telecommunications vendor Ericsson in 1994, as a wireless alternative to RS-232 cables. It uses the free unlicensed 2.4 GHz ISM (Industrial, Scientific, and Medical) radio band and adopts the FHSS (Frequency Hopping Spread Spectrum) transmission technique to send packets while reducing interference. It employs the master-slave communication mode. Bluetooth has evolved for the last twenty years, from v1.0 (1999) to Bluetooth 5.2 (2019), coming out with better power consumption, stronger security, higher data rate, and longer range.

In 2010, the Bluetooth SIG (Special Interest Group) released Bluetooth v4.0+LE (Low Energy), or Bluetooth Low Energy, or simply BLE [Blu18a]. This new wireless technology includes two sub-specifications: Bluetooth smart, also known as BLE single mode; and the Bluetooth smart ready, also known as BLE dual mode. These two sub-specifications have completely different physical\(^3\) and link layers, which result in two different protocol stacks: BLE dual mode and BLE single mode. BLE dual mode implements the classic Bluetooth stack, which is used in Bluetooth v1.1 to Bluetooth v3.0+HS (High Speed), as

\(^3\)Classical Bluetooth applies FHSS (Frequency Hopping Spread Spectrum) by hopping over 79 channels at 1600 hops per second, whereas BLE applies FHSS over 40 channels at the same hopping rate. Also, the hopping sequence is easy to generate in BLE than in classical Bluetooth.
well as the Bluetooth smart stack. BLE single mode implements only the Bluetooth smart stack. Bluetooth devices operating over the single mode stack are not compatible with classic Bluetooth devices [Blu18a].

2.3.2 Bluetooth Security

Bluetooth technology provides security services through different security mechanisms. In the following paragraphs, we present the commonly used ones.

**Non-Discoverable Mode.** Bluetooth allows devices to be set to a private mode known as non-discoverable mode. This mode hides the presence of a device from other Bluetooth devices. A device in this mode can only be reached out by other Bluetooth devices that know its Bluetooth device address.

**Pairing Mechanism.** Pairing allows two Bluetooth devices to authenticate each other and negotiate on a set of security parameters to derive a master key called link key. This link key is employed further to derive other keys that will be used to guarantee secure communications. Currently, there are three pairing mechanisms: the legacy pairing used in Bluetooth v1.0 to v2.0+EDR (Enhanced Data Rate), the SSP (Secure Simple Pairing) used in v2.1+EDR (Enhanced Data Rate) to v4.1+LE (Low Energy), and the Secure Connections used in Bluetooth v4.2+LE to 5.2:

**Confidentiality Mechanism.** Bluetooth has three encryption modes, but only two of them provide confidentiality: encryption mode 1, in which there is no encryption; encryption mode 2, in which only unicast traffic is encrypted; and encryption mode 3, which encrypts all the traffic. From Bluetooth v1.0 to Bluetooth v4.1+LE, the encryption is based on SAFER+, while Bluetooth v4.2+LE to 5.2 uses AES.

**Security Modes.** Bluetooth security modes define when and where security procedures such as authentication and encryption shall be initiated. There are four different security modes [Blu18b], security mode 1, 2, 3 and 4. Security mode 1, also known as unprotected
2.4 ZigBee Technology

2.4.1 ZigBee Overview

ZigBee is a wireless technology based on the IEEE 802.15.4 standard. It allows resources-constrained devices, such as power-limited devices, to communicate over the radio and form a WPAN (Wireless Personal Area Network). It is the most used wireless technology in home automation and smart lighting [Mil11]. It was initially conceived in 1998 and then standardized in 2003. ZigBee has evolved in the last sixteen years, from the first version of ZigBee (2004) to ZigBee 3.0 (2016) [Tex18], coming out with better power consumption, flexibility, inexpensive deployment [All18], security, and new network topology options [Mil11]. ZigBee uses 2.4 GHz ISM band but can also operate on other bands [Gis08]. It allows a nominal range of 10 to 100 meters with a data rate varying from 20Kbps to 250Kbps [Aft17; All18].

2.4.2 ZigBee Security

ZigBee technology has the following five main concepts of security [VHP+13]:

**Trust Center.** In a ZigBee network, the device that has more physical resources than all other devices is called the network coordinator or trust center. All other devices are called nodes. The network coordinator is assigned the responsibility of managing the security of the whole network by creating and managing three types of security keys: master, link, and network. The master key is used for securely exchanging other secret keys. Link keys are
per-link keys used to encrypt messages that are exchanged between two nodes. Finally, the network key is used by new nodes joining the network.

**Authentication and Encryption.** In ZigBee, the data is encrypted using 128-bit AES algorithm with CCM* mode. This operational mode is a slightly modified version of CBC-MAC (Counter with Cipher Block Chaining Message Authentication Code), allowing authentication and encryption [Zig08].

**Data Integrity and Freshness.** ZigBee uses CCM* algorithm to generate the message integrity code. This code ensures that the data has not been altered while being exchanged. It also uses a 32-bit counter to distinguish between new and old frames for freshness [Zig08].

**Security Levels.** ZigBee provides two different security levels, namely, high security (commercial security) and the standard security (residential security). The key difference between these two levels resides in how the cryptographic keys, such as the network key, are managed and distributed over the network. The high security provides key confidentiality by allowing the network controller to send the network key in an encrypted format. However, in the standard security level, the network key is sent unencrypted. This makes it easy to be eavesdropped and learned by an attacker as demonstrated in [VHP+13]. Meanwhile, the high security mode entropy relies on a pre-installed master key, which is shared among all devices. Therefore, the compromise of one single node jeopardizes the entire network.

**Key Management.** In a ZigBee network, security keys are distributed in three different ways. The first way (in the high-security level) consists of transmitting the keys, such as the network key, in its encrypted form. The second way (in the standard security level) consists of transmitting the network key unencrypted. The last mode, which is a trade-off between usability and security, consists of manually installing the keys, such as the network key, onto each legitimate device.
2.5 RFID Technology

2.5.1 RFID Overview

RFID (Radio Frequency IDentification) is a wireless technology designed for automatically identifying, tracking, and collecting data from entities such as objects, humans and animals. It relies on tagging objects to identify, track, and collect data from them using the concept of tag-reader. An RFID-tag or transponder is the unit (e.g., microchip implant) that stores identification information used to identify and track the object carrying the tag. It is mainly composed of a chip for storage and computation, optional battery for power supply, and an antenna for communication [EFG+11]. The RFID-reader however, is a mobile or fixed device that wirelessly interrogates RFID-tags for object identification and tracking. A third party known as backend database is sometimes used in RFID systems. This backend database is requested by the RFID-reader for each tag identification through secure channels.

2.5.2 RFID Security

As RFID-tags can be attached to any object or implanted in any living-being, the possibility of reading personally-linked information without consent has raised serious privacy concern. This concern resulted in the development of many security mechanisms to provide security properties as discussed in the following paragraphs.

Encryption and Key Management. Most RFID applications use symmetric cryptography, e.g., 3DES in ePassports. In such cases, a key management scheme is used. This is because tags and readers share a secret key that is tag-specific, and none of the two parties can start identifying itself to the other party. In fact, on the one hand, if the tag starts identifying itself by sending its identity in plaintext, all other readers operating on the same frequency can read that identity and trace that tag. On the other hand, if the reader starts authenticating itself to the tag, it does not know which secret key to use since it does not know which tag it is interrogating. Some schemes have been proposed in the literature to
2.5. RFID TECHNOLOGY

solve this paradox [Jue04; Ohk04; HM04; MW04; WSR+03; Dim05].

Authentication. To provide authentication, RFID applies challenge-response-based authentication protocols. Initially, a symmetric key is shared between the tag and reader. The tag (a.k.a., prover) proves to the reader (also known as verifier) the right possession of the key without revealing it. This consists of sending a challenge from the reader to the tag. The latter performs some cryptographic operations using the shared key to produce a response and sends it back to the reader. The reader performs slightly the same cryptographic operations using the shared key to check whether the results of its computations are equal to the ones received from the tag. If the results are similar, the reader authenticates the tag. If a mutual authentication is required, the protocol runs in reverse. Existing authentication protocols are based on symmetric cryptography [FWR05; Pir06; AF05]. Asymmetric cryptography however, is less frequently adopted [GJR07; Wol05; A J96].

Distance-Bounding Protocol. The distance-bounding protocol is a lightweight authentication protocol that in addition to checking that one communication party (e.g., tag or reader) possesses the correct secret key, checks whether the distance between the reader and tag is below a given threshold [BC93]. This distance is measured by either the signal strength RSSI (Receiving Signal Strength Indicator) [FR03] or the RTT (Round Trip Time) that takes for an RFID-reader to send a challenge and receive its response from an RFID-tag. Conceptually, a distance-bounding protocol runs in three phases: (1) The slow phase, also known as initial or setup phase, where the tag and reader agree on session parameters, such as nonces. (2) The fast phase, also known as distance-bounding phase, timed phase, or critical phase, where challenge-response rounds occur and the RFID-reader measures the round trips. (3) The verification phase, also known as final signature or authentication phase, where the reader ensures that the fast phase was executed faithfully so that it can use the RTT to calculate the distance. This is done by checking the correctness of all round-trip times and the RFID-tag’s proof of knowledge of a valid signature.
2.6 Conclusion

In this chapter, we have reviewed the most used wireless technologies in IoT, namely, Wi-Fi, Bluetooth, ZigBee, and RFID. We have given an overview about each technology and presented the security mechanisms that each technology provides to enforce security services.

Although the existing security mechanisms, in general, and authentication mechanisms, in particular, that the reviewed wireless technologies provide have a certain level of security, we believe that the application of these mechanisms will not last for too long. In fact, as IoT is rapidly transforming the Internet into a thing-to-thing communication system, the need for new authentication protocols, mainly T2T authentication protocols, is rising.

In the next chapter, we propose an attack classification for IoT. We adopt the classification to review the attacks that occurred in the last two decades on the considered wireless communication technologies – Wi-Fi, Bluetooth, ZigBee, and RFID.
Chapter 3

Taxonomy and Analysis of IoT Wireless Attacks

In this chapter, we develop an attack classification for IoT. The classification allows a simple, but concise categorization of attacks. We adopt the classification to review the attacks that occurred in the last two decades on Wi-Fi, Bluetooth, ZigBee, and RFID. We also provide possible countermeasures that can be applied to mitigate certain attacks.

3.1 Introduction

Wireless networks that adopt short-range wireless technologies constitute the largest and fast-growing part of IoT, but also the most vulnerable part of IoT as they connect billions of insecure devices to the Internet. This creates a wide attack surface for cybercriminals to generate cyberattacks at the large scale, as well as, at the local scale. Thus, it is important to have a clear understanding of possible attacks on these short-range wireless technologies so that new security solutions can be properly designed to mitigate existing attacks. Moreover, classifying attacks has always been a good practice to help security-engineers better understand possible attacks on a given system and easily identify the vulnerabilities to propose countermeasures. Also, by providing a taxonomy, one can easily focus on specific types of attacks rather than having a haphazard long list of attacks that would require additional efforts to filter and understand. In this light, we devote this chapter to develop a security
3.2. TAXONOMY OF ATTACKS

service-based attack classification. The classification allows a simple, but concise categorization of attacks in a user-friendly way. We adopt the classification to review the attacks that occurred in the last twenty years on Wi-Fi, Bluetooth, ZigBee, and RFID technology, and discuss possible countermeasures to mitigate the attacks.

3.2 Taxonomy of Attacks

3.2.1 IoT Security Threats

In general, Internet of Things (IoT) is subject to two types of security threats, namely, accidental threats and intentional threats. Accidental threats represent the set of threats which are not expected and not involving any intentional parties. It principally targets the physical security of IoT, such as fires, earthquakes, floods, pandemics, explosions and landslides, or software security, such as device failures and software bugs. Intentional threats however, also known as attacks, assume the involvement of an intended party, known as attacker, who undertakes a set of illegal actions to cause harm to the IoT infrastructure. If an attack is successfully conducted, we call it an intrusion. Technically, an attack is an intrusion attempt. If an attack has been performed using only information technology utilities, e.g., computers and software, we call it a cyberattack. In this chapter, we limit our scope to consider only outsider\(^1\) attacks.

3.2.2 Fundamental Security Services

In this subsection, we define the fundamental security services, also known as information assurance pillars, defined in the DoD\(^2\) Information Assurance Certification and Accreditation Process [DoD02].

\(^1\)Attacks can be classified either as insider or outsider attacks. Insider attacks are generated by a trusted and authorized party within the network (e.g., dishonest employee), whereas outsider attacks are generated by an unauthorized party from outside the network (e.g., a hacker).

\(^2\)U.S. Department of Defense.
3.2. TAXONOMY OF ATTACKS

Authentication. This service aims to prove that an entity, e.g., an individual, software, or device, is effectively what it claims to be. It is generally set up by proving the possession of a secret (something you know, e.g., password or key), possession of a personal physical item (something you have, e.g., access card), and/or personal features (something you are, e.g., fingerprints, facial, and iris recognition).

Confidentiality. It is also known as secrecy. This service aims to protect the content of the stored and transmitted data from being disclosed to unauthorized parties. It is essentially carried out using encryption or steganography techniques.

Integrity. This service aims to guarantee that the content of stored or transmitted data has not been accidentally or intentionally been modified.

Availability. This service assures that system services and resources are instantly and continuously available for users, when needed.

Non-repudiation. This service prevents any involved party from denying any performed legal or illegal operation (e.g., sent, received, executed, or modified). It is generally provided by the use of digital signatures which are commonly used in asymmetric cryptography.

Furthermore, besides the considered fundamental security services, we do not deny the existence of many other overlapping security services, which include but not limited to, auditability, accountability, authorization, trust, traceability, anonymity, liveness, and synchronization. It is not possible to derive a useful orthogonal classification by considering all the above mentioned security services. As a result, we only consider the fundamental security services that also cover the other security services.

3.2.3 Attack Classification Scheme

Classifying attacks has always been an effective practice to help security-engineers better understand possible attacks on a given information system and easily identify the vulnerabilities to propose countermeasures. Also, by providing a classification of attacks, one can
easily focus on specific types of attacks rather than having a haphazard long list of attacks that would require additional efforts to understand and filter attacks of one’s concern. A large number of attack classifications have been proposed in the literature [IW08]. The most fundamental ones are as follows: the active-passive attack classification proposed by Kent [Ken77], the internal-external attack classification used by McNamara [McN98], the protocol layer-based attack classification introduced by McHugh [McH00], and the attack classification by Stallings [Sta95].

We believe that active-passive [Ken77] and internal-external [McN98] classifications are too broad and much more abstract. The protocol layer-based classification [McH00] classifies attacks according to the protocols being exploited to conduct attacks. This classification becomes hard to apply when different protocol stacks (e.g., due to the heterogeneity of IoT) are being used. Moreover, as some attacks operate at multiple layers, it becomes hard to determine which layer an attack belongs to, which may lead to different opinions regarding the classification of a given attack.

Stallings’ classification [Sta95] appears more general and decisive. Based on the impact of the attack over an asset, attacks are categorized using four general classes, namely, interruption, interception, modification, and fabrication. These classes are described as follows [Bro04]: The interruption class groups attacks where the availability of an asset is disrupted (e.g., server becomes non-responsive). The interception class refers to attacks where an unauthorized access to an asset is made (e.g., intercepting network messages and reading their contents). The modification class groups unauthorized tampering with an asset (e.g., unauthorized modification of a router’s routing table). Finally, the fabrication class concerns attacks where a fictitious asset is created (e.g., cloning an RFID card). This classification can actually be perceived as a security service-based classification that categories attacks according to which security service is compromised, essentially, authentication, confidentiality, integrity, and availability.

Although Stallings’ classification seems more general and decisive, we believe that the
classification needs to be extended to cover certain attacks in a more precise way. For example, based on the descriptions given in [Bro04] for these four general classes, cracking a password is perceived as an interception attack since the attacker has gained an unauthorized access to the asset (i.e., the password). Notwithstanding, we rather perceive this attack as an attack that belongs to a different and much broader category. In fact, nowadays, many security protocols adopt a key establishment protocol as part of the authentication protocol. That is, during the authentication of two communicating parties, the latter use authentication information (e.g., a shared password or keys) to prove their identities to each other (e.g., by exchanging the hash of the password mixed with some nonces) and derive a set of cryptographic keys to be used for securing future communications, i.e., enforcing message confidentiality, integrity, and authenticity. In this case, if an attacker manages to crack the password, the attacker will consequently manage to derive all cryptographic keys that are used to ensure message confidentiality, integrity, and authenticity. In other words, cracking the password is an attack that affects multiple security services at a time. Therefore, we need another class that groups the kind of attacks that can breach multiple security services at a time, regardless of the intention of the attacker.

We enrich the classes of Stallings’ classification with a new class called domination\textsuperscript{3} to group attacks that compromise more than one security service at a time. Hence, the new classification scheme first classifies attacks according to affected wireless communication technology. Second, in each wireless communication technology, attacks are classified into the following five classes:

**Fabrication.** This class includes attacks that aim to impersonate trusted entities in IoT infrastructures to gain certain privileges and perform illegal actions. For example, an attacker spoofs a master entity in a wireless IoT network and orders slave entities to change their functions, causing the breach of other security services.

\textsuperscript{3}This term is taken from the concept of domination in graph theory. It has been used in [BM15] to define a set of vital vulnerabilities that can be exploited to generate different types of attacks.
### 3.2. TAXONOMY OF ATTACKS

**Interception.** This class covers all types of attacks that aim to compromise the confidentiality of IoT wireless infrastructures. For example, an attacker (in this case called eavesdropper) captures wireless traffic over the air and analyzes the traffic to extract confidential and private information.

**Modification.** This class covers all types of attacks that aim to illegally modify the content of messages and stored data. As an example, in multihop-based infrastructures, an attacker may intercept network messages, change their contents, and then relay the messages to IoT nodes that have not yet received those messages.

**Interruption.** This class contains attacks that aim to deny legitimate parties to benefit from a set of services provided by an IoT infrastructure. For example, an attacker may cause a set of nodes in a wireless IoT network to shutdown.

**Domination.** This class comprises attacks that aim to compromise multiple security services at a time. An attack in this class can be a pre-condition for other attacks of other classes. For example, an attacker may crack the password of a Wi-Fi network. This password is then used to generate all cryptographic keys used for authentication (fabrication), encryption (confidentiality), and data integrity (modification). By knowing the network password and impersonating the network access point, the attacker can shutdown specific clients from the network (availability).

We develop the attack classification scheme illustrated in Figure 3.1 to group the attacks that occur in IoT short-range wireless resource-constrained infrastructures with respect to the considered wireless technologies. In the next sections, we review the attacks that occurred in the last two decades on the considered technologies.

Furthermore, we note that other works have studied the classification of vulnerabilities (flaws). For example, the classification by Howard et al., [HL98] classifies vulnerabilities as being design, implementation, or configuration issues; and the classification by Landwehr et al., [LBM+94], which categories flaws into three different criteria: Genesis (the flaw’s origin),
3.3. ATTACKS ON WI-FI TECHNOLOGY

Attacks on Short-Range Wireless Communication Technologies for IoT

<table>
<thead>
<tr>
<th>Technology</th>
<th>Fabrication (Subsection 3.3.1)</th>
<th>Interception (Subsection 3.3.2)</th>
<th>Modification (Subsection 3.3.3)</th>
<th>Interruption (Subsection 3.3.4)</th>
<th>Domination (Subsection 3.3.5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wi-Fi</td>
<td>Fabrication (Subsection 3.3.1)</td>
<td>Interception (Subsection 3.3.2)</td>
<td>Modification (Subsection 3.3.3)</td>
<td>Interruption (Subsection 3.3.4)</td>
<td>Domination (Subsection 3.3.5)</td>
</tr>
<tr>
<td>Bluetooth</td>
<td>Fabrication (Subsection 3.4.1)</td>
<td>Interception (Subsection 3.4.2)</td>
<td>Modification (Subsection 3.4.3)</td>
<td>Interruption (Subsection 3.4.4)</td>
<td>Domination (Subsection 3.4.5)</td>
</tr>
<tr>
<td>ZigBee</td>
<td>Fabrication (Subsection 3.5.1)</td>
<td>Interception (Subsection 3.5.2)</td>
<td>Modification (Subsection 3.5.3)</td>
<td>Interruption (Subsection 3.5.4)</td>
<td>Domination (Subsection 3.5.5)</td>
</tr>
<tr>
<td>RFID</td>
<td>Fabrication (Subsection 3.6.1)</td>
<td>Interception (Subsection 3.6.2)</td>
<td>Modification (Subsection 3.6.3)</td>
<td>Interruption (Subsection 3.6.4)</td>
<td>Domination (Subsection 3.6.5)</td>
</tr>
</tbody>
</table>

Figure 3.1: Wireless IoT attack classification scheme w.r.t. Wi-Fi, Bluetooth, ZigBee, and RFID wireless technologies. The corresponding subsection where a technology and attacks are discussed is indicated in parenthesis.

time of introduction (during development, maintenance, or during system operation), and location (software or hardware). We see this type of classification as a second step to classifying attacks. In fact, after classifying an attack and identifying that it affects a particular security service, e.g., authentication, then we can adopt flaw classification as a second step to determine and locate the vulnerability that was behind the generation of the concerned attack.

3.3 Attacks on Wi-Fi Technology

3.3.1 Fabrication Attacks on Wi-Fi

In the following subsections, we present the attacks that violate the authentication service in Wi-Fi. These attacks are mainly due to a partial implementation of authentication on network traffic or due to some flaws in the authentication protocols. Figure 3.2 illustrates
3.3. ATTACKS ON WI-FI TECHNOLOGY

Entity Spoofing

Identity Spoofing. In this scenario, an attacker spoofs the identity of a Wi-Fi device to impersonate it and gain certain privileges. This can be done by spoofing the MAC address of the target Wi-Fi device, the SSID (i.e., in case of spoofing an access point), or both. This attack is easy to implement since nowadays most Wi-Fi network interfaces support the MAC address changing option as well as the “Master mode” to emulate access points.

Entity Spoofing

Identity Spoofing. In this scenario, an attacker spoofs the identity of a Wi-Fi device to impersonate it and gain certain privileges. This can be done by spoofing the MAC address of the target Wi-Fi device, the SSID (i.e., in case of spoofing an access point), or both. This attack is easy to implement since nowadays most Wi-Fi network interfaces support the MAC address changing option as well as the “Master mode” to emulate access points.

Entity Spoofing

Identity Spoofing. In this scenario, an attacker spoofs the identity of a Wi-Fi device to impersonate it and gain certain privileges. This can be done by spoofing the MAC address of the target Wi-Fi device, the SSID (i.e., in case of spoofing an access point), or both. This attack is easy to implement since nowadays most Wi-Fi network interfaces support the MAC address changing option as well as the “Master mode” to emulate access points.

Entity Spoofing

Identity Spoofing. In this scenario, an attacker spoofs the identity of a Wi-Fi device to impersonate it and gain certain privileges. This can be done by spoofing the MAC address of the target Wi-Fi device, the SSID (i.e., in case of spoofing an access point), or both. This attack is easy to implement since nowadays most Wi-Fi network interfaces support the MAC address changing option as well as the “Master mode” to emulate access points.

Entity Spoofing

Identity Spoofing. In this scenario, an attacker spoofs the identity of a Wi-Fi device to impersonate it and gain certain privileges. This can be done by spoofing the MAC address of the target Wi-Fi device, the SSID (i.e., in case of spoofing an access point), or both. This attack is easy to implement since nowadays most Wi-Fi network interfaces support the MAC address changing option as well as the “Master mode” to emulate access points.

Entity Spoofing

Identity Spoofing. In this scenario, an attacker spoofs the identity of a Wi-Fi device to impersonate it and gain certain privileges. This can be done by spoofing the MAC address of the target Wi-Fi device, the SSID (i.e., in case of spoofing an access point), or both. This attack is easy to implement since nowadays most Wi-Fi network interfaces support the MAC address changing option as well as the “Master mode” to emulate access points.

Entity Spoofing

Identity Spoofing. In this scenario, an attacker spoofs the identity of a Wi-Fi device to impersonate it and gain certain privileges. This can be done by spoofing the MAC address of the target Wi-Fi device, the SSID (i.e., in case of spoofing an access point), or both. This attack is easy to implement since nowadays most Wi-Fi network interfaces support the MAC address changing option as well as the “Master mode” to emulate access points.

Entity Spoofing

Identity Spoofing. In this scenario, an attacker spoofs the identity of a Wi-Fi device to impersonate it and gain certain privileges. This can be done by spoofing the MAC address of the target Wi-Fi device, the SSID (i.e., in case of spoofing an access point), or both. This attack is easy to implement since nowadays most Wi-Fi network interfaces support the MAC address changing option as well as the “Master mode” to emulate access points.

Entity Spoofing

Identity Spoofing. In this scenario, an attacker spoofs the identity of a Wi-Fi device to impersonate it and gain certain privileges. This can be done by spoofing the MAC address of the target Wi-Fi device, the SSID (i.e., in case of spoofing an access point), or both. This attack is easy to implement since nowadays most Wi-Fi network interfaces support the MAC address changing option as well as the “Master mode” to emulate access points.

Entity Spoofing

Identity Spoofing. In this scenario, an attacker spoofs the identity of a Wi-Fi device to impersonate it and gain certain privileges. This can be done by spoofing the MAC address of the target Wi-Fi device, the SSID (i.e., in case of spoofing an access point), or both. This attack is easy to implement since nowadays most Wi-Fi network interfaces support the MAC address changing option as well as the “Master mode” to emulate access points.

Entity Spoofing

Identity Spoofing. In this scenario, an attacker spoofs the identity of a Wi-Fi device to impersonate it and gain certain privileges. This can be done by spoofing the MAC address of the target Wi-Fi device, the SSID (i.e., in case of spoofing an access point), or both. This attack is easy to implement since nowadays most Wi-Fi network interfaces support the MAC address changing option as well as the “Master mode” to emulate access points.
3.3. ATTACKS ON WI-FI TECHNOLOGY

**Countermeasure.** Although it is not that easy to mitigate spoofing, detecting such activity is rather possible. Using wireless intrusion detection systems [Ker08], it is possible to detect the presence of two identical devices operating in the network [LBT11]. For instance, a spoofing access point can be localized by analyzing synchronization frames\(^5\) generated by access points and detecting the presence of frames carrying the same BSSID and SSID, but with different timestamps.

**Message Spoofing**

**Packet Forging.** As authentication is not available in Wi-Fi management and control frames, attackers can easily forge them. In most cases, the attacker creates a frame and indicates the source address as the address of a device which has higher privileges, such as the access point. The devices which receive those forged frames accept them and process them as if they were sent from the true source.

**Countermeasure.** Packet forging can be mitigated by requiring authentication on all types of Wi-Fi frames. For instance, every connected device should be able to verify whether a received frame is coming from a legitimate source or not. To that end, Wi-Fi devices can employ MFP (Management Frames Protection) mechanism, which is optional in WPA and WPA2, but mandatory in WPA3. An alternative consists of using WPA-Enterprise with X509 digital certificates.

**WEP Packet Replay.** Technically, WEP mechanism does not guarantee data freshness. This allows an attacker to capture previously exchanged WEP packets and replay them later on to gain some privileges. For instance, if the attacker captures the challenge-response messages during a previous WEP authentication, the attacker can infer the used keystream. By knowing the IV (Initialization Vector) that was used to generate the keystream, the

\(^5\)Wi-Fi frames are network packets generated at the MAC layer. Synchronization frames are commonly known as beacons. They are periodically broadcasted by access points to indicate their presence in the neighborhood and disseminate information about the Wi-Fi network.
attacker runs multiple association attempts until the access point asks for a response which uses that known IV. In this case, the attacker responds correctly to the challenge and gets successfully authenticated.

**Countermeasure.** When data freshness is correctly implemented in an authentication protocol, an attacker will not be able to replay old messages. This countermeasure has been implemented in WPA and WPA2, which aim to replace WEP. Although WPA and WPA2 are relatively more secure than WEP, it is highly recommended to switch to WPA3, which is more secure than WEP, WPA, and WPA2 mechanisms.

### 3.3.2 Interception Attacks on Wi-Fi

Wi-Fi networks have been demonstrated to be vulnerable to interception attacks [JS04; TB09; N A13; BGW01; K G15]. This is fundamentally related to the broadcast nature of the wireless medium along with the implementation flaws discovered in the adopted encryption mechanisms, e.g., RC4. In the following subsections, we review the most known interception attacks on Wi-Fi. Figure 3.3 illustrates an attack-defense tree for attacking a Wi-Fi IoT infrastructure through interception attacks.

**Reconnaissance**

**Sniffing and Packet Analysis.** Wi-Fi allows the use of a non-secure mode called open mode. In this mode, no confidentiality is provided and all Wi-Fi frames are sent unencrypted over the radio channel. An attacker can easily capture a number of Wi-Fi frames to analyze them and extract sensitive information such as credentials and private information.

**Countermeasure.** The most obvious security initiative that can be adopted to mitigate this attack is to use any of the encryption mechanisms provided by either WEP or WPA. Yet, in some circumstances, certain Wi-Fi networks are intentionally left open for user flexibility, such as the ones provided in supermarkets, large retail shops, or even airports.
In such networks, security has to be implemented in the upper layers to use upper-layer security protocols, e.g., TLS (Transport Layer Security). If none of these security measures are used, it is strictly recommended not to use such networks to perform any authentication that involves the use of credentials (e.g., email account). Yet, a new alternative consists of using the OWE (Opportunistic Wireless Encryption) to establish an encrypted connection. Although a password is not shared a priori between a client and an access point, OWE allows them to establish a shared secret key using Diffie-Hellman key establishment protocol.

**Network Discovery.** In this scenario, an attacker uses a network adapter in “Monitor mode”. The attacker utilizes wireless scanning tools to scan all radio channels to detect and discover nearby Wi-Fi networks. If the attacker is interested in a particular network, it can learn a considerable amount of information related to that network. The information may include BSSID, network SSID, associated stations, approximate location, radio channel, security mechanism, and the brand of the used access points. These information can be
exploited for more sophisticated attacks.

**Countermeasure.** The network administrator should reduce the power transmission of its access points so that it only covers the operational area. It can also set the network configuration so that its SSID is not broadcasted and it is kept hidden. Finally, the use of a discrete SSID name may reduce the chance for attackers to link a particular SSID to a given organization Wi-Fi network and setting it as a target.

**Wardriving.** In this attack, attackers collaborate by driving around cities, neighborhoods, and villages, to scan for Wi-Fi networks that use open access mechanism or WEP. They use dedicated tools and cheap devices along with a GPS (Global Positioning System) device to record or tag the locations of the discovered insecure Wi-Fi networks in a map. The map is then shared among the attackers for future attacks. Other variants of this attack are warcycling (using bicycle), wartraining (while inside trains), warwalking, warjogging, and wardroning or warflying (using drones). We note that wardriving is a particular case of network discovery attack, as attackers target only WEP-enabled and open Wi-Fi networks, whereas network discovery target all Wi-Fi networks. Thus, it can be considered as a refinement for network discovery attack in Figure 3.3.

**Countermeasure.** The Wi-Fi network administrator should avoid using the broken security mechanisms such as WEP, or leave the network insecure. This prevents the attackers (wardrivers) from selecting the network as a good target network. Hiding the network SSID is also a good initiative.

**Physical Attack on Access Points.** Many wireless access points have their security information (e.g., logname, password, BSSID, SSID, WPA passphrase, or WPS PIN code) printed on the back or front of the device. Thus, if the access point is not kept in a secure location, an attacker can sneak by the access point and read current credentials to use them later on. The attacker can also steal devices and gain physical access to their memory to extract important information about the whole network.
Countermeasure. Wi-Fi access points must be equipped with physical security. These devices should not carry any indication about the network security settings, such as usernames and passwords. It is also recommended to place access points at places which are not easily accessible. This prevents attackers from reaching them.

Cryptanalysis

Keystream Reuse Attack. In RC4, the keystream is the concatenation of a 40-bit to 104-bit WEP-key along with an IV (Initialization Vector). The IV changes randomly or incrementally for each packet depending on the implementation. This provides a unique keystream for each packet. Nonetheless, because of the small size of the IV (24-bit), all IV possible combinations (i.e., $2^{24}$) are rapidly consumed (few seconds at 5Mbps) [JS04]. This allows an attacker who eavesdrops an ongoing communication for some time to be able to capture packets encrypted with the same keystream. By having two ciphertexts encrypted with the same keystream, the attacker can compute the xor of the plaintext of the two packets. If the attacker manages to guess at least one plaintext, it will be able to decrypt the remaining plaintexts [DN96].

Countermeasure. The size of the IV has been increased to 48 bits in TKIP (Temporal Key Integrity Protocol). Also, the way the IV is used in TKIP is more secure than it used to be in WEP. However, it is recommended to use CCMP (Counter Mode CBC-MAC Protocol) encryption mechanism rather than TKIP to avoid dealing with keystream reuse.

WEP Packet Decryption. In this attack, an attacker starts by eavesdropping a WEP authentication and tries to capture the challenge (sent in plaintext) as well as its response. Then, it xors them together to obtain the used keystream. By knowing the IV (sent unencrypted) that was used for generating the keystream, the attacker would be able to decrypt all packets that were encrypted using the same keystream.
**Countermeasure.** WEP mechanism does not provide forward secrecy. Encryption algorithms that are based on xoring the plaintext by a keystream (e.g., RC4) should not apply the same keystream twice. This provides forward secrecy.

**ChopChop Attack on RC4.** This attack was posted in the NetStumbler forum by a person under the pseudonym KoreK in 2004 [Kor04]. It allows an attacker to interactively decrypt the last $m$ bytes of an RC4 encrypted packet by sending $m \times 128$ packets to the network. It exploits the linear property of the XOR logical operator used by the RC4 algorithm for encryption, and by the CRC32 algorithm to compute the ICV code for data integrity. The attacker intercepts a target encrypted packet and chops off the last byte which invalidates the ICV code of the packet. Then by assuming the plaintext value of the chopped byte, the attacker adjusts the ICV code so that it becomes valid. Indeed, when the attacker assumes the correct byte, it receives a response from the access point. This response indirectly indicates that the assumption on the last byte was correct. The attacker repeats this process to guess all remaining bytes of the packet.

**Countermeasure.** RC4 and CRC32 algorithms have serious flaws due to some properties such as the linearity of the XOR operator. Algorithms that have this kind of property must be implemented in a very careful manner so that attackers cannot decrypt messages or tamper with messages and adjust their integrity code by flipping some bits. The AES symmetric cipher can be used along with different operational modes to mitigate this attack. This requires the use of WPA2 or WPA3 mechanisms. Nevertheless, the network administrator has to make sure that its Wi-Fi device network cards do not contain the KrØØk vulnerability\(^6\) which allows attackers to decrypt some WPA2 (AES-CCMP) packets.

**ChopChop Attack on TKIP.** This attack [TB09] allows the attacker to decrypt packets when TKIP is used with a long re-keying interval. In particular, when the range of IPv4 addresses used in a Wi-Fi network are known and the access point is operating the IEEE

---

\(^6\)KrØØk (CVE-2019-15126), discovered in 2019, is a hardware vulnerability residing in many Wi-Fi chips manufactured by Broadcom and Cypress.
3.3. ATTACKS ON WI-FI TECHNOLOGY

802.11e, the attack becomes easier. The attacker captures encrypted ARP-requests\(^7\) or responses and replays them a number of times in a ChopChop style on different QoS (Quality of Service) channels that still have a lower TSC (TKIP sequence counter). If the access point replies, then the guess was successful and the attacker manages to read the encrypted bytes. More sophisticated variants of this attack were reported in [N A13; K G15].

**Countermeasure.** Considering the network configurations that are exploited by this attack, an obvious solution consists of using a shorter TKIP re-keying interval.

### 3.3.3 Modification attacks on Wi-Fi

The following attacks allow attackers to modify the content of transmitted packets and to adjust their integrity code in such a way so that the packets look as if they were sent from a trusted source. Victim devices receive the packets and process them. Figure 3.4 illustrates an attack-defense tree\(^4\) based on modification attacks on Wi-Fi IoT infrastructures.

**Modification on WEP**

**ICV Tampering.** WEP mechanism adopts the CRC32 algorithm to generate an ICV (Integrity Check Value) to guarantee data integrity. It has been demonstrated in [BGW01] that an attacker can modify the content of a message and adjust the ICV value accordingly to make it valid. The CRC method used to compute the ICV is called a linear method (or affine) in which an attacker can predict which bits in the ICV will be flipped if the attacker changes a single bit in the message.

**Countermeasure.** The CRC algorithm is usually used for error detection and correction. It is not an adequate algorithm for integrity protection, in particular, to protect against intentional tampering. We highly recommend to use WPA2 or WPA3 where data integrity codes are generated using AES-Cipher Bloc Chaining-Message Authentication Code.

---

\(^7\)ARP (Address Resolution Protocol) is a link layer protocol that translates a logical 32-bit IPv4 address of a connected device into its physical 48-bit MAC address.
3.3. ATTACKS ON WI-FI TECHNOLOGY

Modification on WEP
- Conduct identity spoofing

Modification on WPA-TKIP
- Conduct MIC reversing
- Use one way functions for ICV computation

Figure 3.4: An ADTree based on modification attacks on a Wi-Fi IoT infrastructure (○: attacks, □: defenses, O—O: attack refinements, and ○...□: mitigation).

Modification on WPA-TKIP

Micheal Algorithm Attack. This attack is a consequence of the ChopChop attack on TKIP described in Subsection 3.3.2. When the ChopChop attack is performed on TKIP, an attacker manages to get a plaintext along with its corresponding MIC (Message Integrity Check) code. The attacker would be able to reverse the algorithm (as it is not a one way function [Woo04]) and recover the MIC key that was used to compute the MIC. This allows the attacker to modify the contents and regenerate the MIC using the disclosed key.

Countermeasure. The Micheal algorithm has been shown to contain many security flaws [Woo04; Fer02]. It is not a one way function and hence can be reversed. Thus, data integrity functions that do have such properties should not be used to preserve data integrity. WPA2 or WPA3 would be a better alternative as data integrity codes are generated using AES-CBC-MAC, which is thus far considered secure.
3.3.4 Interruption Attacks on Wi-Fi

Wi-Fi is entirely vulnerable to attacks on network availability. Practically, we emphasize on denial of service attacks. We have noticed that almost all attacks on Wi-Fi availability are due to a partial implementation of authentication in Wi-Fi. Figure 3.5 illustrates an attack-defense tree\(^4\) based on interruption attacks on a Wi-Fi IoT infrastructure.

![Attack-Defense Tree](image)

**Device Deauthentication.** The IEEE 802.11 management frames (e.g., disassociation request/response and deauthentication request/response frames) are not authenticated when WEP, WPA-PSK, and WPA2-PSK mechanisms are used. This allows an attacker to spoof any Wi-Fi device and send forged frames over the network. In the deauthentication attack, the attacker spoofs the access point and repeatedly sends forged deauthentication frames to connected devices and cause their permanent disconnection [BS03]. Another way
of performing this attack on certain access points was discussed in [LZ19a]. It consists of establishing a connection using OSA (Open System Authentication) with an access point using the access point’s MAC address (self-connection). As a consequence, certain access points react to such authentication attempt by sending a deauthentication frame to the entire network. This would deauthenticate all connected stations.

**Device Disassociation.** Similar to the previous attack, an attacker spoofs the access point and sends forged disassociation requests to connected devices and causes their disassociation from the network. The target devices get disassociated but not deauthenticated. They just have to re-associate to join the network again [BS03].

**Device Reassociation.** In this scenario, the attacker spoofs a legitimate Wi-Fi device which is associated with a given BSS and tries to reassociate it with a second BSS without any disassociation from the first one. In this way, the attacker creates inconsistencies in the network configuration causing several network protocol execution failures [JS04].

**Packet Wasting.** The IEEE 802.11 defines a power saving mode that allows Wi-Fi devices with limited power supply to switch into sleep mode to save some energy. During the power saving period, the access point buffers all packets destined to devices in sleep mode. This requires all Wi-Fi devices to be synchronized with the access point to wake up at the right time to retrieve their respective buffered packets. The key synchronization information are periodically broadcasted by the access point using the TIM (Traffic Indication Map) field of the beacon management frame. When a Wi-Fi device wakes up from the power saving mode, it requests its buffered packets if there are any from the access point. The access point delivers the packets to its destination and cleans its buffer to save memory space. In such circumstances, an attacker spoofs a legitimate Wi-Fi device while it is sleeping and causes the access point to deliver the packets and clean its buffer. Thus, when the legitimate Wi-Fi device wakes up and requests for its packets, the access point informs that device that there is nothing buffered for it [BS03].
3.3. ATTACKS ON WI-FI TECHNOLOGY

**Device Desynchronization.** This attack aims to cause disturbance on the power saving mode. The attacker spoofs the access point and sends forged beacon management frames that contain wrong synchronization information. This would cause Wi-Fi stations to wake up from the power saving mode at the wrong time [BS03].

**Traffic Freezing.** In this scenario, the attacker spoofs a legitimate Wi-Fi device and sends forged management frames informing the access point that the device is switching into power saving mode. This will considerably drain real-time traffic sent to the legitimate Wi-Fi device [JS04].

**Sleep Deprivation.** In this scenario, the attacker spoofs the access point and sends forged beacon frames containing information that indicates the presence of buffered packets for devices in power saving mode. The devices in the power saving mode send a request to retrieve their packets and stay awake for the entire beacon interval if a response is not received. By repeating this process, the attacker prevents the legitimate Wi-Fi devices from using the power saving mode and thereby drains their batteries [JS04; BS03].

**Countermeasure.** To mitigate the previous attacks, the 802.11 management frames must be authenticated. Originally, WEP and WPA-PSK did not provide any authentication for management frames. However, since the IEEE 802.11w amendment, it has become possible to use the MFP (Management Frame Protection) and mitigate all previous attacks. An alternative consists of using WPA-Enterprise with X509 digital certificates.

**Flooding**

**Battery Exhaustion.** In this attack, the attacker sends a flood of encrypted and meaningless traffic to devices with limited resources (e.g., Wi-Fi sensors). Those devices consume a large amount of energy by processing that network traffic before dropping them off.

**Countermeasure.** This attack is effective when the target device cannot distinguish whether the incoming traffic is bogus or legitimate. Also, if the target device has to perform
many cryptographic operations before concluding whether to drop or not a given packet, the attack will have a significant negative impact. If data freshness is considered, an attacker cannot flood old messages or predict future messages by spoofing devices. Moreover, the encryption algorithm should be implemented in such a way so that the target device can perform some lightweight pre-checking on the received packets before performing any expensive cryptographic operation.

Protocol Misusing

**RTS Request Misuse.** The IEEE 802.11 standard specifies a four-way packet transmission protocol called virtual carrier-sense or RTS/CTS (Request To Send/Clear To Send). This protocol allows a Wi-Fi device to allocate the radio channel to reliably send its packets. In this scenario, an attacker repeatedly sends RTS requests asking to allocate the radio channel for a long period. If the radio channel is granted to the attacker, all connected Wi-Fi devices are then denied from accessing the radio channel to send their packets [BS03].

**Countermeasure.** The network administrator must ensure that the radio channel is fairly shared and used among the associated Wi-Fi devices. For example, it can configure the access points to accept a limited number of RTS-requests per hour and per Wi-Fi device.

**Greedy Behavior.** To access the radio channel using the CSMA/CA protocol, all connected Wi-Fi devices sense the radio channel for its availability. If the radio channel is found to be clear, all Wi-Fi devices wait for a certain amount of time known as DIFS (DCF Interframe Space) before starting the transmission of their packets. If the channel is found to be busy, before or after waiting for DIFS, all Wi-Fi devices wait till the radio channel becomes clear. Once it becomes clear, all Wi-Fi devices wait for another DIFS and

---

*Distributed Coordination Function (DCF)* is a concurrent-based access mode where all Wi-Fi devices have the same chance to access the radio channel. The other mode is PCF (Point Coordination Function), where the access to the radio channel is controlled by the access point.
compute a random timer (uniformly chosen in between 0 and CW-1, where CW is the contention window, usually set to 15). The timer is then decremented while the radio channel is clear and the timer is greater than 0. The first Wi-Fi device whose timer expires, starts transmitting its packets. Meanwhile, all other Wi-Fi devices abstain from decreasing their timer as long as the channel is busy. Under these circumstances, an attacker violates the rules and starts transmitting before the expiry of the shortest possible timer. This will have two disproportional impacts. First, the data rate of the attacker will increase considerably as it is taking the whole network bandwidth. Second, the data rate of the other devices will slow down and may get nullified [BS03].

**Countermeasure.** The greedy behavior can be detected using an intrusion detection system. The system monitors how the radio channel is shared and used among a certain number of Wi-Fi devices. If a device unfairly uses the radio channel, the network administrator may suspend that device from the network for sometime or disconnect it. However, such an aggressive countermeasure can be exploited by an attacker to disconnect legitimate devices by spoofing the latter and conducting a greedy behavior attack.

**Collisions and Jamming**

**Packets Trashing.** In this scenario, the attacker sends random packets exactly at the same time where a legitimate Wi-Fi device is transmitting its packets. This causes a collision of packets which results in a wrong integrity code or FCS (Frame Check Sequence). These corrupted packets are automatically discarded upon their reception due to FCS verification error [JS04; LG05].

**Channel Jamming.** Usually, in a Wi-Fi network, communications occur on a fixed radio channel on the 2.4 GHz band. In this attack, an attacker generates random signals (noise) on the operational radio channel and causes the connected Wi-Fi devices to believe that the radio channel is busy. This drains the network performance and denies legitimate devices
from accessing the radio channel to send their packets.

**Countermeasure.** The above two attacks can be detected by analyzing the radio channels but cannot be mitigated. One of the techniques that can be employed is to automatically switch to another radio channel when the collision rate (respectively, the data rate) increases over (respectively, decreases below) a certain threshold. Also, the network administrator can set up a mechanism that can localize from where a specific network traffic or radio signal is coming from and hence may try to localize the source, i.e., attacker.

### 3.3.5 Domination Attacks on Wi-Fi

In the following subsections, we enumerate Wi-Fi attacks which compromise more than one security services. Figure 3.6 illustrates an attack-defense tree based on domination attacks on a Wi-Fi IoT infrastructure.

Figure 3.6: An ADTree based on domination attacks on a Wi-Fi IoT infrastructure (○: attacks, □: defenses, ○—○: attack refinements, and ○...□: attack mitigation).
Social Engineering

**Access Point Cloning.** This attack is also known as Evil twin. In this scenario, the attacker sets its adapter into master mode (i.e., access point mode) and adapts its network settings to be similar to a target access point settings (i.e., same MAC address, SSID, and radio channel). The attacker then boosts the signal strength to monopolize the radio channel and leaves the network with no security mechanism. This attracts careless users to connect to the attacker’s access point and use free Internet. Since no security is setup, the attacker analyzes the network traffic to extract any credentials. A more interesting scenario occurs when WPA-Enterprise is used with one-way authentication, where supplicants do not have to authenticate the WPA authenticator. The supplicants would have the option of “skip certificate validation” or “accept any certificate” to complete the authentication. The attacker may mislead supplicants to connect to the attacker’s access point instead of the legitimate one.

**Countermeasure.** This attack can be detected by setting a wireless IDS (Intrusion Detection System), such as Kismet [Ker08], that can detect the presence of identical access points within the same area [LBT11]. The IDS captures and analyzes the network traffic to detect access points with the same SSID, same MAC address, same (or different) security mechanism, but with different beacon timestamps. When WPA-Enterprise is used, mutual authentication must be established. Supplicants should not have the choice of “skipping certificate validation” or “accepting any certificate”. Such a policy is enforced in WPA3.

**Out of Band Attacks**

**Wi-Fi Backdoor.** Most access points and routers, with wireless capabilities, either bought from a retail shop or offered by an ISP (Internet Service Provider), come with default security settings (e.g., logname=admin, password=admin or logname="" and password=admin). It is the responsibility of the subscriber to change the default settings. An attacker, who
is subscribed to an ISP, tests the connectivity with all possible IP addresses that are in its network subnet. For example, if its IP address is 67.193.191.125, the attacker pings all IP addresses from 67.193.191.1 to 67.193.191.254. If an IP address replies to the ping, the attacker web-browses the IP address for the login page of the remote router. If the credentials of that router are left to default and that device allows connections from outside (i.e., Internet), the attacker will be able to login into the subscriber’s router and learn a number of sensitive information related to the subscriber’s itself or the Wi-Fi network, such as WEP/WPA key, SSID, connected clients, phone number, email address, subscriber’s address, and subscriber’s name.

**Countermeasure.** The network administrator has to change the default network and security configurations, such as the network SSID (changed to a discrete name), the IP address range, user names and passwords. It should also disable non secure mechanisms, such as WEP and WPS, on both frequency bands, i.e., 2.4GH and 5Ghz.

**Cryptanalysis**

**Online WEP Key Cracking.** In 2001, the key scheduling algorithm of RC4 used in the WEP mechanism was shown to contain severe design flaws [JS04; ASW+02; TB09; N A13; SIR02; FMS01; BGW01; SIR04; TWP07]. These flaws can be exploited by attackers to recover the WEP key and decrypt all network communications. Few years later, in 2004, researchers [SIR04] demonstrated that an attacker equipped with an ordinary computer can gradually reconstruct the WEP key in less than 2 hours. If an attacker passively eavesdrops a large number of WEP-encrypted packets (around 4,000,000 to 6,000,000 packets), it will be able to perform a byte by byte keystream recovery till recovering the whole WEP key. Interestingly, in the same year, a person under the pseudonym KoreK [Kor04] posted on the NetStumbler forum an improved version of the technique [SIR04]. Its technique reduces the required number of packets for cracking the WEP key to 700,000 packets [TB09] (500,000...
packets [TWP07]). Three years later (2007), researchers [TWP07] demonstrated that a 104-bit WEP key can be cracked in 60 seconds using 35,000 to 40,000 packets (with 0.5 probability of success) and using 85,000 packets (with 0.95 probability of success). Once the key is disclosed, the attacker can fabricate, decrypt, and/or modify the content of packets.

**Countermeasure.** The user should not use WEP mechanism as well as the devices that only support WEP. The WEP key can be cracked easily using modern computers. As Wi-Fi Alliance recommends, we also suggest the use of WPA2 or WPA3 instead of WEP.

**Offline WPA Key Cracking.** This attack aims to find the WPA password of a given Wi-Fi network. An attacker starts by eavesdropping a communication between a Wi-Fi station and an access point and tries to capture the four-way-handshake messages (by forcing a re-authentication). This handshake consists of four EAPoL messages containing values generated by both parties to prove to each other the knowledge of the correct password. Upon capturing the four EAPoL messages, the attacker operates a brute force procedure or uses a dictionary of words to find out the right password that was used during the four-way-handshake. This attack may take decades to succeed on ordinary computers if the password is strong enough. However, it may also take less than a second if the password is in the attacker’s dictionary. There are some cheap online cloud services, such as WPACracker.com [WPA09], that can be used to crack a WPA key in a shorter time. The attacker just has to capture the handshake and upload it to the cloud service.

**Countermeasure.** The network administrator has to make sure that the used WPA passwords in its Wi-Fi network fulfill certain password security patterns. These patterns include the length of the password (e.g., must be at least 6 characters) and the used letters (e.g., mixture of uppercase, lowercase, special characters, and numbers). The password should also be updated regularly and kept secret.

---

9EAPoL: Extensible Authentication Protocol over LAN.
WPA3 Key Cracking. In April 2019, researchers [VR19] discovered a set of vulnerabilities named Dragonblood. These vulnerabilities were discovered in the SAE (Simultaneous Authentication of Equals) handshake (a.k.a., dragonfly) used in WPA3-SAE. They demonstrated that by abusing timing or cache-based side-channel leaks (from the password encoding method\textsuperscript{10}), it is possible to recover the WPA3 password using password partitioning attacks. The same work showed that it is possible to trick a Wi-Fi client into downgrading from WPA3-SAE to WPA2-PSK. This would allow an attacker perform offline WPA2 key cracking attack.

Countermeasure. It is recommended [VR19] not to use a set of multiplicative groups such as group 22, 23, and 24. Also, it is recommended to use ECC DH-groups over MODP and exclude MAC addresses during password encoding. This would decrease side-channel leaks. Furthermore, to mitigate the downgrading attack, Wi-Fi clients should remember if a network supports WPA3-SAE. Wi-Fi clients should not connect to a Wi-Fi access point that indicates the support of only WPA2-PSK if the same access point has been previously saved as a WPA3-capable access point.

Key Re-installation. This set of attacks were introduced in 2017 under the name of KRACKs (Key Reinstallation Attacks) [VP17b]. It exploits the fact that some WPA implementations allow the retransmission of the third EAPoL message of the WPA four-way-handshake if an acknowledgment is not received. By doing so, the receiver reinstalled a previously installed keychain each time it receives this third EAPoL message. In addition to that, it resets the transmit packet counter as well as the receive replay counter. This forces the receiver (usually the supplicant) to reuse the same key twice (i.e., data is encrypted using the same key twice). The attacker exploits this to generate multiple attacks. To that end, the attacker first sets up a man-in-the-middle scenario between the supplicant

\textsuperscript{10}WPA3 applies two password encoding methods: (1) hash-to-curve is used when ECC (Elliptic Curve Cryptography) is adopted to encode the password into an elliptic curve point. (2) hash-to-element is used when MODP (Multiplicative groups modulo a prime) is adopted to encode the password into a group element.
3.4. ATTACKS ON BLUETOOTH TECHNOLOGY

and the access point during a four-way-handshake and prevents the supplicant acknowledgment message (i.e., the fourth EAPoL message) from reaching the access point. This consequently induces the access point to resend the third EAPoL message again to the supplicant. The latter reinstalls the derived PTK keychain and resets the nonces used by the encryption mechanism. This allows the attacker to replay and decrypt certain messages (in case of TKIP, CCMP, and GCMP) and/or forge packets (in case of TKIP and GCMP). Furthermore, if the packets can be decrypted, the attacker can perform higher level attacks.

**Countermeasure.** The network administrator must ensure that the WPA implementation used in its network meets the following criteria: (1) Does not allow the retransmission of the third EAPoL message during the four-way-handshake. (2) Does not reset the nonce if the key is reinstalled [VP17b].

**WPS Online Cracking.** WPS (Wi-Fi Protected Setup) was discovered to have a serious design flaw which can easily be exploited to brute force the PIN code and retrieve the WPA passphrase. Pixiewps [Pix14] and Reaver [Rea11], can be used for this purpose.

**Countermeasure.** To mitigate this attack, the administrator can perform one of the following: (1) Disable the WPS mechanism on both radio bands, the 2.4 GHz and the 5GHz. (2) Restrict the number of WPS PIN code failure attempts to 3 and delay the next attempt by 30 minutes.

3.4 Attacks on Bluetooth Technology

3.4.1 Fabrication attacks on Bluetooth

In the following paragraphs, we review the existing attacks on Bluetooth authentication. These attacks allow an attacker to impersonate a legitimate Bluetooth user to benefit from certain privileges and cause harm to the network. Figure 3.7 illustrates an attack-defense tree\textsuperscript{4} based on fabrication attacks on a Bluetooth infrastructure.
3.4. ATTACKS ON BLUETOOTH TECHNOLOGY

Figure 3.7: An ADTree based on fabrication attacks on a Bluetooth IoT infrastructure (○: attacks, □: defenses, ○—○: attack refinements, and ○...□: mitigation).

**Entity Spoofing**

**Bluespoofing.** Each Bluetooth device is identified by its 48-bit long unique BD_ADDR (Bluetooth Device Address) and a UTF-8 encoded user-friendly name of 248-byte maximum length. In Bluespoofing, an attacker spoofs the identity of a target Bluetooth device and impersonates it to gain unauthorized access to certain services.

**Countermeasure.** Spoofing a Bluetooth device is relatively easy. However, the attacker first needs to know the device address and user-friendly name of its target device. If the target device uses the non-discoverable mode along with the anonymity mode [GPS04], the attacker will have difficulty to learn the information needed for spoofing. Also, disabling Bluetooth when not needed is a good security practice.

**Man in The Middle Attack.** In [HH07; HH08; HT08; HT10; BWM12; SMS18], it was demonstrated how a MITM attack is possible when using SSP (Secure Simple Pairing) with passkey-entry association mode. If an attacker guesses the passkey that was previously used and knows that it will be reused in a future pairing, it will be able to impersonate...
both trusted parties during the future pairing. This is possible because there is no way to authenticate and check whether the exchanged public keys during the second phase of the SSP belong to the right entities or not since there is no certification authority.

**Countermeasure.** A lightweight certification authority must be implemented in such a way so that it certifies the ownership of a public key by a given Bluetooth device. Furthermore, it is highly recommended not to use the same SSP-passkey twice. A better alternative consists of upgrading to Secure Connections pairing, where MITM is hard to perform.

**Relay Attack.** In this attack, the attacker stands in the middle of two legitimate Bluetooth devices and tricks them to get them connected and believe that they are in close proximity. The attacker manages to set up this scenario by just relaying messages throughout a built tunnel. The attacker does not modify the content of the messages, but just relays them. The purpose is to establish a connection from a further distance in the same way as it occurs when both devices are close to each other. The tunnel can be implemented in different ways. Researchers [LÇA⁺04] have demonstrated relay attacks on Bluetooth legacy (i.e., Bluetooth versions before v2.1+EDR) by implementing the tunnel with a Bluetooth device that can impersonate one or both legitimate devices.

**Countermeasure.** One of the techniques that can be applied against relay attacks is to impose the devices to use contextual information extracted from their immediate environment [WLZ19a]. Their close proximity may be verified based on the similarity between the contextual information. Such information can include temperature, humidity, radio signals, distance between the two devices, and geographic location.

### 3.4.2 Interception attacks on Bluetooth

In this section, we present attacks that aim to affect data confidentiality in Bluetooth communications. Figure 3.8 illustrates an attack-defense tree based on interception attacks on a Bluetooth IoT infrastructure.
3.4. ATTACKS ON BLUETOOTH TECHNOLOGY

Interception attacks on Bluetooth

Perform reconnaissance

Exploit protocol vulnerabilities

Perform out-of-band attacks

Conduct FM-AM interception

Conduct side-channel attacks

Conduct bluesniffing

Conduct bluesniping

Conduct blueprinting

Conduct bluesniping

Conduct blueprinting

Use non-discoverable mode and turn off Bluetooth when not used

Use encryption mechanism, e.g., SAFER+ or AES

Apply different side-channel prevention techniques

Avoid using Bluetooth FM transmitter for sensitive data transmission

Figure 3.8: An ADTree based on interception attacks on a Bluetooth infrastructure (○: attacks, □: defenses, ○→○: attack refinements, and ○...□: attack mitigation).

Exploiting Protocol Vulnerabilities

Bluesnarfing. This attack is also known as Bluestumbling [LHH07]. It consists of exploiting a security vulnerability in the OBEX (OBject EXchange) protocol to gain unauthorized access to a Bluetooth device and copy sensitive information from the device. Such information include people addresses, calendar, contact list, call/message history, files, and other device specific information.

Countermeasure. This attack is due to a vulnerability in the OBEX protocol. This vulnerability is fixed and the network administrator must ensure that all Bluetooth devices have the updated version of the OBEX protocol.

Car Whispering. It was discovered by Trifinite Group in 2005 [H05]. Car whispering is a technique used by attackers to hack a car’s hands-free Bluetooth system. It exploits the fact that a car hands-free system uses a 4-digit PIN code which is in most cases set by
the manufacturer to “0000” or “1234”. Once connected, the attacker can insert or record audio and interact with other drivers on the move.

**Countermeasure.** Bluetooth devices that use PIN code-based authentication (i.e., legacy pairing) must use complex passphrases instead of PIN codes. If the use of PIN code cannot be avoided, the Bluetooth user must ensure that its devices are not using default values such as “0000” or “1234”.

**Reconnaissance**

**Bluesniping.** As Bluetooth was designed to be used for short-range, attackers are constrained to be located within the radio range of their targets. In 2004, a group of hackers conceived a hardware device called Bluesniper. This device, made essentially of a Yagi-antenna, allows an attacker to send and receive Bluetooth signals one mile away from its target [Her04b]. This allows the attacker to be more discrete. In [LZ19b], we have shown, using a Yagi-antenna, that it is possible to intercept BLE traffic from 425 meters away.

**Countermeasure.** Bluetooth users should reduce the transmission power to only cover the needed range. However, as in most cases, reducing the transmission power of a Bluetooth device is not possible (e.g., on a smartphone), it is highly recommended to use the Bluetooth non-discoverable mode and switch Bluetooth off when not needed. Anonymity is also an effective measure against traceability.

**Blueprinting.** In this scenario, attackers try to find out the details about nearby Bluetooth devices. Information such as Bluetooth device address, make, model, firmware version, provided services, and channels, are collected for a malicious future use. For example, an attacker can use SDP (Service Discovery Protocol) to collect necessary information and exploit them to generate other attacks [MH04].

**Countermeasure.** In Bluetooth security mode 1, 2, and 4, any device can perform service discovery as well as other operations (e.g., echo-request/echo-reply using L2PAC protocol)
on remote devices using SDP and learn useful information. This operation does not need any credentials (paring-free connection [LZ18]). However, in security mode 3, SDP can only be used if the user knows the credentials (pairing-based connection [LZ18]). In this case, the Blueprinting will be made harder.

**Bluetracking.** The attacker tracks a Bluetooth device address along with its user-friendly name and follows its movements to learn private information, such as the house address, the workplace address, the frequently visited places, and the current location [JW01].

**Countermeasure.** A Bluetooth user can adopt the non-discoverable mode along with the anonymity mode [GPS04] to hide its presence and become untraceable. It is also recommended to switch off Bluetooth when not needed.

**Bluesniffing.** This attack consists of eavesdropping ongoing Bluetooth communications to capture Bluetooth packets and extract sensitive information, such as voice (e.g., during a conversation), files, or even passwords (e.g., when using a non secure Bluetooth keyboard). For transmitting data, Bluetooth technology uses FHSS (Frequency-Hopping Spread Spectrum) to minimize interferences. It also uses packet whitening (or scrambling) for improving error resilience and security. Therefore, for the attacker to be able to capture and correctly interpret all packets being exchanged, it should know the frequency hopping sequence used between two devices as well as how to unwhiten the packets. In [SB07], it was demonstrated that the frequency hopping sequence can be determined using both the address and the clock of the master device. They also managed to unwhite packets using the lower six bits of the clock. Nowadays, it is easy to sniff an existing Bluetooth communication using dedicated hardware tools such as Ubertooth One and Sniffle [LZ19b].

**Countermeasure.** Bluetooth users must use the available encryption mechanisms. If the devices run Bluetooth v4.1+LE or earlier versions, then SAFER+ must be used. If the devices run Bluetooth v4.2+LE or newer, the AES encryption mechanism is employed.
3.4. ATTACKS ON BLUETOOTH TECHNOLOGY

Out of Band Attacks

**Side-Channel Attacks.** As the latest versions of Bluetooth use AES, it is possible to conduct different techniques of side-channel attacks. Techniques, such as DPA (Differential Power Analysis) and SPA (Simple Power Analysis) [KJJ99], can be applied to infer information that can be used to disclose the secret keys [AFM18].

**Countermeasure.** To mitigate side-channel attacks, researchers have proposed multiple approaches that could be used to counter these attacks. These approaches include, but not limited to, masking [SPQ05], cross-copying [Aga00], conditional assignment [MPS+06], bucketing [KD09], and predictive timing mitigation [ZAM11].

**Out-of-Band Interception.** Some cars do not have any Bluetooth interface. However, this does not prevent certain drivers from using a Bluetooth radio transceiver (e.g., T10 FM transmitter) that is plugged into the vehicle power outlet socket. The Bluetooth transceiver receives audio data from a paired Bluetooth device (e.g., driver’s phone) and transmits the audio data as AM (Amplitude Modulation) or FM (Frequency Modulation) radio signals on an arbitrary frequency (i.e., chosen by the driver). The driver sets its car radio filter (or car stereo) on the same frequency to receive those signals and play them on the car speakers. This creates a vulnerability since the driver is downgrading from an encrypted communication to an unencrypted communication (Radio AM or FM). Attackers just have to eavesdrop on AM and FM channels and listen to what the driver is listening.

**Countermeasure.** This attack concerns cars that do not have a Bluetooth interface. Thus, to mitigate this attack, drivers should not use a Bluetooth transceiver if the data to be broadcasted on the car speakers (e.g., phone call) contain any confidential, private, or sensitive information.
3.4.3 Modification Attacks on Bluetooth

We could not identify a particular scenario that contributes to only modification in Bluetooth communications. However, there exist attacks on Bluetooth that aim to modify the configuration (integrity) of BLE smart devices, such as heart rate monitors, smart lock, lightbulb, smart padlocks, blood GMS (Glucose Monitoring System), wristbands [LZ19b; Rya12; RR16; AWH+15; Cau16; Jas17; Tan18; CHM+14; ZL17], and the configuration of Bluetooth networks.

Device Integrity Modification. This attack consists of sending forged write-commands to a Bluetooth smart device in such a way so that the execution of the commands on the smart device modifies its security configurations (e.g., password) and parameters settings (e.g., lightbulb brightness). For example, We demonstrated how we can remotely (from 100 meters away) modify the authentication password stored on a bicycle lock and use the new password to unlock the bicycle. We also showed how the brightness of a smart lightbulb can be boosted to 255% [LZ19b].

Countermeasure. The attack on BLE device integrity is due to the “Just Works” association mode. In this association mode, any Bluetooth device can connect to a Bluetooth smart device (e.g., a home smartlock) without authentication and send unauthenticated write-commands to the smart device. The latter blindly executes the commands that allow the attacker to modify the integrity of the device (e.g., unlock the smartlock). To thwart this attack, it is highly recommended to use the other association modes. A combination of the “Just Works” and the Out-of-Band association mode can be adopted to implement a stronger authentication.

Network Integrity Modification. It is possible to switch the roles of two Bluetooth devices from “master” to “slave” or vice-versa [LZ18]. In fact, certain devices are vulnerable to bluecutting attack where an attacker can disconnect a device from another by forcing the establishment of a new connection. The attacker $d_0$ spoofs a slave device, say $d_1$, that
is connected to a master device, say $d_2$, and initiates a new connection with device $d_2$. This would disconnect $d_1$ from $d_2$ and establish a new connection with $d_2$. However, this time the attacker $d_0$ holds the role of the master and device $d_2$ holds the role of the slave. This would consequently modify the configuration and behavior of a piconet or scatternet.

**Countermeasure.** As discussed in [LZ18], Bluetooth needs to adopt a security mode that is similar to security mode 3 used in earlier Bluetooth versions. This attack is possible due to the possibility of creating a pairing-free connection when the Bluetooth security mode 4 is used. Security mode 4 allows attackers to initiate a connection with any device without any authentication (i.e., pairing-free connection).

### 3.4.4 Interruption Attacks on Bluetooth

In this section, we present Bluetooth attacks that affect the availability of Bluetooth networks. By generating such attacks, an attacker can cause a Bluetooth network to go down and make all provided services unavailable. Figure 3.9 illustrates an attack-defense tree based on interruption attacks on a Bluetooth IoT infrastructure.

**Social Engineering**

**Bluejacking.** This attack consists of sending anonymous and unsolicited messages, e.g., business cards, with an offensive content using OBEX (OBject EXchange) protocol. The attacker creates a new contact on its device and assigns the offensive content as a name to that contact and then sends that contact card to the target. When the target receives the business card, it displays the message “Would you like to add offensive content to your address book?”. We consider this attack as a denial of service as it interrupts a user from doing something useful and forces the user to do something with a view to wasting his or her time, money, and energy.

**Bluetoothing.** This attack appeared in 2004 as a hoax for arranging dates. It consists
3.4. ATTACKS ON BLUETOOTH TECHNOLOGY

of sending messages containing the word “toothing?” to nearby Bluetooth devices asking
them for a date. Such messages may be considered as a harassment for some people. We
consider this as a denial of service as the victim is forced to stop doing something useful.

**Countermeasure.** The administrator has to make sure that all of its devices are running
the updated version of the OBEX protocol to mitigate the previous attacks.

![ADTree diagram](image)

Figure 3.9: An ADTree based on interruption attacks on a Bluetooth IoT infrastructure
(○: attacks, □: defenses, ○−○: attack refinements, and ○...□: attack mitigation).

**Flooding**

**Bluespamming.** In this scenario, the attacker exploits a vulnerability in the OBEX pro-
tocol to spam a target device with a large amount of crafted files [Mul13; Haa06].

**Countermeasure.** Similar to the two previous attacks, Bluetooth devices should be run-
nning the latest version of OBEX, where the exploited vulnerability is patched.

**Battery Exhaustion.** Bluetooth networks are resource-constrained networks where de-
vices run dedicated algorithms to moderately use and conserve their limited batteries while
performing their tasks. In such conditions, an attacker exploits this energy-related weakness to exhaust batteries of those devices. The attacker repeatedly sends unsolicited and encrypted messages to target devices. These devices run cryptographic algorithms and consume a huge amount of energy before ignoring and dropping those nonsense messages.

**Countermeasure.** Bluetooth protocol must be implemented in such a way so that any device can differentiate an authentic message from a crafted one before any expensive operation. Bluetooth devices can then drop and ignore unsolicited or replayed messages without consuming much energy and without performing expensive cryptographic operations.

### Device Disruption

**Bluechopping.** The purpose of this attack is to disrupt an established Bluetooth piconet. The attacker spoofs the identity of a connected Bluetooth device in a piconet and tries to establish a connection with a master device that manages another piconet. In this case, the network will consider the spoofed device to be linked to both piconets. Consequently, it will disturb the network configuration.

**Countermeasure.** Spoofing must be made hard to perform by using the non-discoverable mode along with the anonymity mode [GPS04]. The administrator can also setup an IDS (Intrusion Detection System) [OR08; SSH18; Haa07] to detect the presence of duplicated Bluetooth devices on a piconet or scatternet.

**Bluesmacking.** The L2CAP (Logical Link Control and Adaption Protocol) allows devices to send echo-request and receive echo-reply messages from remote devices to check the connectivity (round-trip time). In this scenario, an attacker sends large-size echo-requests to its target. Upon the reception of the requests, certain device’s Bluetooth stack crashes driving the system into a livelock. Recently, a set of twelve vulnerabilities, called SweynTooth (ICS-ALERT-20-063-01), were discovered on many BLE devices. These vulnerabilities allow attackers to remotely crash a BLE device by sending non-standardized packets.
3.4. ATTACKS ON BLUETOOTH TECHNOLOGY

Countermeasure. The network administrator has to make sure that all of its devices implement a Bluetooth stack that rejects packets that have a non-standardized size and format (e.g., contain empty fields) or the stack should be able to handle large size packets.

Bluedumping. This attack occurs when two devices have already paired in the past and generated the shared link key for future communications [SW05]. The attacker spoofs one of the two devices and requests the other one to re-perform the pairing from the beginning by claiming the loss of the link key. The other device accepts the request and discards the stored link key. The attacker then aborts the connection. Thus, the spoofed device cannot automatically connect to the other device since the latter no longer has the link key. The pairing should be re-performed.

Countermeasure. This attack remains possible as long as the Bluetooth protocol allows a device to forget its link key. A network administrator can set its devices for non-discoverable mode to make the spoofing harder for the attacker as the latter primarily needs its target Bluetooth device’s address and user-friendly name.

Channel Jamming

Bluejamming. In this attack, the attacker continually sends random radio signals all over the used communication channels. This denies legitimate Bluetooth devices from accessing the radio channel and sending their data. An alternative approach consists of setting up a device (known as Bug) that has the same identity as a legitimate one which is currently connected to a piconet. When a device communicates with the legitimate device, the legitimate device and the “Bug” device will simultaneously respond and jam each other.

Countermeasure. From a practical point of view, jamming Bluetooth communications implies generating noise signals all over 79 channels in the Bluetooth band (40 channels in BLE), which is not practical for attackers. Such jamming can be detected by analyzing the
Bluetooth band and localizing from where specific disruptive signals are generated. Moreover, the network administrator can configure its Bluetooth devices to frequently change the hopping sequence.

### 3.4.5 Domination attacks on Bluetooth

In this section, we present Bluetooth attacks in which an attacker can perform a number of security breaches that affect multiple security services in an IoT Bluetooth infrastructure. Figure 3.10 illustrates an attack-defense tree based on domination attacks on a Bluetooth IoT infrastructure.

![Attack-defense tree](https://example.com/attack-defense-tree.png)

**Figure 3.10:** An ADTree based on domination attacks on a Bluetooth network (○: attacks, □: defenses, ○—○: attack refinements, and ○...□: attack mitigation).

#### Cryptanalysis

**Offline/Online PIN Cracking.** The offline PIN cracking attack is also known as PIN crunching. The attacker eavesdrops a pairing between two devices then uses the captured packets to brute force the PIN code that was used during that pairing. The time it takes to crack such a PIN code depends on its length. It has been demonstrated that a 4-digit PIN
3.4. ATTACKS ON BLUETOOTH TECHNOLOGY

Code can be cracked in less than 0.06 sec on an old Pentium IV 3GHz HT computer [SW05]. Once the PIN code is cracked, all secret keys can be generated. The attacker can intercept, decrypt, fabricate, and modify packets, and may cause interruption as well. In online PIN cracking, the attacker tries to connect with the target device by guessing different PIN code values. The attacker changes its BD_ADDR address every time a PIN guess fails. The attacker bypasses the ever-increasing delay between retries. This attack works well if a fixed or short PIN code is used.

Countermeasure. The network administrator has to make sure that its network uses simple secure pairing and not legacy pairing. If the legacy pairing cannot be avoided, then the administrator has to ensure that its devices use complex passphrases instead of simple and short PIN codes.

Offline/Online SSP Passkey Cracking. The offline SSP passkey cracking attack concerns the secure simple pairing when used with passkey-entry association mode. The attacker first captures all messages exchanged during an SSP pairing (in passkey-entry mode). Then it runs around 20 tests before figuring out the passkey that was used. Interestingly, if the same passkey is used in the later sessions, the attacker can decrypt and read the messages. Also, if the key is unchanged, the attacker can fabricate packets and cause interruption as well. In the online SSP passkey cracking attack, the attacker establishes a man-in-the-middle scenario by spoofing both communicating devices and performs a bit-by-bit test to determine the passkey. During a secure simple pairing with passkey-entry mode, in authentication stage 1, both parties authenticate and prove to each other the possession of a 20-bit passkey (e.g., \( p_k = b_0, \ldots, b_{19} \)). Each device, in each \( i^{th} \) round proves to the other device that it possesses the right bit \( b_i \). If a party gets the wrong bit, the pairing is aborted by the other party. The attacker exploits this abortion mechanism to brute force the passkey as follows: In a given round \( i \), the attacker assumes that \( b_i = 0 \). Then, if the round ends successfully, the attacker concludes that \( b_i = 0 \). Otherwise \( b_i = 1 \).
If $b_i$ is wrong, the other party aborts the communication. The attacker repeats the pairing using the previously learned $b_i$ bits until it figures out the whole passkey [SMS18].

**Countermeasure.** The user has to make sure that if SSP pairing is used along with the passkey-entry mode, the passkey has to be changed for every session and should not be used twice. Also, the use of non-discoverable mode reduces the possibility of being spoofed during an online passkey cracking attack. Turning Bluetooth off when not needed is also a good security practice.

**Exploiting Vulnerabilities in Protocols**

**Bluebugging.** In this attack [Her04a], the attacker uses the RFCOMM protocol over the serial port channel to establish a connection with the target device without any pairing. Once connected, the attacker runs on the target device a set of commands (e.g., AT commands) to perform the following: sending messages (SMS or MMS), making phone calls, reading contacts, and changing the phone configurations.

**Countermeasure.** The administrator has to make sure that its devices do not accept any serial connection through RFCOMM without asking for an authorization. Devices that do not have any high-level authorization mechanism must not be used.

**HeloMoto Attack.** This attack exploits incorrect processing of “trusted device” handling on certain Motorola devices [Lau13]. The attacker initiates a connection using the OBEX push profile and pretends sending a vCard. The sending process is interrupted by the attacker whose profile is stored on the trusted device list of the target device. By taking advantage of this entry on that list, the attacker connects to the headset profile of the target device without any authentication and uses AT commands to control it.

**Countermeasure.** The network administrator has to make sure that all its Bluetooth devices, in particular, Motorola devices, are running the updated version of OBEX. This will prevent attackers from generating the previous attack.
3.5. ATTACKS ON ZIGBEE TECHNOLOGY

3.5.1 Fabrication attack on ZigBee

In the following paragraphs, we enumerate different attacks that aim to bypass the authentication mechanisms used in ZigBee. Figure 3.11 illustrates an attack-defense tree based on fabrication attacks on a ZigBee IoT infrastructure.
3.5. ATTACKS ON ZIGBEE TECHNOLOGY

Message Spoofing

**Rogue Acknowledgment.** The IEEE 802.15.4 specification does not provide any authentication, confidentiality, or data integrity protection for the acknowledgment frames. An attacker can spoof any device and send acknowledgment frames to cause another device to believe that its frames have been correctly received by the destination. At the same time, the attacker ensures that it intercepts frames sent by a ZigBee device before sending any spoofed acknowledgment to the other party [SW04].

**Countermeasure.** The IEEE 802.15.4 specification must provide authentication for all management frames to prevent attackers from spoofing the network coordinator or ZigBee devices and forging spoofed packets. ZigBee Alliance may implement the management frame protection function (IEEE 802.11w) to solve the problem.

**Packet Injection.** This attack is also known as PIP (Packet In Packet) attack [GBM+11]. It allows an attacker to insert (hide) a packet inside a normal packet payload that is permitted onto the network. By exploiting a bit error in the original packet (i.e., outer frame), the attacker can force its malicious packet (i.e., inner frame) to be interpreted instead of the original packet. This attack is used to bypass any firewall or intrusion detection system.

**Countermeasure.** A solution to this attack was discussed in [BAK+12]. It uses bit-stuffing which is an error detection mechanism used in HDLC protocol to inhibit control information to appear in the payload of a frame.

**Replay Attack.** ZigBee uses a 32-bit frame counter to differentiate old frames from new ones. Old frames are discarded. In [Wri11], the authors have demonstrated that a replay attack is still possible. Other researchers [CWL10] have demonstrated the attack using a software tool called KillerBee [Wri11].

**Countermeasure.** Some researchers [OHA+14] have suggested that ZigBee Alliance should

---

11HDLC (High-Level Data Link) is a link layer protocol developed by the ISO (International Organization for Standardization) in 1979.
integrate a timestamp within the encryption mechanism to mitigate replay attacks. Also, data freshness can be implemented at a higher level protocol, e.g., included in the message authentication protocol [BLA+16].

Figure 3.12: An ADTree based on interception attacks on a ZigBee infrastructure (☐: attacks, □: defenses, ☐−☐: attack refinements, and ☐...☐: attack mitigation).

### 3.5.2 Interception attacks on ZigBee

In the following paragraphs, we enumerate various passive and active attacks that could be launched by an attacker to intercept, extract, and reveal sensitive information about a ZigBee network. Figure 3.12 illustrates an attack-defense tree based on interception attacks on a ZigBee IoT infrastructure.
3.5. ATTACKS ON ZIGBEE TECHNOLOGY

Out of Band Attacks

**Physical Attack.** In this scenario, an attacker physically gains access to a ZigBee device (e.g., by stealing it). The attacker uses a set of sophisticated hardware and software tools to extract sensitive information such as security keys, which are generally stored in an unencrypted format in a flash memory [DF14; LY15; Goo09]. This attack is made easy as almost all ZigBee devices are not built tamper-resistant.

**Countermeasure.** The administrator can set up an intrusion detection system which upon detecting an unplanned removal of a ZigBee device from the network, invalidates the keys and generates new ones. Another approach consists of augmenting the ZigBee devices with anti-tamper measures in such a way so that the devices start erasing their content upon detecting any physical tampering [Zil15]. However, the ZigBee network administrator has to be aware that such countermeasure can be exploited by an attacker to cause a denial of service attack by intentionally provoking the ZigBee devices to erase their contents and become unavailable.

**Side Channel Attacks.** ZigBee uses the AES symmetric encryption algorithm along with the CCM* mode to encrypt the data and to generate the message integrity code. Researchers [OC16; dS10] have demonstrated that it is possible to retrieve the secret keys by conducting a side-channel analysis on a ZigBee device. This attack assumes that the attacker has full (physical) control over the device.

**Countermeasure.** Techniques such as masking [SPQ05], cross-copying [Aga00], conditional assignment [MPS+06], bucketing [KD09], and predictive timing mitigation [ZAM11], can be used to mitigate side-channel attacks in ZigBee.

Reconnaissance

**Network Discovery.** In this scenario, the attacker eavesdrops the radio channel and tries to discover available ZigBee networks. If a network is detected, the attacker can (using
dedicated tools [Wri11]) inject packets to make the network react and disclose information related to its configuration. For instance, as part of a network discovery process, ZigBee devices send a beacon request frame on each radio channel to discover ZigBee routers or ZigBee coordinators. If a router or a coordinator receives the request frame, it responds by disclosing its PAN ID (Personal Area Network IDentifier), source address, and other useful information [CWL10]. The attacker just has to mimic a network discovery process.

**Countermeasure.** Thus far, there is no clear or simple countermeasure for this attack as beacon request frames are essential to ZigBee network discovery process as discussed in [OHA+14]. Implementing a comprehensive countermeasure for this attack requires changing the whole ZigBee protocol.

**Sniffing and Analysis**

**Same-Nonce Attack.** ZigBee uses the AES-CBC-MAC algorithm to provide encryption and data integrity. The algorithm uses a nonce along with the encryption key to produce a unique output. Similar to the key-reuse attack in Wi-Fi, if the same nonce was used along with the same key to encrypt two successive messages, an attacker would be able to recover partial information about the plaintext [WAM14]. A ZigBee device can be forced to reuse the same nonce with the same key by causing a power failure on the device.

**Countermeasure.** The network administrator has to make sure that the nonces are not used twice with the same key. One practical solution is to refresh the key after all possible values of a nonce have been used [WAM14].

**Packet Sniffing.** In this scenario, an attacker uses a ZigBee network sniffer, such as KisBee [Ker13] or KillerBee [Wri11], to capture exchanged packets over the radio. The packets are analyzed later on using a protocol analyzer, such as Wireshark, to extract sensitive information. This is possible as in most cases, ZigBee networks (e.g., in a Wireless Sensor Network) do not apply encryption just to save some energy. Also, if the standard
security level is used and the network key has not been pre-installed onto the ZigBee devices, then there is a chance to capture the network key. Indeed, the latter will be sent unencrypted by the network coordinator to every ZigBee device joining the ZigBee network [FR03].

**Countermeasure.** The administrator has to make sure that its network uses AES to encrypt the data and the new cryptographic keys are distributed securely. No secret information should be sent unencrypted (assumption set by the ZigBee Alliance [Zig08]). Also, it is a good initiative to preload the devices with cryptographic keys using an out-of-band channel to prevent their interception over the air.

### 3.5.3 Modification Attacks on ZigBee

Thus far, no attack was reported on breaching data integrity when AES-CBC-MAC is used. The only situation where this may happen is when an attacker manages to learn the key that is used to compute the MIC (Message Integrity Code). Furthermore, data integrity is not restricted to packets being sent over the radio. It also covers the protocols and programs running on a device. If an attacker gets physical access to a ZigBee device, it connects to the device and modifies internal data structures, such as static routing tables and application protocols. Then, the device behaves differently in the network. For instance, compromised devices can be used to generate Ad hoc architecture-related attacks that affect availability.

In the case of a sybil attack, the compromised node absorbs all packets by declaring itself as having multiple identities [Dou02]. Through a wormhole attack, compromised nodes are used to establish a hidden tunnel to communicate and trick other nodes that are far from each other to make them believe that they are close to each other [CWL10]. In the case of a blackhole attack, the compromised node drops all packets that it is supposed to forward [PH03]. In a selective forwarding attack, the compromised nodes select which packet to drop or forward. Finally, in a sinkhole attack, the compromised nodes are positioned next to the network coordinator (a.k.a., sink) and drop all packets coming for the sink [ZZM08].
3.5.4 Interruption Attacks on ZigBee

In the following paragraphs, we enumerate different attacks that aim to make a ZigBee network partially or completely unavailable. Figure 3.13 illustrates an attack-defense tree based on interruption attacks on a ZigBee IoT infrastructure.

Figure 3.13: An ADTree based on interruption attacks in a ZigBee infrastructure (○: attacks, □: defenses, ○−○: attack refinements, and ○...□: attack mitigation).

**Frame Trashing.** As a protection against replay attacks, ZigBee technology uses a 32-bit frame counter at the network layer to distinguish between fresh and old frames. This frame counter is not encrypted and it is reset to zero after updating the network key by the coordinator or the network administrator. If for any reason, the frame counter has been reset and the network key has not been updated, an attacker can inject a forged frame (by copying an old encrypted frame payload) and set the frame counter to its maximum. This enforces all ZigBee devices to drop all future frames upon their reception [SW04].
3.5. ATTACKS ON ZIGBEE TECHNOLOGY

**Countermeasure.** The ZigBee protocol must ensure that the network key is updated and not used twice after the expiring of the frame counter [SW04]. This prevents an attacker from using an old encrypted message and replaying it back with manipulated frame counter.

**Greedy Behavior Attack.** Similar to IEEE 802.11, an attacker violates the CSMA/CA protocol and applies the back-off algorithm to its benefits. The attacker does not wait for a random timer and interframe intervals but monopolizes the access to the radio channel.

**Countermeasure.** The network administrator can set up a network watcher to monitor device behaviors and how the shared radio channel is used. If a device is suspected for behaving maliciously, the administrator can, for example, suspend that device for the time being. However, such an aggressive countermeasure can be exploited by an attacker to suspend legitimate devices by spoofing the latter and conducting a greedy behavior attack.

**Power Draining**

**Battery Exhaustion.** In most cases, ZigBee devices are powered by a 3A battery and equipped with a sensor unit (e.g., MicaZ of CrossBow). These devices sleep most of the time and wake up when an event occurs. An attacker can generate a large amount of bogus and encrypted traffic to be processed by the ZigBee devices and prevent them from going to sleep mode. This considerably drains their power [PZV+06].

**Countermeasure.** One approach to deal with this attack is to set up an intrusion detection system to detect abnormal behaviors. An alternative is to structure the encryption and authentication of packets in such a way so that devices can pre-distinguish legitimate traffic from illegitimate traffic before processing them to avoid unnecessary draining of power.

**End-Device Sabotage.** Depending on the application, ZigBee end-devices can spend most of their time in a power-saving mode while sensing other measures such as temperature, humidity and pressure, using dedicated sensors. During this phase, the power consumption is very low. In the wake-up period of the duty-cycle, those devices consume a considerable
amount of power by sending poll requests to retrieve their data (from the network coordinator). The data was sent to those devices while they were in power-saving mode. An attacker abuses this mechanism by spoofing the network coordinator and sending broadcast or multi-cast poll replies to all poll requests to keep the devices awake. This would considerably drain their battery [VHP+13].

**Countermeasure.** The ZigBee protocol must ensure that poll requests and responses are authenticated and refreshed [BLA+16] to prevent attackers from spoofing any ZigBee device and replaying the old messages. Also, applying a mechanism that is similar to management frame protection, which is used in Wi-Fi, would be a perfect option.

**PAN-ID Conflict Attack.** Usually, a ZigBee network adopts the infrastructure mode. In this mode, a set of devices are associated with a network coordinator. Each device has its own unique PAN-ID (Personal Area Network IDentifier) and is aware of its coordinator’s PAN-ID. The existence of more than one coordinator’s PAN-ID in the same network causes a conflict which is automatically detected and reported to the coordinator. This initiates a conflict resolution procedure by generating and sharing a new PAN-ID. An attacker exploits this procedure to continually send fake conflict notification messages obliging the coordinator to initiate the resolution procedure. This will consequently drain the power source of the resource-constrained devices and delay their communications as well [SDK08].

**Countermeasure.** The user must ensure that its network provides authentication for the conflict resolution messages [BLA+16].

**Media Disruption**

**Guaranteed Time Slots Attack.** Similar to the RTS/CTS (Request To Send/Clear To Send) mechanism used in the IEEE 802.11, the IEEE 802.15.4 standard uses the GTS/ACK (Guaranteed Time Slots/Acknowledgments) mechanism to allocate the channel and guarantee a collision-free transmission [IEE03]. A ZigBee device sends a GTS-request to the
network coordinator which acknowledges its reception and takes a decision on whether it accepts or rejects the request. If the request is accepted, the device is notified in the next beacon management frame. In this circumstance, an attacker intercepts the beacon frames to learn when the GTS transmissions will take place and plans to perform random jamming. This would disrupt the transmission and cause collisions which are not supposed to happen in GTS/ACK [SDK08].

**CSMA/CA Exploit.** It is also known as link-layer jamming. In this scenario, an attacker floods a radio channel with bogus frames to unnecessarily occupy the channel. This will prevent legitimate ZigBee devices from accessing the radio channel to send their data as long as the channel is occupied [LHdH+05].

**ZigBee Radio Jamming.** In this scenario, the attacker generates random signals over the radio channel and causes interference. This will paralyze the network and prevent legitimate Zigbee devices from accessing the radio channel.

**Countermeasure.** Media disruption has always been a difficult class of attacks to mitigate. There exist some techniques that are based on radio monitoring to detect and localize unusual signals. This may be applied to detect jamming and collision when they occur and take appropriate action such as switching channels.

### 3.5.5 Domination Attacks on ZigBee

Zigbee defines a certain number of application profiles, such as HAPAP (Home Automation Public Application Profile) and ZLL (ZigBee Light Link Profile). These profiles define how messages are formatted, sent, and processed. This allows ZigBee devices from different vendors to properly communicate with each other within the framework of a particular application (e.g., home automation). To be compatible with other devices of different manufacturers, ZigBee devices have to implement a standard interface which subsequently implies the use of standard cryptographic keys. For example, the default trust center
3.6 ATTACKS ON RFID TECHNOLOGY

link key defined by ZigBee Alliance is 0x5A 0x69 0x67 0x42 0x65 0x65 0x41 0x6C 0x6C 0x69 0x61 0x6E 0x63 0x65 30 39 [Zig09]. This key is used by the trust center to encrypt the network key and send it to the devices joining the network. If an attacker can capture the encrypted network key during joining, it will decrypt the key using the standard trust center link key. This may compromise the confidentiality of the whole network as well as its availability.

**Countermeasure.** The default trust center link key should not be used since the key is considered as public knowledge and thus provides the same level of security as in the unencrypted scheme. In many cases, this default and standard key has been removed from ZigBee v3.0. For personal ZigBee applications, it is highly recommended to create the cryptographic keys and physically upload them into the devices rather than sending them over the radio [BLA+16].

3.6 Attacks on RFID Technology

3.6.1 Fabrication Attacks on RFID

In this section, we review the attacks that affect RFID authentication. These attacks allow an attacker to impersonate an RFID-tag and bypass RFID-based authentication systems, such as keyless entry systems and contactless authentication systems. Figure 3.14 illustrates an ADTree³ based on fabrication attacks on RFID systems.

**Fabrication on Entities**

**Shoplifting.** It is also known as boosting or five-fingers-discount. In many retail shops, at the shop entrance, washroom entrance, or the exit doors of the shop, an EAS (Electronic Article Surveillance) system is installed. This system detects EAS-tagged items that are sold in the retail shop and have not been disabled [Jue06]. For example, items that are being intentionally or accidentally taken away from the shop without paying their price will trigger
the EAS alarm. Shoplifting is considered an attack not based on the fact of stealing items from a shop but based on the fact of stealing the RFID-tag for further reverse-engineering. Nowadays, attackers commonly bypass EAS systems by applying different techniques, such as hiding the item in cheap foil-lined bags or using an expensive EAS-jammer.

***Countermeasure.*** Some EASs are equipped with the ability to detect foiled-lined bags and magnetic items and the customers are informed not to enter with such items into the shop. They can leave them at the entrance and collect those back before leaving the shop.

**Location-based Attacks.** This type of attacks have two main features: (1) In a normal circumstance, an RFID-tag is instantaneously activated within the range of an RFID-reader. As a result, the RFID-reader makes the wrong assumption that the RFID-tag is in its close proximity. (2) These attacks operate at the physical-layer which make them difficult, if not impossible, to mitigate using upper-layer security protocols.
3.6. ATTACKS ON RFID TECHNOLOGY

– **Distance Fraud.** The distance fraud attack allows an RFID-tag operating outside the authorized range to convince an RFID-reader that it is within the authorized range [SIR02]. The RFID-tag uses either a crafted antenna or starts sending the responses before the challenges are received to reduce the delays that may result from being outside the authorized range. The latter case can be prevented by sending multiple challenges with a strict condition that the responses must be dependent on the challenges. This attack has more effects on RFID applications where the access rights change according to the physical location.

– **Mafia Fraud.** This is also known as a relay attack [DGB87]. This attack can be performed regardless of which cryptographic system is being used and how powerful it is. It is a man or men-in-the-middle attack (depending on the number of relays). It takes place when an RFID-reader unawarely interacts with a rogue RFID-tag that manages to fool the reader into thinking that it is directly communicating with the legitimate RFID-tag. The rogue RFID-tag relays the challenges sent from the RFID-reader to the legitimate RFID-tag as well as the responses sent from the legitimate RFID-tag to RFID-reader. It has been shown that contactless smart cards (i.e., ISO-14443 standard) are vulnerable to relay attacks [KW05b]. Similarly, in [CLP05a], the authors have presented a system to carry out relay attacks on ISO-14443A (e.g., digital passport, Atmel AT88SC153 smart card, and Ticket for FIFA World cup 2006 [Kit16]). In [FDC10], the authors demonstrated how to use a relay attack to break the passive keyless entry system of various modern cars. Moreover, hundreds of high-end cars were stolen using this attack all over the world in 2019. Note that the terms mafia-fraud attack and relay attack are interchangeable. However, some authors consider mafia-fraud attack more sophisticated and active than relay attack by assuming that in a mafia-fraud attack, attackers can manipulate and modify the messages rather than simply relaying them as in a relay attack.

– **Terrorist Fraud.** In this attack [Des88], the adversary receives some support from a
3.6. ATTACKS ON RFID TECHNOLOGY

legitimate RFID-tag, e.g., with necessary information to impersonate the latter. This information does not contain any clue about the security parameters, such as the secret key. Also, this information allows the adversary to pass only a single run of the protocol.

- Distance Hijacking. In this attack [CRC11], a rogue RFID-tag convinces an RFID-reader that it is at a distance which is different from the actual distance. This is done by making use of a legitimate RFID-tag to provide the tag with a false upper bound on the distance between the reader and the tag.

Countermeasure. To mitigate location-based attacks, RFID protocols apply different techniques to ensure that an RFID-tag is inside the operational range of an RFID-reader. Classical approaches are based on measuring the round trip time of the messages. Other approaches are based on the measurement of RSSI (Receiving Signal Strength Indicator) indicator, GPS (Global Positioning System) location, temperature, light intensity, and voice recognition. A practical technique, called Faraday cage, consists of using dedicated gadgets that protect the tag from being interrogated by unauthorized readers. Passive gadgets, such as metal-shielding, cover tags and prevent radio signals from reaching them. The tags remain inactive until the owner performs an action, such as pressing a button, opening a cover, or entering biometrics or password. Reactive gadgets however, such as the Vaultcard RFID blocking card [Vau18], send strong jamming signals upon detecting a reading signal.

RFID Cloning. This attack consists of replicating an RFID-tag to create a rogue tag that is used for impersonating the authentic tag and bypassing checkpoint. Such attack has been performed to introduce bogus counterfeit pharmaceuticals and medications tagged with authentic cloned RFID-tags of legitimate medicals [Jue05]. Another proof of concept was demonstrated in DEFCON 2015 [Gru07], by creating an identical copy of the German passport using cheap off the shelf hardware.

Countermeasure. RFID-tags must be augmented with a technology that prevents cloning
(e.g., HID iClass RFID-cards), or at least, allows detecting a forged RFID-tag from an authentic one. Steganography or watermarking can be used to hide information inside authentic RFID-tags. A better alternative consists of using PUFs (Physical Unclonable Function) to implement security protocols on RFID systems.

**Social Engineering.** An attacker employs social engineering techniques to compromise an RFID authentication system and gain unauthorized access to restricted locations. For example, an attacker may conduct a tailgating attack over any person entering an access restricted building that requires a badge or access card.

**Countermeasure.** RFID users should be aware of their surroundings. Attackers use different smart social engineering techniques to distract users, gain their trust, to perform unauthorized access to certain services or physical locations. In certain circumstances, security officers are employed to secure access to critical locations.

**RFID-Tag Switching.** In this scenario, an attacker targets a RFID-tag which is tagged to a valuable object (e.g., items in the supermarket). Since tags present poor physical security, the tags that are not protected from external trespassers and can easily be captured, removed or swapped. In this attack, the attacker switches the tag of an expensive RFID-item with the one of a cheaper item to pay less at the supermarket checkout. Such an attack is possible because certain back-end servers cannot check and establish the correct association between the tag and the item.

**Countermeasure.** The cashiers should be aware of the approximate price of the items in the supermarket so that it immediately realize a tag-switching on time.

**Fabrication on Messages**

**Replay Attack.** If the messages that are exchanged between an RFID-tag and an RFID-reader do not contain any fresh nonces, an attacker can reuse old messages and replay them again to gain similar access or privileges.
3.6. ATTACKS ON RFID TECHNOLOGY

**Countermeasure.** Data freshness must be provided by the authentication protocol used by the RFID application to prevent attackers from replaying old messages and gaining unauthorized access to restricted services.

---

### Figure 3.15: An ADTree based on interception attacks on an RFID infrastructure (○: attacks, □: defenses, ○−○: attack refinements, and ○...□: attack mitigation).

#### 3.6.2 Interception Attacks on RFID

Some sophisticated RFID-tags store not only an identification number but also other personal information which may be strictly private. For example, the VeriChip-tag is a human-implantable RFID-tag designed especially for medical-record indexing. By scanning a patient’s tag, a hospital can easily locate the patient’s medical record [Hal05]. If this information is not secured, any passive or active eavesdropper can extract and learn sensitive information and then use those to perform further attacks. For example, the eavesdropper may threaten victims to publish their private data if they do not pay a certain amount
3.6. ATTACKS ON RFID TECHNOLOGY

of money. In the following paragraphs, we enumerate different attacks on RFID confidentiality. Figure 3.15 illustrates an attack-defense tree based on interception attacks on an RFID IoT infrastructure.

**RFID Eavesdropping.** Similar to other wireless technologies, RFID is also subject to eavesdropping. In this attack, the attacker uses a high-gain antenna in order to capture ongoing communications between two legitimate RFID devices, e.g., tag and reader. The attacker will be able to learn information such as the protocol being used (i.e., its message chart) or the attacker may perform traffic analysis to extract sensitive information.

**Object Inventorying.** RFID-tags store in their internal memory a unique serial number which they use for identification. Certain tags, e.g., EPC (Electronic Product Code) tags, carry other information about the item to which they are attached. The other information may include the manufacturer of the object and product code, also known as stock keeping unit. Thus, a person carrying an EPC-tag is subject to object inventorying. An attacker can silently read what objects the person is carrying and harvest personal information.

**Countermeasure.** The RFID protocol must apply lightweight (ultra-lightweight) symmetric cryptography or ECC (Elliptic Curve Cryptography) to provide data confidentiality and privacy. Sensitive information must be kept secret while being stored and processed. Any information that can be used to identify a given entity must not be revealed to unauthorized parties. Moreover, to preserve the privacy of their customers, retail shops apply EAS-killing that consists of deactivating all associated tags of the purchased items upon payment. Thus, an eavesdropper would not be able to capture information about customer’s shopping list and infer private information.

**Object Tracking.** RFID-tags generally start transmitting first after being power-supplied by a nearby RFID-reader. They send their unique identification number over the air, in most cases unencrypted, in order to identify themselves to RFID-readers. A passive eavesdropper can easily use a rogue RFID-reader with a powerful antenna to detect RFID-tags
in the neighborhood and track particular tagged-objects.

**Countermeasure.** The static identifier of a given tag may be traced when being transmitted over the air. Applying only encryption on the identifier will just transform the identifier into a meta-identifier which remains static and traceable. Thus, it is recommended to use a new nonce whenever the tag is requested by an reader. The use of the nonce along with encryption makes the transmitted information useless and unique for each session which harden the tracking process. Other techniques can also be used as well to make a RFID-tag untraceable [Jue04; FR03; G P04; JP03].

**Side-Channel Attacks.** Currently, there are two forms of side-channel attacks on RFID, timing-based and power-based. A timing-based attack consists of extracting information (e.g., secret keys) from the variations of the processing times. A power-based attack consists of extracting information from the variations of the power consumption [CLP05a; OS06].

**Countermeasure.** The RFID protocol must be implemented in such a way so that side-channel attacks become very hard to realize. Masking [SPQ05], cross-copying [Aga00], conditional assignment [MPS+06], bucketing [KD09], and predictive timing mitigation [ZAM11] can be adopted while implementing the protocol. These techniques make it harder for an attacker to perform side-channel attacks.

**Reverse Engineering.** RFID-tags and RFID-readers are subject to physical attacks. In this scenario, an attacker captures an RFID-tag or RFID-reader, and applies reverse-engineering to extract information such as the used protocol, cryptographic keys, and other confidential information.

**Countermeasure.** To physically secure RFID devices, additional measures should be taken. Traditional security measures can be used such as cameras, guards, and misuse detectors (e.g., an alarm is triggered upon RFID-tag removal).
3.6.3 Modification Attacks on RFID

An attacker may take over an RFID-tag or reader and try to modify the internal protocol in order to adapt it to its needs. For instance, the attacker can modify the reader’s functions in such a way so that it authenticates the attacker RFID-tag as a legitimate tag to bypass certain authentication systems.

3.6.4 Interruption Attacks on RFID

Due to the small size and limited resource capacity, RFID-tags are attractive target devices for denial of service attacks. In the following paragraphs, we enumerate RFID attacks on availability. Figure 3.16 illustrates an attack-defense tree based on interruption attacks on an RFID IoT infrastructure.

**RFID Desynchronization.** In some applications, the RFID-tag security parameters, e.g., secret key, need to be updated. The attacker sets a man-in-the-middle scenario and prevents the RFID-tag from being synchronized and updated with the RFID-reader. Thus, the tag will be containing old security parameters and will fail in all later authentication challenges.

**Countermeasure.** The network administrator must ensure that its RFID devices are kept updated. A more secure and not scalable way to do that is to physically update the RFID-tags and RFID-readers using dedicated devices.

**Unauthorized Tag-Disabling.** In this scenario, an attacker uses a rogue reader to manipulate an tag so that it becomes permanently or temporarily unavailable. This can be achieved by removing or destroying a physical tag (e.g., applying pressure, chemical exposure, or trimming off any visible antenna), misusing the kill command, or using a dedicated device (e.g., RFID-Zapper) to disable the tag. This will prevent all legitimate readers from communicating with the vandalized tag [BM17; MRT10].

**Countermeasure.** Most sensitive RFID-tags are equipped with a high noise-level alarms that trigger when the tag is undergoing an abnormal pressure.
3.6. ATTACKS ON RFID TECHNOLOGY

RFID Signal Jamming. In this scenario, an attacker uses an RFID-device to generate random signals over the used RFID frequencies (e.g., LF, HF, or UHF) to cram the radio channel and disrupt the correct function of RFID-tags and readers.

Countermeasure. RFID jamming can be detected by setting an RFID radio scanner that triggers an alarm when it receives useless and unexpected RFID signals. Some avoidance techniques consist of switching the operational frequency between the RFID-tag and the RFID-reader. For example, certain UHF-RFID readers in the US (e.g., those operating between 902.0 MHz and 928.0 MHz), employ frequency hopping spread spectrum (FHSS) to avoid interferences. These RFID-readers change their operational frequency from time to time. However, this only makes sense if the bandwidth ($w$) of the used frequency band is wide enough.

3.6.5 Domination Attacks on RFID

In this section, we review RFID attacks that affect multiple security services.

Offline RFID-Tag Key Cracking. Most RFID systems use a challenge-response mechanism along with a shared secret key between an RFID-tag and an RFID-reader for authentication. Due to the resource-constrained nature of some RFID-tags, very short keys...
are used (e.g., 40-bit keys in a Digital Signature Transponders or DST). This makes the brute force attack possible to crack the secret key and clone the RFID-tag \cite{BGS05}. Some researchers cracked a car DST-key in less than 30 minutes and stole their own car as well as purchased gas using a cloned SpeedPass \cite{Jue06}.

**Countermeasure.** When short keys cannot be replaced by longer keys, then it is recommended to limit the use of a short key for a limited time. The key should be changed frequently. Otherwise, it will be cracked and used to compromise the system.

### 3.7 Conclusion

In this chapter, we have develop an attack classification for IoT. The classification allows a simple, but concise categorization of attacks. We have adopted the classification to review the attacks that occurred in the last two decades on Wi-Fi, Bluetooth, ZigBee, and RFID. We have also discussed possible countermeasures that can be applied to mitigate or at least detect certain attacks.

Considering the reviewed attacks on the considered technologies and the existing security mechanisms that these technologies provide, we have observed that most attacks were due to flaws on authentication protocols. We claim that authentication is the most important and critical security service, in the sense where compromising authentication would in most cases lead to the compromising of the remaining security services. Thus, we assert that if authentication is vigorously considered and correctly implemented, a large number of attacks will be completely mitigated.

In the next chapter, we will analyze the authentication protocols that these technologies provide and present new vulnerabilities on these protocols. Also, we perform some studies to propose a new countermeasure to mitigate relay attacks for auto-theft.
Chapter 4

Vulnerability Identification and Mitigation in Wireless IoT

In this chapter, we analyze the authentication protocols that Wi-Fi and Bluetooth technologies provide, and present new security vulnerabilities on these protocols. We exploit the vulnerabilities to generate various types of attacks, in particular, attacks that affect the availability of these technologies. Also, we will propose countermeasures to mitigate the attacks. Furthermore, as part of physical security of vehicles (which are considered as connected Things), we will perform some studies to propose a new countermeasure to mitigate relay attacks (which exploit RFID authentication protocols) for auto-theft.

4.1 Introduction

The number of IoT (Internet of Things) devices, i.e., Things, is exponentially increasing every year. At the same time, the security of each connected device cannot be controlled and verified by any system before or after connecting the device. Also, as many device vendors choose cheap and fast option over good quality when it comes to “quality, time, and cost, pick any two”, millions of insecure devices are already strolling in the market, if not already connected. Furthermore, as IoT adopt existing security protocols, irrespective of whether they are known by the community to be secure or insecure, the connected devices are subject to known attacks [The16].

Interestingly, exploiting vulnerabilities in authentication protocols to generate attacks
on availability has become a common attack methodology that cybercriminals follow. They require a cheaper cost, less effort, but could create lot of damage and expensive consequences. Therefore, in this chapter, we focus on exploiting authentication protocols for the sake of generating denial of service attacks. Moreover, in the context of attack-defense arm race, there are still attacks that have not been mitigated. An example of such an attack is the relay attack for auto-theft. We devote the last part of this chapter to propose a countermeasure to mitigate this attack.

4.2 Wi-Fi Compromise

In this section, we present various potential attacks on Wi-Fi availability. These attacks aim to cause denial of service to Wi-Fi users by preventing them from successfully connecting to Wi-Fi networks. Among these attacks, 4 of them target WPA3-SAE security mechanism. The remaining attacks (5 attacks) target WPA2-PSK, where some of them can be inherited by WPA3-SAE if certain security measures are not considered. Most of the attacks (8 out of 9) that we present in this section exploit the race-condition vulnerability, which we present in the next subsection. Also, before discussing any attack, we begin by presenting the phases of the authentication protocol that is being targeted by the attacks. This would help the reader understand the attacks accordingly. We also present the attack environment that is used to generate the attacks before demonstrating any attack further.

4.2.1 Race-Condition Vulnerability

Race-condition vulnerability (CAPEC-26) results from the conception that existing authentication protocols, in general, lack intelligence (there is no notion of smart authentication protocols). This lack of intelligence comes from the fact that an authenticating party moves to the next step of the protocol based on the first message that it receives from the other authenticating party at a given stage of the protocol execution (i.e., protocol trace). In
other words, if an authenticating party Bob sends a particular message \( m \) multiple times to the authenticating party Alice, then Alice will automatically consider the first instance of the message \( m \) that it receives and will take further actions based on the content of \( m \). This is perceived as a normal behavior with respect to the protocol specifications and known communication standards. However, if for some reason the message \( m \) is not authenticated (which is the case in most authentication protocols' initial phase), the authenticating party Alice cannot verify that the first instance of message \( m \) that it has received has indeed been sent by the authentic party Bob, and not by a third party Charlie that has spoofed Bob.

We refer to this vulnerability by race-condition, as the legitimate party (unawarely) and the attacker (maliciously) will concurrently run the authentication protocol and the output of the protocol will be oriented and affected by the order of the reception of the messages. Depending on the protocol's specifications, the authentication protocol may behave in a way that is beneficial to the attacker. For example, one of the straightforward decision that is taken by the protocols is to abort the communication. This would constitute a potential flaw to generate DoS attacks.

We note that race-condition vulnerabilities have been discussed in the literature in the context of multi-threading operating systems (CAPEC-26). However, exploiting race-condition in the context of networking and authentication protocols, we have not found any research work that exploit the vulnerability to generate network communication attacks.

\subsection*{4.2.2 WPA3-SAE Authentication Phases}

The WPA3 (Wi-Fi Protected Access 3) authentication consists of three phases: (1) The SAE handshake. (2) The association phase. (3) The 4-way handshake phase, as illustrated in the MSC\(^1\) of Fig. 4.1. The first phase is also known as Dragonfly. It consists of four messages in

\footnote{MSC (Message Sequence Chart) is a graphical language for the description of the interaction between different components of a system. It is standardized by the ITU (International Telecommunication Union). In this thesis, we have used the LaTeX package developed by Mauw et al. [MB01] to draw the MSCs.}
which the supplicant\(^2\) and the access point (authenticator) use the shared network password to derive the shared key PMK (Pairwise Master Key). This phase is illustrated with much details in the MSC\(^1\) of Fig. 4.2 and discussed in the next paragraph. The second phase consists of two messages, where the supplicant sends an association request and the access point replies back by an association response. During this phase, the supplicant indicates in the association request which security parameters (i.e., authentication, encryption, and authentication key management algorithms) it wishes to use. The access point confirms or rejects the parameters in the association response message. Finally, in the last phase, both parties use the previously derived PMK key to execute the classical 4-way handshake to derive and install the PTK (Pairwise Transient Key), which is the session key.

During the SAE-handshake phase shown in Fig. 4.2 (Phase 1 in Fig. 4.1), both parties, i.e., the supplicant and the access point, agree on a cryptographic domain, ECP (Elliptic Curve groups) or MODP (Modular Exponential groups). Depending on the cryptographic domain, they use the shared network password along with a hash-to curve or hash-to-group algorithm to transform the password into an elliptic curve point \(P\) (when ECP is used) or into a multiplicative group modulo a prime element (when MODP is used). In both cases, the output is denoted by \(PWE\) (PassWord Element). In what follows, we only consider the ECP domain for WPA3-SAE description and experimentation. Each authenticating party \(i \in \{S, A\}\) generates two random values, \(rand_i\) and \(mask_i\). These two random values are used to compute two commit values, \(scal_i = (rand_i + mask_i) \cdot r\) and \(elem_i = Inv(mask_i \cdot PWE)\). The value \(scal_i\) is a scalar, whereas \(elem_i\) is an elliptic curve point which corresponds to the inverse \((Inv)\) of the point that results from the elliptic point multiplication ‘\(\cdot\)’ of the scalar \(mask_i\) by the elliptic point \(PWE\). Once computed, each party sends to the other one an authentication message with an authentication sequence.

---

\(^2\)In 802.1X terminology, Wi-Fi users are called supplicants. They authenticate themselves to the access point, which is known by the authenticator. In the rest of this section, we use the term Wi-Fi supplicant and Wi-Fi user interchangeably.
4.2. WI-FI COMPROMISE

Figure 4.1: WPA3-SAE authentication mechanism, where **Phase 1** is the SAE-handshake phase, **Phase 2** is the association phase, and **Phase 3** is the WPA2-4-way-handshake phase. The notation \( M_x \rightarrow y \) indicates a message \( M \) sent from \( x \) to \( y \). Also, \( E_x \) indicates an element \( E \) that is generated by \( x \). For \( y = \Gamma \), the destination is set to the broadcast MAC address.

number set to 0x0001\(^3\). This message is also known as the commit message and contains the tuple \((scal_i, elem_i)\).

Upon receiving the tuple, each party verifies whether the values of \( scal_i \) and \( elem_i \) are within the curve definition domain or not, i.e., \( scal_i \in [1, r] \) and \( elem_i \in \xi(F_p) \), where \( r \) is

\(^3\)In the IEEE 802.11 standard, the authentication sequence number indicates the type of the authentication frame: 0x0001 is used to indicate an authentication request frame, whereas 0x0002 is used to indicate an authentication response frame.
the prime order of the generator $G$ of the finite cyclic group $\mathbb{G}$ that defines the addition operation in the elliptic curve $\xi$. The used elliptic curve $\xi$ is defined over the prime finite field $\mathbb{F}_p$ of order $p$ (large prime number). Once verified, both parties compute a token $tok$. This token is the result of applying a HMAC (Keyed-Hash Message Authentication Code) over five concatenated elements (the concatenation is denoted by ‘‘|’’ in Fig. 4.2). For $i, j \in \{S, A\}$ and $i \neq j$, the first element is $F(rand_i \cdot (scal_j \cdot PWE \circ elem_j))$, where $F$ is a hash function and ‘‘$\circ$’’ is the elliptic curve point addition operator. The second element is $F(elem_i)$, the third is the scalar $scal_i \in [1, r[$, the fourth is $F(elem_j)$, and the fifth is the scalar $scal_j \in [1, r[$. Each party $i \in \{S, A\}$ sends its token $tok_i$ to the other party in an authentication message with an authentication sequence number set to 0x0002. This message is also known as the commit message. Each party verifies the correct derivation of

Figure 4.2: Simultaneous Authentication of Equals Handshake, also known as Dragonfly. The notation $M_{x \rightarrow y}$ indicates a message $M$ sent from $x$ to $y$. Also, $E_x$ indicates an element $E$ that is generated by $x \in \{S, A\}$. 

$scal_i = (rand_i + mask_i) [r]$ and $elem_i = Inv(mask_i \cdot PWE)$,
$tok_i = H(F(rand_i \cdot (scal_i \cdot PWE \circ elem_j)) \mid F(elem_j) \mid scal_j \mid F(elem_i) \mid scal_i)$, where $i, j \in \{S, A\}$ and $i \neq j$. 
the token by the other party. A token constitutes a proof of knowledge of the password for a given party. If both tokens are validated, the SAE-handshake succeeds and both parties use the value $F(rand_i \cdot (scal_j \cdot PWE \circ elem_j))$ as the shared PMK, which is used as a seed in the last phase to perform the 4-way-handshake.

4.2.3 Attacks on WPA3-SAE through Race-Condition

In the following subsections, we present four denial of service attacks on WPA3-SAE: (1) Attack on the 4-way-handshake downgrade protection. (2) Attack on SAE-handshake’s commit values. (3) Attack on the group/curve negotiation. (4) Attack on SAE’s commit token (the incorrect-token attack). The four attacks exploit the race-condition vulnerability (cf., Subsection 4.2.1) on WPA3-SAE handshake to deprive supplicants from connecting and joining WPA3 networks. We describe each attack individually and show its practical implementation. As the same countermeasure can be applied to mitigate the four attacks, we discuss the countermeasure in Subsection 4.2.6. First, we describe the environment used to generate the first three attacks as follows.

**Attack Environment (Attack 1, 2, and 3)**

To put the first three attacks into practice, we have adopted the network architecture shown in Figure 4.3. It consists of two Raspberry Pis B3+ and one laptop. The first Pi runs hostapd-2.7\(^4\) Linux utility (on Raspbian OS) to emulate a WPA3-SAE access point. The second Pi runs wpa_supplicant-2.7\(^5\) Linux utility (on Ubuntu MATE) to emulate a WPA3-SAE supplicant. The access point is configured to use WPA3 with SAE key management algorithm and AES-CCMP for encryption. It operates on channel 6 with an SSID set to

\(^4\)hostapd-2.7 is an open source package that allows to emulate access points on a computer. The version 2.7 supports the use of WPA3-PSK authentication protocol. It can be downloaded from https://w1.fi/releases/hostapd-2.7.tar.gz.

\(^5\)wpa_supplicant-2.7 is an open source package that allows to implement Wi-Fi supplicant on a computer. The version 2.7 supports the use of WPA3-PSK. It can be downloaded from https://w1.fi/releases/wpa_supplicant-2.7.tar.gz.
4.2. WI-FI COMPROMISE

QRST\_WPA3. We have also augmented the two Raspberry Pis with a Wi-Fi interface (ODROID Wi-Fi Module 4) as the built-in Wi-Fi network card does not support the WPA3-SAE as well as the monitor mode. We have also configured the supplicant with the correct network settings to be able to connect to the access point. We have run the access point and then run the supplicant which successfully got authenticated and associated to the access point. As the Wi-Fi network interfaces that were in our possession do not support MFP\(^6\), we have not enabled this option. Although MFP is mandatory in WPA3, enabling or disabling it does not affect the discussed attacks. In fact, it is infeasible to protect the management frames that are sent before the 4-way handshake (those are sent prior key establishment) and hence the discussed attacks are still feasible. Finally, the attacker uses the laptop (HP Probook 6560b) that runs hostapd-2.7 on Linux (Ubuntu 16.04 LTS) to emulate an evil twin of the access point. We have set the attacker’s security mechanism to be WPA2-PSK or WPA3-SAE depending on the attack scenario and set its SSID to QRST\_WPA3 and hidden. This allows the attacker to be as passive as possible. In fact, only one SSID=QRST\_WPA3 will appear on the supplicant’s device screen when scanning for available Wi-Fi networks.

\[\text{WPA3 Authenticator} \quad \text{WPA3 Attacker} \quad \text{WPA3 Supplicant}\]

\text{Raspberry Pi B3+ (Raspbian OS)}
\begin{align*}
\text{hostapd-2.7} \\
\text{SSID=DZD16050} \\
\text{Key_Mgmt=WPA-SAE} \\
\text{MFP=Disabled} \\
\text{Channel=6} \\
\text{MAC=7c:dd:90:eb:f2:f1}
\end{align*}
\begin{align*}
\text{Laptop HP6560b (Ubuntu 16.04 LTS)} \\
\text{hostapd-2.7} \\
\text{SSID=DZD16050} \\
\text{Key_Mgmt=WPA-SAE} \\
\text{MFP=Disabled} \\
\text{Channel=6} \\
\text{MAC=7c:dd:90:eb:f2:f1}
\end{align*}
\begin{align*}
\text{Raspberry Pi B3+ (Ubuntu MATE)} \\
\text{wpa_supplicant-2.7} \\
\text{SSID=DZD16050} \\
\text{Key_Mgmt=WPA-SAE} \\
\text{MFP=Disabled} \\
\text{Channel=6} \\
\text{MAC=7c:dd:90:79:f7:03}
\end{align*}

Figure 4.3: Attacker environment (from left to right): A Raspberry Pi implementing a legitimate WPA3-SAE access point, a laptop (HP Probook 6560b) used by an attacker to spoof the access point, and a Raspberry Pi implementing a legitimate WPA3-SAE supplicant.

\(^6\)MFP (Management Frame Protection) was introduced as part of the IEEE 802.11w amendment to add protection to management frames that are originally not authenticated and hence can be easily spoofed for denial of service attacks.
4.2. WI-FI COMPROMISE

Attack on the 4-way Handshake Downgrade Protection

Observation. In a Wi-Fi network that adopts the infrastructure mode, the access point periodically broadcasts management frames called beacons. These beacons reveal information about the network settings, such as synchronization information, BSSID (Basic Service Set IDentifier), SSID (Service Set IDentifier), and security information. The security information are revealed in an elementary structure called RSNE (Robust Security Network Element), which informs Wi-Fi supplicants that are interested in connecting to the network, about the supported security mechanisms (in a cipher-suite). The cipher-suite indicates which authentication, encryption, and authentication key management algorithms are supported by the access point. The supplicants can then choose the highest security mechanism that they can support from the received cipher-suite and apply it for the authentication and communication.

In WPA3, the supplicant and the access point go through three phases as illustrated in the MSC\(^1\) of Fig. 4.1. Specifically, during the 4-way-handshake, both the supplicant and access point verify whether the RSNE that the other party wishes to use is still the same and has not been modified by a third party. The access point checks whether the \(rsne_S\) of the supplicant is supported. The supplicant however, checks whether the indicated RSNE in the beacon frames and probe responses (i.e., \(rsne_A\)) are the same. If the supplicant detects an RSNE mismatch, it passively aborts the handshake. In case of the access point, the latter sends a rejection message that could be a deauthentication frame. In a nutshell, this prevents an attacker from spoofing beacon or probe response frames and announcing weaker RSNE to trick supplicants into choosing a weaker cipher-suite rather than a more secure one [VP17a].

Attack Generation. An attacker exploits this abortion behavior and sets up a MITM (Man In The Middle) attack as illustrated in the MSC\(^1\) of Fig. 4.4. By spoofing the legitimate access point and broadcasting beacon frames that announce weaker cipher-suite,
such as WPA2-PSK \((\text{rsne}_X\) in Fig. 4.4) instead of WPA3-SAE \((\text{rsne}_A\) in Fig. 4.4). The supplicant may choose WPA2-PSK over WPA3-SAE and start the authentication with the access point. At this point, the attacker stays idle and watches the scene. The supplicant will detect that the access point is actually supporting WPA3 in addition to WPA2 when it receives the third message of the 4-way handshake (viz., last message in Fig. 4.4). The supplicant detects a mismatch and aborts the connection. The attacker repeats this scenario again and again to deprive the supplicant from connecting to the Wi-Fi network. The
attacker just has to send beacon frames at a higher rate\textsuperscript{7} and rapidly reply to supplicant’s probe requests.

To experiment the attack on the 4-way-handshake downgrade protection, we have configured the WPA3 supplicant in a way so that it can connect to WPA2-PSK or WPA3-SAE access points. We have configured the attacker access point to behave as an evil twin of the legitimate access point but using WPA2-PSK instead of WPA3-SAE. We have started both access points and then executed the supplicant. We have observed (using \texttt{Wireshark}) that the supplicant has chosen to operate the WPA2-PSK instead of WPA3-SAE. In fact, after receiving probe responses from both access points (indicating the supported cipher-suite in the RSNE), the supplicant has replied back by sending an authentication frame (seq=\texttt{0x0001}) indicating the authentication algorithm \texttt{0x0}, i.e., Open System, to be used. Interestingly, both access points have replied with an authentication frame (seq=\texttt{0x0002}). The supplicant has proceeded by sending an association request in which it has indicated the selected RSNE (i.e., WPA-PSK-CCMP). The legitimate access point replied first by sending an association response indicating a rejection message with a status code \texttt{0x002b}. This code carries the message “Invalid AKMP”, which indicates invalid authentication key management protocol. The supplicant has also received the association response from the attacker (with success massage \texttt{0x0000}), but it was ignored as the supplicant has already aborted the authentication right after the rejection. We can see that an attacker can easily trick a supplicant into choosing WPA2-PSK instead of WPA3-SAE, which is the first goal in this attack. We were able to repeat this attack scenario and deprive the legitimate supplicant from connecting to the right access point. Even if the legitimate access point was configured to advertise the capability of operating both WPA2-PSK and WPA3-SAE (which is not possible in \texttt{hostapd-2.7}), the authentication would have happened through the Open System authentication. Then, during the 4-way-handshake, in particular, after

\textsuperscript{7}Typically, beacons are sent every 100 time units (beacon interval), where a time unit is 1.024ms. The attacker can change the beacon interval to be 15 instead of 100.
receiving the third EAPoL\(^8\) message, the supplicant would have aborted the authentication due to RSNE mismatch and have had restarted the authentication again.

**Attack on WPA3-SAE’s Commit Values**

**Observation.** As described in Subsection 4.2.2, the WPA3-SAE handshake runs through two subphases (viz., Fig. 4.2): commit and confirm. Specifically, during the first subphase both the supplicant and the authenticator generate a tuple \((\text{scal}_i, \text{elem}_i)\) and send it to the other party using a commit message (an authentication message with seq=0x0001). Each party \(i \in \{S, A\}\) verifies whether the tuple \((\text{scal}_{j \neq i}, \text{elem}_{j \neq i})\) contains values that are within a predefined range. If one of the parties finds out that the received tuple is out of the predefined range, i.e., \(\text{scal}_{j \neq i} \not\in [1, r]\) or \(\text{elem}_{j \neq i} \not\in \xi(F_p)\), the handshake is aborted.

**Attack Generation.** An attacker exploits this value-range checking operation to cause a denial of service on the supplicant. When the supplicant sends its first commit message (containing commit values \(\text{scal}_S\) and \(\text{elem}_S\)), the attacker in a race condition with the legitimate authenticator, replies with a rejection message as if the generated commit values \((\text{scal}_S, \text{elem}_S)\) were out of the range. The attacker just has to spoof the authenticator and reply first to the supplicant with a crafted commit message that carries an “out of range” error information. The supplicant receives the latter message and aborts the handshake as illustrated in the MSC\(^1\) of Fig. 4.5 (consider \(m_2 = m_4\) and error code\(^1\) to be “Invalid commit values”). The attacker performs this injection repeatedly, at specific instants of time, and prevents legitimate supplicants from connecting to the network.

To implement this attack, we have modified the source code of hostapd-2.7\(^9\) in such as way so that the attacker’s access point replies to the supplicant’s commit message with a commit message that contains the rejection status code 0x0001 (stating “Unspecified failure”). Next, we have run the two access points followed by the supplicant. We have

---

\(^8\)EAPoL (Extensible Authentication Protocol over LAN) is a network protocol used in 802.1X for authentication. It uses EAP protocol over Ethernet.  
\(^9\)We have modified the code located in `/hostapd-2.7/src/ap/ieee802_11.c`.
observed that the supplicant has sent its commit message and then received a first commit message from the attacker access point. As the message contained the rejection message, the supplicant has straightforwardly aborted the authentication. Although it has received the second commit message from the legitimate access point, the latter message got ignored. The access point has re-transmitted the commit message many times before aborting the authentication process.

**Attack on WPA3-SAE’s Group/Curve Negotiation**

**Observation.** During the SAE handshake, the supplicant sends to the authenticator a commit message indicating which elliptic curve or multiplicative group (denoted by group\_id\_S) it wishes to use along with the tuple (scal\_S, elem\_S). If the authenticator does not support the desired elliptic curve or multiplicative group, it sends a commit message to the supplicant to inform it that the access point does not support the desired multiplicative group or elliptic curve group.

**Attack Generation.** The attacker exploits this protocol behavior and sets up a MITM attack between the supplicant and the authenticator. It waits for a supplicant to send a commit message and quickly replies with a forged commit message informing the supplicant that the authenticator does not support the desired group\_id\_S before the supplicant receives the commit message from the legitimate authenticator. The attacker repeats this attack each time the supplicant proposes whatever cryptographic option and prevents the supplicant from connecting to the network as illustrated in the MSC\(^1\) of Fig. 4.5 (consider \(m_2 = m_5\) and error\_code\_2 to be “Unsupported Diffie-Hellman-group”). Thus, to implement this attack, we have modified the same file (i.e., ieee80211.c) in such a way so that the attacker’s access point replies to commit messages with a rejection message that contains the status code 0x004d (stating “Authentication is rejected because the offered finite cyclic group is not supported”). We have run the attack and have observed
that each time the supplicant tried to initiate the authentication, it receives the rejection message from the attacker’s access point first. We have run this attack during 30 minutes and have observed that the supplicant has performed 23 authentication attempts and all of them failed. The supplicant has tried to authenticate using the ECP finite cyclic groups 19, 20, 21, 25, and 26 (then repeating from 19) and have failed in each of them due to the attack. In this way, we have successfully managed to deprive the supplicant from getting connected to the right access point.

**Attack Environment 2 (Attack 4)**

To put the fourth attack on WPA3-SAE (i.e., incorrect-token attack) into practice, we have adopted the network architecture shown in Fig. 4.6. It consists of two Raspberry Pis B3+
and one laptop. The first Raspberry Pi (on the left) runs `wpa_supplicant-2.7` Linux utility (on Ubuntu MATE) to implement a WPA3-SAE supplicant. The second Raspberry Pi (in the middle) runs `hostapd-2.7` Linux utility (on Raspbian OS) to implement a WPA3-SAE access point. The access point is configured to use WPA3 with SAE key management algorithm and AES-CCMP for encryption. It operates on channel 1 with an SSID set to DZD16050. The attacker (on the right) uses a laptop (HP Probook 6560b) that runs `wpa_supplicant-2.7` on Linux (Ubuntu 16.04 LTS) to emulate a WPA3 supplicant. To spoof the legitimate supplicant, we have set the attacker’s MAC address to be similar to the legitimate supplicant’s MAC address. We have configured the supplicant with the correct network settings to be able to connect to the access point. However, we have configured the attacker with the wrong network settings in order to produce an incorrect token.

As the built-in wireless interface of the Raspberry Pi does not support WPA3-SAE nor the monitor mode, we have augmented the Pis with a wireless card (ODROID Wi-Fi Module 4) that fulfills our requirements. Also, similar to the previous attack environment, we have disabled MFP which does not impact our attack and results.
4.2. WI-FI COMPROMISE

Attack on WPA3-SAE’s Confirm Token

Observation. As discussed in Subsection 4.2.2, during the confirm subphase of the WPA3 authentication, each communication party \( i \in \{S, A\} \) computes a commit value, known as token \( tok_i \) and then sends the token to the other party so that the latter can verify the correct derivation of the PMK by the former party. However, if a party, say \( i \in \{S, A\} \), finds out that a token \( tok_j \), where \( j \in \{S, A\} \) and \( j \neq i \), has not been correctly computed, it immediately aborts the authentication and informs the other party in the next message. Immediately aborting might sound too drastic. In fact, from the specification perspective, an incorrect token certainly means that the other party does not know the correct password or has not computed the token in the right way. However, from a security perspective, receiving an incorrect token would also mean that an attacker may be involved in the authentication process. If the protocol does not consider the latter perspective, an attacker could act in a race condition and inject a crafted commit message that contains an incorrect token. Consequently, if a party receives and processes the crafted commit message before the legitimate commit message, the latter may abort the authentication and restart it.

Attack Generation. An attacker can target any authentication attempt from a WPA3 supplicant to an access point (authenticator) and inject a crafted commit frame that contains an incorrect token on behalf of one of the parties and at a specific instant of time. This would certainly cause the authentication to be aborted if the crafted commit message is received first and processed by one of the parties before receiving the legitimate commit message. The attacker repeats this injection at specific times to make sure that its crafted commit message is received by the target before the target receives the legitimate message. This attack scenario is illustrated in Fig. 4.7, where \( tok_X \) denotes the incorrect token.

To experiment the attack, we have started by connecting the legitimate supplicant to the access point. To that end, we have first launched the WPA3-SAE access point which started
advertising beacon frames that contain public network information, such as the BSSID (Basic Service Set Identifier), SSID (Service Set Identifier), synchronization information, and RSNE (Robust Security Network Element) which carries the supported security mechanisms as a cipher-suite. Next, we have launched the legitimate WPA3-SAE supplicant. The WPA3-SAE authentication (illustrated in Fig. 4.1) has taken place and the supplicant has successfully managed to get authenticated and associated to the access point.

Now that the network is created and registered in the supplicant (which would allow future auto-connection), we disconnect the supplicant to generate the attack scenario. We launch the legitimate supplicant and the attacker supplicant at exactly the same time. We have observed that both supplicants started the authentication by sending a commit message. Both messages have been received by the access point, which replied back with a commit message. Next, the supplicants send their confirm message to the access point.
As the confirm message that has been generated by the attacker has been received first by the access point, the latter replies back with a confirm message that carries an error status code 0x0001, which states “Unspecified failure”. The access point has also received the confirm message from the legitimate supplicant, but ignored the latter message. The supplicant then receives the access point’s confirm message and immediately aborts the authentication. By repeating this attack in a synchronous way, we have managed to prevent the legitimate supplicant from successfully connecting to the access point within an hour of execution. We have used a Python script that sends confirm messages that contain an incorrect token right after detecting a commit message sent from the victim.

Figure 4.8: WPA2-PSK authentication mechanism, where **Phase 0** is the probing phase, **Phase 1** is the standard IEEE 802.11 authentication phase, **Phase 2** is the association phase, and **Phase 3** is the 4-way-handshake phase. The notation $M_x \rightarrow y$ indicates a message $M$ sent from $x$ to $y$. Also, $E_x$ indicates an element $E$ that is generated by $x$. For $y = \Gamma$, the destination is the broadcast (i.e., FF:FF:FF:FF:FF:FF).
4.2. WI-FI COMPROMISE

4.2.4 WPA2-PSK Authentication Phases

WPA2-PSK (Pre-Shared Key) authentication consists of executing the four phases illustrated by the MSC\(^1\) of Figure 4.8. It allows two authenticating parties, namely, the authenticator and the supplicant, to authenticate each other and derive cryptographic keys. The derived keys are used for encrypting and authenticating data frames as well as protecting their integrity. During the probing phase (Phase 0) the supplicant broadcast a probe_request frame to scan for available nearby access points. Nearby access points reply by sending a probe_response frame. The supplicant, based on how it is configured or what access point the user would like to connect to, processes the response frame that came from the desired access point. It chooses a cipher-suite (i.e., authentication protocol, a key management protocol, and the encryption protocol) among the cipher-suites that the access point supports and that are indicated in the response frame in the RSN (Robust Secure Network) capabilities field. Then, the supplicant starts the standard IEEE 802.11 authentication phase (Phase 1), where the supplicant sends an authentication frame to the access point (with transaction number 0x0001) and receives an authentication frame (with transaction number 0x0002) from the latter.

Next is the association phase (Phase 2), where the supplicant reveals to the access point (AP) which cipher-suite it has chosen to adopt. If the chosen cipher-suite is supported by the AP, the latter replies with an association response containing a status code 0x0000 (i.e., association successful). The last phase, called 4-way handshake (Phase 3), consists of four EAPoL\(^8\) messages. The phase is initiated by the AP (authenticator) by sending an EAPoL message containing a nonce. The supplicant receives the nonce and generates its own nonce then uses both nonces along with many other parameters, such as the shared password and the MAC addresses of both parties, to derive a chain of keys called PTK (Pairwise Transient Key). The supplicant then sends it nonce in an EAPoL message along with a message integrity code (MIC) computed over the message using a key extracted
from the PTK (KCK: Key Confirmation Key). The authenticator goes through the same steps to generate the PTK and then computes the MIC over the supplicant’s message and compares the MIC with the one received from the supplicant. If both MICs are similar, the authenticator replies to the supplicant with a third EAPoL message containing a group key (GTK). The supplicant finalizes the authentication by sending a fourth EAPoL message as a confirmation to the authenticator. At this stage, both parties install the PTK to be used for encryption, message authentication, and data integrity.

4.2.5 Attacks on WPA2-PSK through Race-Condition

In this subsection, we present four denial of service attacks on WPA2-PSK by exploiting the race-condition vulnerability. These attacks aim to prevent Wi-Fi supplicants from successfully getting connected to a legitimate Wi-Fi access point. Although there exist many ways to perform denial of service attacks on WPA2-PSK (cf., Subsection 3.3.4 of Chapter 3), the attack scenarios that we present in this subsection target Wi-Fi supplicants that are trying to get connected to a Wi-Fi network, whereas, most of the existing denial of service attacks on WPA2-PSK, target already connected Wi-Fi supplicants. In most cases, the attacks exploit the fact that IEEE 802.11 management frames are not authenticated.

Attack Environment 3

To demonstrate the first three attacks, we have used a set of hardware and software entities. These entities are grouped into the following five components. Component 1 (C1), a laptop HP ProBook 6560b, running Linux Ubuntu 16.04 LTS operating system and hostapd-2.7. Component 2 (C2), two Wi-Fi supplicant devices, a smartphone Samsung J7-2016 (Android 8.1.0) and a tablet Huawei MediaPad M5 lite (Android 8.0.0). These two supplicants are not MFP-capable. Component 3 (C3), a Wi-Fi access point, Cisco WAP150, that is MFP-capable. Component 4 (C4), a Desktop, Dell precision T7500, running Linux Ubuntu 16.04 LTS operating system along with a Wi-Fi USB-dongle (ODROID
Wi-Fi Module 4). It also runs airdump-ng and Wireshark for traffic analysis. Component 5 (C5), a wireless router, Kisslink WR1410. In the rest of the section, we refer to a component \( i \) by \( C_i \).

**Denial of Service Attack using MFP**

**Observation.** Although IEEE 802.11w amendment was released in 2009, there are still many devices in the market that do not implement this amendment. This consequently prevents those devices from being able to connect to a Wi-Fi access point that requires its Wi-Fi supplicants to use MFP. In fact, we have observed that when a Wi-Fi supplicant that does not support MFP tries to establish a connection with an access point that is MFP-enabled, the connection attempt fails. The supplicant stops interacting with the access point upon the reception of the \textbf{probe_response} frame as a response to the supplicant’s \textbf{probe_request} frame. The \textbf{probe_response} frame that is sent by the access point contains, in its RSNE (Robust Security Network Information Element) capabilities field, the information that the access point is MFP-capable and that the access point requires the use of MFP. Therefore, when a Wi-Fi supplicant, that is not MFP-capable, tries to connect to a legitimate access point that does not use MFP in the presence of an MPF-enabled evil twin, there is a high probability that the connection will fail due to the race condition that involves both access points. If the evil twin is faster than the legitimate access point, the supplicant’s connection attempts will fail.

**Attack Generation.** To demonstrate this attack, we have used \( C_1 \) to \( C_4 \). We have used \( C_1 \) as a legitimate Wi-Fi access point that has an SSID \textbf{Ship}_2.2.4\,GHz and operates on the radio channel 1 (2.412 GHz). It uses WPA2-PSK (WPA2 with Pre-Shared Key) with CCMP\footnote{CCMP (CTR with CBC-MAC Protocol) is an encryption protocol used in WPA2. It adopts AES (Advanced Encryption Standard) in CTR mode.} encryption. We have used \( C_2 \) as two Wi-Fi supplicants that are not MFP-capable. We have used \( C_3 \) as the attacker’s access point. We have configured this access point to be
4.2. WI-FI COMPROMISE

Figure 4.9: The effect of the denial of service attack using MFP on Wi-Fi supplicants, Samsung J7-2016 (left) and Huawei MediaPad M5lite (right).

MFP-enabled. It operates on the same channel and uses the same security mechanism as C1. However, as we cannot change the MAC address (i.e., BSSID) of C3, we have changed the MAC address of C1 to the MAC address of C3. This is due to the ability of using the macchanger software utility in Linux Ubuntu and the capability of changing the MAC address of the network card of C1. In this way, we have created an evil twin of C1. We have used C4 to monitor and analyze the traffic using airdump-ng and Wireshark software utilities. We note that the attacker (C3) was located around 10 meters away from the legitimate access point (C1). Also, the supplicants (C2) were located one meter away from C1. To run the attack, we have started both access points (C1 and C3) and then have tried to connect both supplicants (C2). At the same time, we have intercepted all the traffic that was exchanged between C1, C2, and C3 using airdump-ng packet sniffer and analyzed the traffic using Wireshark packet analyzer (in C4).

Both Wi-Fi supplicants have failed to connect to the legitimate access point to the extent where both supplicants have declared on the user’s interface that the password is incorrect.
(viz., Figure 4.9). For example, during a 5-minute time frame, we have observed more than 563 authentication attempts from the tablet where all of them have failed. The complete attack scenario is illustrated by the MSC\textsuperscript{1} of Fig. 4.10.

During the probing phase (Phase 0 in Fig. 4.10), the supplicants have sent a probe\_request. Both access points have replied with a probe\_response indicating all their capabilities, including the cryptographic capabilities in the RSNE capabilities field. Depending on which AP’s probe\_response is received first by the supplicants, we have observed two scenarios:

1. If the attacker’s probe\_response is received first, the supplicants abort the connection immediately as they are not MFP-capable. This is illustrated by the MSC\textsuperscript{1} of Figure 4.11.
2. If the legitimate access point’s probe\_response is received first, the authentication phase (Phase 1) and the association phase (Phase 2) will be successfully executed with the legitimate access point. However, in the 4-way-handshake phase (Phase 3), the supplicants will abort the connection upon sending the second EAPoL\textsuperscript{8} message. As the supplicants are not associated with the attacker, the latter reacts to the EAPoL\textsuperscript{8} message by sending a disassociation frame with status code 0x0007, which states “Class 3 frame received from nonassociated STA”. The supplicants receive the disassociation frame and disconnect. We note that, even if MFP\textsuperscript{77} is being used by the attacker, the disassociation frame would still be processed by the supplicants. In fact, at that stage of authentication (Phase 3), the cryptographic keys used for MFP\textsuperscript{77} are not yet derived.
Figure 4.10: DoS attack using MFP capability over WPA2-PSK to cause supplicant connection deprivation. **Phase 0** is the probing phase, **Phase 1** is the authentication phase, **Phase 2** is the association phase, and **Phase 3** is the WPA2 4-way handshake. The notation $M_{x \rightarrow y}$ indicates a message $M$ sent from $x$ to $y$ (such that $x, y \in \{ S, A \}$). Also, $E_x$ indicates an element $E$ that is generated by $x \in \{ S, A, X \}$. 

$$m_0 = \text{Probe}_{\text{request}}(\text{capabilities}_S, \ldots , \text{ssid}_A)_{S \rightarrow A},$$

$$m_1 = \text{Probe}_{\text{response}}(\text{rsne}_A, \ldots , \text{capabilities}_A, \text{ssid}_A, \text{bssid}_A)_{A \rightarrow S},$$

$$m'_1 = \text{Probe}_{\text{response}}(\text{rsne}_X, \ldots , \text{capabilities}_A, \text{ssid}_A, \text{bssid}_A)_{A \rightarrow S},$$

$$m_2 = \text{Authentication}(\text{seq} = 0x0001, \ldots , \text{status} = \text{"Successful"})_{S \rightarrow A},$$

$$m_3 = \text{Authentication}(\text{seq} = 0x0002, \ldots , \text{status} = \text{"Successful"})_{A \rightarrow S},$$

$$m_4 = \text{Association}_{\text{request}}(\text{rsne}_S, \ldots , \text{capabilities}_S)_{S \rightarrow A},$$

$$m_5 = \text{Association}_{\text{response}}(\text{status} = 0x0001, \ldots , \text{capabilities}_A)_{A \rightarrow S},$$

$$m_6 = \text{EAPoL}_{\text{msg}}(\text{nonce}_A, \ldots )_{A \rightarrow S},$$

$$m_7 = \text{EAPoL}_{\text{msg}}(\text{nonce}_S, \text{rsne}_S, \ldots , \text{MIC})_{A \rightarrow S},$$

$$m_8 = \text{Disassociation}(\text{status} = 0x0007, \ldots )_{A \rightarrow S}.$$
4.2. WI-FI COMPROMISE

![Diagram of Wi-Fi Compromise]

Figure 4.11: Denial of service attack using MFP capability over WPA2-PSK to cause supplicant connection deprivation. **Phase 0** is the probing phase. In this scenario, the Wi-Fi supplicant receives and processes the probe response sent by the attacker before the one sent by the authenticator. The notation $\text{M}_{x \rightarrow y}$ indicates a message $\text{M}$ sent from $x$ to $y$. Also, $E_x$ indicates an element $E$ that is generated by $x \in \{S, A, X\}$.

**Denial of Service Attack using Incorrect Password**

**Observation.** Based on the same vulnerability, an attacker can cause a DoS attack on a supplicant by claiming that its provided WPA-password is incorrect during the execution of the 4-way-handshake. In fact, in the second EAPoL $^8$ message that the supplicant sends to the access point, the supplicant uses the WPA-password to derive a cryptographic key (called KCK: Key Confirmation Key) that is used to compute a message integrity code (MIC) for that message. The access point checks the MIC and replies accordingly. If the MIC is correct, the access point continuous the authentication, otherwise, it aborts. Hence, since the attacker’s WPA-password is different from the one used by the legitimate access point, the attacker would generate a message to inform the supplicant about the incorrectness of the password. This would force the supplicant to stop the authentication process if the latter receives that message before the message that is sent by the legitimate
access point (i.e., third EAPoL message of the 4-way handshake phase). Again, a race condition exists due to the real concurrency between both access points which will determine the output of the attack.

**Attack Generation.** To demonstrate this attack, we have used C1, C2, C4, and C5. In this scenario, C1 was used as the attacker’s access point. Thus, we have set its network settings to be similar to the ones of the legitimate access point (i.e., C5) but with a different password. C2 (Wi-Fi supplicants) and C4 (traffic sniffer and analyzer), were used in the same way as in the previous attack. C5 operates as the legitimate access point. Its SSID was set to Ship_0_2.4GHz. It communicates on the radio channel 1 (2.412 GHz) and uses WPA2-PSK (WPA2 with Pre-Shared Key) with CCMP encryption with no MFP. We note that the attacker’s access point (C1) and the legitimate access point (C5) were located within a building and separated by a distance of 10 meters. Also, the supplicants (C2) are located one meter away from the legitimate access point (C5).

We have started both access points and then have tried to connect both Wi-Fi supplicants to the access point with SSID “Ship_0_2.4GHz” (only one appears on the user’s list). We have noticed that both supplicants have failed to connect after several attempts. The same message of Figure 4.9 was displayed to both supplicants. We have analyzed the traffic that was intercepted during the attack. We have observed the communication scenario described in MSC of Fig. 4.12. Irrespective of which first EAPoL message the supplicants consider (i.e., either the one sent by the legitimate access point or the one sent by the attacker’s access point), the attack will succeed.
4.2. WI-FI COMPROMISE

Consider $m_1$ and choose $rsne_S$ s.t. $rsne_S \subseteq rsne_A$.

Phase 0

$m_0 = \text{Probe request (capabilities}_S, \ldots, \text{ssid}_A)_{S \rightarrow A}$,
$m_1 = \text{Probe response (rsne}_A, \ldots, \text{capabilities}_A, \text{ssid}_A, \text{bssid}_A)_{A \rightarrow S}$,
$m_2 = \text{Authentication (seq = 0x0001, \ldots, status = “Successful”)}_{S \rightarrow A}$,
$m_3 = \text{Authentication (seq = 0x0002, \ldots, status = “Successful”)}_{A \rightarrow S}$,
$m_4 = \text{Association request (rsne}_S, \ldots, \text{capabilities}_S)_{S \rightarrow A}$,
$m_5 = \text{Association response (status = 0x0001, \ldots, capabilities}_A)_{A \rightarrow S}$,
$m_6 = \text{EAPoL msg}_1(nonce, \ldots)_{A \rightarrow S}$,
$m_7 = \text{EAPoL msg}_2(nonce, rsne_S, \ldots, \text{MIC})_{A \rightarrow S}$,
$m_8 = \text{Disassociation (status = 0x000e, \ldots)}_{A \rightarrow S}$.

Figure 4.12: DoS attack using incorrect password over WPA2-PSK to cause supplicant connection deprivation. **Phase 0** is the probing phase, **Phase 1** is the authentication phase, **Phase 2** is the association phase, and **Phase 3** is the WPA2 4-way handshake. The notation $M_{x \rightarrow y}$ indicates a message $M$ sent from $x$ to $y$ (such that $x, y \in \{S, A\}$). Also, $E_x$ indicates an element $E$ that is generated by a communicating party $x$.

In the first case, if the supplicants receive the first EAPoL message from the legitimate access point, they will generate and send a second EAPoL message with a MIC that is computed based on the nonce value that they have received in the first EAPoL message. When
the attacker receives that second EAPoL message from the supplicants, the attacker generates and sends a disassociation frame to the supplicants with a status code 0x000e, which states “Message Integrity Code (MIC) failure”. The supplicants abort the connection upon the reception of the frame.

In the second case, if the supplicants receive the first EAPoL message from the attacker’s access point, they will generate and send a second EAPoL message with a MIC that is computed based on the attacker’s proposed nonce value. The legitimate access point will respond to the supplicants with a disassociation frame (with status code 0x000e), which terminates the connection. We note that the equivalent attack on WPA3-SAE for this attack is the attack on SAE’s commit token which we have discussed in Subsection 4.2.3, where the attacker use a random password to generate an incorrect commit token failing the authentication of the supplicant.

**Denial of Service Attack using WPA3**

**Observation.** During the WPA/WPA2 authentication, the supplicant chooses a cipher-suite (i.e., encryption and authentication key management protocol) to be adopted. The cipher-suite is selected from the set of cipher-suites that the access point indicates on the RSNE (Robust Security Network Information Element) capabilities field of the probe response frame sent during the probing phase (Phase 0 in Fig. 4.10). The supplicant then indicates its choice on the RSNE capabilities field of the association request sent during the association phase (Phase 2 in Fig. 4.10). Upon the reception of the association frame from the access point, the latter checks whether the chosen cipher-suite is supported. If it is supported, the authentication continuous to the next phase. Otherwise, if the cipher-suite is not supported, the access point informs the supplicant about that and aborts the connection. Hence, since a WPA/WPA2 supplicant cannot support WPA3, it will receive a negative response from the access point if it tries to connect. By setting the attacker’s access point to operate WPA3 and interfere during WPA/WPA2 authentication, there will
be a possibility to reject the cipher-suite that the supplicant proposes before the supplicant receives a positive message from the legitimate access point. This would disconnect the supplicant, which would fail to connect after multiple connection attempts.

Figure 4.13: DoS attack using WPA3-SAE over WPA2 to cause supplicant connection deprivation. Phase 0 is the probing phase, Phase 1 is the authentication phase, Phase 2 is the association phase. The notation $M_{x\rightarrow y}$ indicates a message $M$ sent from $x$ to $y$ (such that $x, y \in \{S, A\}$). Also, $E_x$ indicates an element $E$ that is generated by $x$.

**Attack Generation.** To show this attack, we have used the same components as in the previous attack (i.e., C1, C2, C4, and C5). However, in this attack scenario, C1’s network settings were set to be similar to the ones of the legitimate access point (i.e., C5) but operating WPA3-SAE with CCMP encryption instead of WPA2-PSK-CCMP. C2
4.2. WI-FI COMPROMISE

(supplicants), C4 (packet sniffer and analyzer), and C5 (legitimate access point), were used exactly in the same way as in the previous attack.

We have started by powering up the attacker’s access point. We have observed that the smartphone (Samsung J7-2016) displays the WPA3 access point (attacker’s AP) as an access point that operates WPA2-Enterprise, whereas the tablet (Huawei MediaPad M5 lite) does not even display it. We have powered up the legitimate access point and then have tried to connect the supplicants to the access point that appears on the Wi-Fi network list (there is only one SSID “Ship_0_2.4GHz”). We have noticed that both supplicants have failed to connect to the WPA2 access point and the supplicants were informed about incorrectness of the password. Interestingly, the same message of Figure 4.9 is displayed to the user after multiple connection attempts failures. To understand what has exactly happened, we have analyzed the intercepted packets using Wireshark. We illustrate the communication scenario in the MSC\(^1\) of Fig. 4.13. During the authentication, in particular, during the association phase (Phase 2 in Fig. 4.13), each supplicant has sent an association request that contains its chosen cipher-suite. Upon the reception of the association requests by the attacker, the latter has replied to the supplicants with an association response containing a status code 0x002b, which states “Invalid AKMP”). This status code indicates to the supplicants that there is an issue with the selected cipher-suite (as it only accepts WPA3) and the authentication must be stopped. This has disconnected the supplicants and made them try again and again, with a failure at each attempt.

This attack could also happen on WPA3 networks. In fact, if the supplicants are using WPA3, the attacker just has to set up an evil twin that claims to use a cipher-suite that is not recognized (e.g., WPA4) and not supported by the supplicants. The attacker will inject an association frame with status code “Invalid AKMP” to prevent the supplicants from proceeding further with the authentication process.
Denial of Service using Tear down Protection Mechanism

Observation. The WPA2-PSK authentication runs through three phases. During the first phase, usually, the Wi-Fi supplicant sends an authentication frame (with an authentication sequence number set to $0x0001^3$) to the access point. The access point replies back with an authentication frame (with an authentication sequence number set to $0x0002^3$). At this time, the supplicant is considered to be authenticated to the access point (with respect to 802.11 standard). Then, in the second phase, the supplicant sends to the access point an association request indicating which security parameters (i.e., group cipher suite, pairwise cipher suite, authentication key management, and other capabilities) it wishes to use in the RSNE information field of the frame. The access point checks these security parameters and replies back with an association response to indicate to the supplicant that it is now associated and they can start the 4-way-handshake. If the access point does not accept what the supplicant wishes to use as security parameter, it replies back with an association response that carries a rejection message (indicated in the status code of the frame control filed). When a supplicant receives a rejection message, it immediately aborts the authentication process and retries again after a short time, even if, right after that, it receives another association response that indicates an acceptance message from the same access point. An attacker can spoof an access point and inject a rejection association response to force the supplicant into aborting the authentication before it receives the acceptance association response from the true access point.

Attack Generation. To generate this attack, we have used the same laptop (HP Probook 6560b) that runs Linux (Ubuntu 16.04 LTS) and have executed the Linux open source hostapd-2.7 software tool to emulate an access point that operates WPA3-SAE. We have spoofed the Cisco WAP150 access point which was configured to use WPA2-CCMP encryption and MFP. Thus, we have actually created an evil twin for the Cisco WAP150 by spoofing its MAC address, SSID, and setting the evil twin on the same radio channel.
The evil twin however, uses WPA-SAE instead of WPA2-CCMP. Then we have used an Apple iPhone 6s to connect to the Cisco WAP150 access point after we have previously connected to it then disconnected (i.e., the Wi-Fi network is registered on the iPhone 6s). The smartphone has failed to connect. It has tried so many times to reconnect but without any success.

To understand the scenario, we have used a second wireless interface (ODROID Wi-Fi module 4) to intercept the packets and then display them on Wireshark. We have found that the authentication phase was performed with success even though the iPhone has received two authentication frames, one from the Cisco WAP150 and one from our hostapd. Then right after that, the iPhone has sent an association request indicating an RSNE that matches the one proposed by the Cisco WAP150. The hostapd has replied first by sending a rejection association response with a status code 0x002b. This code states the message “Invalid AKMP”. The second association response (carrying a success status code 0x0000) from the Cisco WAP150 did not have any impact as the iPhone has aborted the authentication immediately. The iPhone has tried the authentication again, but each time it reaches the stage of association, the Cisco WAP150 rejects the iPhone back by sending an association response with a status code 0x001e, which states “Association request rejected temporarily; try again later”, along with an association comeback time (197 ms). This is due to the MFP mechanism as the Cisco WAP150 actually believes that the iPhone is still associated with it and that the new association request may be sent from an attacker. The Cisco WAP150 starts the SA (Security Association) tear down protection mechanism to verify whether the supplicant would be able to execute the SA correctly. The Cisco WAP150 has sent an SA-query (which is an action management frame) to the iPhone. The latter is not connected (as it has aborted) and hence does not reply. The access point has sent and resent around 173 SA-queries before sending a disassociation frame with status code 0x0009 (stating “STA requesting (re)association is not authenticated with responding STA”) to the iPhone. This attack scenario is illustrated in Fig. 4.14.
Another scenario that we have observed during the experiments is that before the iPhone has retried the authentication, the Cisco WAP150 has proceeded by sending the first EAPoL message initiating the 4-way-handshake. As the iPhone has already aborted the authentication, the message was lost. Consequently, the Cisco access point has started sending disassociation frames (with status code 0x0002) after having re-transmitted fifteen times the first EAPoL message.
4.2.6 Mitigating Race-Condition Attacks

In this subsection, we provide a countermeasure to mitigate the attacks that we have demonstrated in the previous subsections. These attacks exploit the race-condition vulnerability that is mainly due to the lack of intelligence in existing Wi-Fi authentication protocols, in particular, and authentication protocols, in general. The fact that, at a given stage of an authentication protocol, an authenticating party, say Alice, replies back to the counterpart, say Bob, based on the first message that it receives from Bob, there is a risk that an attacker, Charlie, injects a spoofed message to Alice and that will be interpreted by Alice as a genuine message from Bob. This becomes a real security threat when such messages are not authenticated or cannot be authenticated, which is the case of the messages that are exchanged during the initial phase of an authentication protocol.

Algorithm 1 Authentication Protocol Stage Decision

procedure MOVE OR ABORT

1. $\Delta t \leftarrow v$; ‡ Stage time frame duration
2. $i \leftarrow 0$
3. while ($\Delta > 0$) do
   (a) Receive($m$) ‡ Receive message m
   (b) $B[i] \leftarrow m$ ‡ Buffer message m
   (c) $i \leftarrow i + 1$
   (d) $\Delta t \leftarrow \Delta t - \tau$ ‡ Time to buffer a message
4. for $j \leftarrow 0$, $i - 1$ do ‡ Process buffered messages
   (a) if ($\Psi(B[j]) == 1$) then return True
5. return False ‡ Abort authentication

To mitigate the presented attacks, we propose to augment existing Wi-Fi authentication protocols with some intelligence with respect to the way how the received messages are processed by a given communication party during an authentication. This intelligence consists of selecting one received message (from a given source) among multiple instances of the same message (i.e., same message type). The selected message could have arrived
at anytime. It should not necessarily be the first or the last message being received. At a given stage of the authentication and during a time frame of duration $\Delta t$, the selection algorithm (viz., Algorithm 1), receives messages (Line 3.a) and buffers all instances of an authentication message (Line 3.b) that are expected to be received during that stage. Once the entire time frame duration is used $\Delta t \leq 0$, the algorithm processes the buffered messages to take a decision (Line 4). If there exists a message $m_i$ such that the message allows the authentication to proceed (i.e., $\Phi(m_i) = 1$, where $\Phi(\cdot)$ is a message evaluation function), the decision algorithm returns the value true (Line 4.a) so that the supplicant considers the message $m_i$ and continues the authentication. Otherwise, if none of the buffered messages allow the authentication protocol to move forward (i.e., $\forall i \in \mathbb{N}, \Phi(m_i) = 0$) the protocol returns the value false (Line 5), which makes the supplicant abort the authentication. Applying this Algorithm 1 would certainly require from the supplicant to wait for a duration of at least $\Delta t$ before taking a decision on whether to move forward or abort the authentication protocol. This induces some delay in the execution of the protocol but mitigates the presented attacks. Also, we believe that it is worth to have that delay during the authentication and be able to successfully connect to a Wi-Fi network as the authentication is not a frequent operation. It only occurs during the first connection establishment where all cryptographic keys are derived for a secure connection. Once, the authentication is successfully completed, an attacker will not be able to generate the presented denial of service attacks.

Finally, we note that this countermeasure is completely feasible at the software level with the cheapest cost. It will consist of adding some lines of code to the protocol.

### 4.2.7 Self-connection Attack

In this subsection, we discuss an attack scenario where an attacker spoofs an access point (i.e., authenticator) and tries to establish a connection with that access point. Most access points react to such connection request in a way that can be exploited by attackers to
disconnect legitimate supplicants. If the next generation access points, in particular, those which implement WPA3, adopt the same behavior, then WPA3 will be vulnerable to this attack even if the access points use MFP.

**Observation.** We have observed that access points, in particular, those using WPA/WPA2-PSK, behave differently when we try to establish a connection with them using the OSP (Open System Authentication) authentication algorithm. Some access points allow the completion of the 802.11 standard authentication phase, and during the association phase, they reply with an association response that carries a rejection message of status code 0x000c. This code states “Association denied due to reason outside the scope of this standard”. This is because the association request that is sent to the access point does not contain any RSNE element, which is not supposed to be in WPA/WPA2-PSK. For example, such behavior was observed when we have generated this attack scenario on a Cisco WAP150 access point. Nevertheless, other access points, such as the Cisco DPC3484VM\(^{11}\) (and also the Sagemcom Fast 5250\(^{12}\)), have reacted differently by sending a deauthentication frame to the source with a status code 0x000b, which states “Disassociated because the information in the supported channels element is unacceptable”. Interestingly, when the source (attacker) is spoofing that access point, the deauthentication frame is oddly sent to the broadcast address. This will disconnect all connected supplicants as illustrated in the MSC\(^1\) of Fig. 4.15.

**Attack Generation.** We have used a laptop (HP Probook 6560b) running Linux (Ubuntu 16.04 LTS) to generate this attack. By spoofing the Cisco DPC3484VM access point’s MAC address, we have tried to establish a fake OSP (Open System Authentication) authentication (using aireplay-ng) with the spoofed access point. The access point adopts WPA2-PSK authentication mechanism. We have used another wireless interface (ODROID Wi-Fi module

\(^{11}\)The DPC3484VM is a type of router that is provided by Cogeco Inc. to residential and commercial customers in Ontario, Quebec, and some regions in the USA.

\(^{12}\)The Sagemcom Fast 5250 (a.k.a., Home Hub 2000) is an ADSL modem provided by Bell Inc. to residential and commercial customers in Canada.
4.2. WI-FI COMPROMISE

We have observed that the 802.11 authentication has been successfully realized. Then, after an association request (in OSP format) has been sent, the access point has started broadcasting deauthentication frames causing the disconnection of connected supplicants. We have tried this attack scenario on multiple models of access points. The results of the attack are summarized in Table 4.1.

Table 4.1: Self-connection attack results on different models of Access Points (✓: vulnerable and ×: not vulnerable).

<table>
<thead>
<tr>
<th>Access Points</th>
<th>Vul?</th>
<th>Access Points</th>
<th>Vul?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dlink DWR-921</td>
<td>✓</td>
<td>Sagemcom Fast 5250</td>
<td>✓</td>
</tr>
<tr>
<td>Dlink DIR-605L</td>
<td>✓</td>
<td>Linksys WRT54G</td>
<td>×</td>
</tr>
<tr>
<td>Cisco DPC3484VM</td>
<td>✓</td>
<td>Aruba AP-205</td>
<td>×</td>
</tr>
<tr>
<td>Cisco RV110W</td>
<td>✓</td>
<td>Cisco WAP150</td>
<td>×</td>
</tr>
</tbody>
</table>

We note that the goal of demonstrating this attack is not to show how an attacker can disconnect legitimate clients from a Wi-Fi network as it is easier to perform via deauthentication [BS03]. However, as deauthentication attack is no longer possible in WPA3 due to the use of MFP, we show through self-connection attack another possible way of performing disconnection irrespective of whether MFP is used or not.
**Countermeasure.** The straightforward mitigation for this attack is to ignore any connection attempt that uses the access point MAC address. The Linux `hostapd-2.7` implements such behavior. Upon the reception of an authentication frame from its MAC address, the `hostapd` access point replies with an authentication frame that contains a rejection message with status code `0x0001`. This code states “Unspecified failure”. If this is not possible to do, access points could simply reply to the previous attack with an association response carrying a rejection message (e.g., status code `0x000c`: outside the scope of the this standard) instead of sending a deauthentication to disrupt the connected supplicants. Otherwise, if this implementation flaw will be present in future WPA3 access points, the presented attack will be feasible.

### 4.2.8 Discussion

The race-condition vulnerability may seem obvious to many researchers. This maybe be the reason why it has never been explicitly reported or studied. We perceive this vulnerability as a serious specification and design flaw that needs attention from the research community. In fact, we have demonstrated the feasibility of various attacks by exploiting this vulnerability. More importantly, the presented attacks target Wi-Fi availability, which is a critical security service in a networking paradigm like IoT. These attacks will still be feasible as long as there is no official countermeasure to fix the vulnerability. Also, fixing this vulnerability is not a question of adding a patch to a protocol, but to urge protocol designers toward a new design patterns that consider integrating intelligence within protocols w.r.t. decision making. As a first step toward a more efficient countermeasure, we have proposed a possible solution to mitigate race condition-based attacks. The solution aims to make devices smarter during authentication with the price of delaying the response time. Nevertheless, the solution needs further improvements and evaluation which we leave for future work.

Finally, as per the self-connection attack, the vulnerability is clearly an implementation
flaw. Device manufacturers should invest more into testing the security of their devices before releasing them into the market. Choosing cheaper and faster option over better quality when it comes to “Quality, time, and cost, pick any two” has always made information systems vulnerable and deceived the community.

4.3 Classical Bluetooth Compromise

In this section, we identify a new vulnerability which we call connection dumping vulnerability (CDV). It is mainly due to the Bluetooth specification and the incorrect implementation of the Bluetooth stack by different manufacturers. This vulnerability can be exploited to generate different attacks which aim to abuse the availability of Bluetooth. We generate three attack scenarios which cause disconnection between devices, regardless of which Bluetooth versions and operating systems are being used. We also generate attack scenarios for roles switching and connection deprivation. We demonstrate that an attacker with a very low budget\textsuperscript{13} can negatively affect Bluetooth availability even if these networks are connecting the latest sophisticated devices. Finally, we propose possible mitigations to thwart those attacks.

4.3.1 Connection Dumping Vulnerability

In this subsection, we present the vulnerability that we have discovered in Bluetooth devices. This vulnerability can be exploited to generate devastating attacks on Bluetooth network availability. In fact, if the vulnerability is correctly exploited, an attacker will be able to realize different attack scenarios which mainly cause disconnection between legitimate Bluetooth devices in a Bluetooth network or deprive legitimate Bluetooth devices from establishing usual paired connections.

\textbf{Connection Establishment in Bluetooth.} The pairing process allows two devices to

\textsuperscript{13}The attacker only needs a Bluetooth USB dongle which may cost less than $4.
get connected and authenticated to each other to start a secure communication. Bluetooth secure communications are mainly related to application level. That is to say, in such type of communications, Bluetooth devices use Bluetooth applications running on top of the protocol stack. These applications use adopted protocols such as OBEX and AT Commands, and transport their messages over the RFCOMM or TSC transport protocol. However, there are other service protocols such as SDP (Service Discovery Protocol) and Echo-Request/Reply which do not require Bluetooth devices to be paired in order to use the services. Only, a simple and non-secured ACL (Asynchronous Connectionless Link) connection is needed. Therefore, we can state that there are two types of Bluetooth connections, a pairing-based and a pairing-free connection.

Observation. Conceptually, most Bluetooth devices are designed to accept more than one ACL-connection from different remote Bluetooth devices. This basically, allows the construction of Bluetooth networks such as scaternets. However, we have observed that some Bluetooth devices accept more than one ACL-connection from the same remote Bluetooth device, regardless of whether the connections are pairing-free or pairing-based connections. This appears to be practical since two paired devices, in addition to sending files to each other, send pings and/or request for service discovery from each other as well. Unfortunately, we have discovered that, by allowing the establishment of more than one connection at a time from the same remote Bluetooth device, the ordinary termination of one connection, automatically terminates the other ones. This seems to be an implementation flaw. However, we rather perceive it to be a serious vulnerability that can be exploited by attackers to abuse Bluetooth networks availability. We consider this flaw to be related to the specification of Bluetooth security mode 4, which allows two types of ACL-connections, namely, the pairing-based ACL-connection and the pairing-free ACL-connection.
4.3. CLASSICAL BLUETOOTH COMPROMISE

Vulnerability Exploitation. Consider a Bluetooth device which accepts the establishment of more than one ACL-connection from the same remote Bluetooth device. An attacker exploits the vulnerability by spoofing any Bluetooth device which is paired with the Bluetooth device that accepts multiple connections, and establishes a pairing-free ACL-connection (since it does not know any credentials) with the later Bluetooth device. Then, by terminating the pairing-free ACL-connection, the impersonated device gets automatically disconnected after a certain time interval $\Delta t$ of inactivity from the other device as illustrated in Figure 4.16, where the prover is the impersonated Bluetooth device and the verifier being the Bluetooth device to exploit. We call this vulnerability connection dumping vulnerability or simply CDV, since it can mainly be exploited to dump and cut down legitimate Bluetooth connections between Bluetooth devices without performing any hard cryptanalysis or other difficult tasks such as determining the frequency hopping sequence [SB07] used in the connection or breaking the SSP encryption and authentication mechanism [BWM12; SMS18]. The attacker needs just to impersonate its target and legitimately request services while the target is connected to another device.

CDV Formal Semantics. Let $D = \{d_1, \ldots, d_n\}$ be the set of nearby discovered Bluetooth devices, and let $Paired$ denotes a predicate for a couple of devices $(d_i, d_j) \in D \times D$ indicating whether the two devices are connected through a pairing-based connection or not. If two devices $d_i$ and $d_j$ are paired, we write $Paired(d_i, d_j) = True$. Let $Master$ denotes a predicate for a given Bluetooth device indicating whether that device is a master or a slave device. Hence, if device $d_i \in D$ is a slave device and $Paired(d_i, d_j) = True$ then $Master(d_i) = False$ and $Master(d_j) = True$. Finally, let $Accept$ denotes a predicate indicating whether a given Bluetooth device $d_i \in D$ accepts more than one ACL-connection (paring-based and/or pairing-free) at the same time from the same remote Bluetooth device. If that is the case, we write $Accept(d_i) = True$. Hence, a Bluetooth connection between two devices $d_i \in D$

\[^{14}\text{Note that after the invention of the Bluegun [Her04b], the attacker does not need to be within the short range of its target Bluetooth devices.}\]
and \( d_j \in D \) is vulnerable to the CDV if at least \( \text{Accept}(d_i) = \text{True} \) or \( \text{Accept}(d_j) = \text{True} \). Conversely, a Bluetooth network connecting several devices \( d_i \in \{d_1, \ldots, d_n\} \) is secure from the CDV if \( \forall d_i \in D : \text{Accept}(d_i) = \text{False} \).

Besides detecting at least two paired Bluetooth devices i.e., \((d_i, d_j) \in D \times D\) such that \( \text{Paired}(d_i, d_j) = \text{True} \), the attacker has to target Bluetooth devices satisfying the following assumption \( \Gamma = \text{Accept}(d_i) \lor \text{Accept}(d_j) \). It simply means that if both devices \( d_i \) and \( d_j \) are paired together, at least one of them should accept more than one connection from the other Bluetooth device. At the same time, we emphasize that the attacker assumes that those devices are not set on non-discoverable\footnote{Bluetooth technology allows Bluetooth devices to be set on non-discoverable mode in order to hide their presence to nearby Bluetooth devices.} mode which make it easy for the attacker to perform the spoofing. Otherwise, the attacker should perform brute-forcing attacks [SB07] to detect the presence of nearby devices.

---

**Figure 4.16:** Connection dumping vulnerability exploitation.
4.3.2 Attack Scenarios on Bluetooth Availability

In this subsection, we generate three attack scenarios on Bluetooth security mode 4. These attack scenarios exploit the CDV (Connection Dumping Vulnerability) presented in the previous subsection to affect the availability of Bluetooth networks, in general, and Bluetooth devices, in particular. Note that we only generate three attack scenarios in this thesis, but there may be other attack scenarios that may be generated by exploiting this vulnerability.

**Attack Scenario 1 (SDP Connection Dump)**

This attack scenario exploits the CDV using the SDP (Service Discovery Protocol) protocol to affect the availability of a Bluetooth network. Bluetooth devices use the SDP protocol to discover available services (e.g., printing service) running on remote Bluetooth devices. It is based on client/server communication concept where a SDP client requests (via a SDP-request) the SDP server for service records. The SDP server which maintains a list of service records that describe the characteristics of particular services associated to the server, replies back to the client via SDP-response containing the requested records. To that end, a pairing-free ACL (Asynchronous Connectionless Link) connection is established between the client and the server. Once all service records are received, the connection is ordinarily terminated by the SDP-client.

To exploit the vulnerability, the attacker follows the steps described in Algorithm 1. Basically, the attacker scans and inquires the neighborhood to discover nearby Bluetooth devices (Line 1 and 2). Assuming the attacker has discovered two paired Bluetooth devices where at least one of them accepts more than one ACL-connection at a time from the same source. The attacker spoofs one of the Bluetooth device’s MAC address\(^\text{16}\), user-friendly name, and optionally the device class (Line 4) and then continuously sends SDP requests

\(^{16}\)In a Bluetooth network, each Bluetooth device is uniquely identified by a 48-bit Bluetooth device MAC address denoted by BD_ADDR.
to the device that accepts multiple ACL-connections (Line 5). This will consequently dis- 
connect the spoofed device. In fact, when a SDP request is sent to the device that accepts 
multiple connections, a pairing-free ACL-connection is established without any authentica-
tion. This connection is then normally terminated by the attacker after all service records 
are retrieved by sending an ACL-disconnect request. The remote server disconnects upon 
receiving the disconnection request and replies back with a disconnection response. After a 
short time of inactivity from the SDP server, the spoofed device disconnects.

Algorithm 2 SDP Connection Dump

1. Scan for nearby Bluetooth devices;
2. Identify a set of target Bluetooth devices \( D \in \{d_1, \ldots, d_n\} \);
3. Assuming \( \Gamma = True \), select a couple of Bluetooth devices \( (d_i, d_j) \in D \times D \);
4. If \( \text{Accept}(d_i) \ \text{and} \ \text{Accept}(d_j) = True \) then spoof device \( d_s \in \{d_i, d_j\} \);
   (a) Spoof BD_ADDR address of device \( d_s \);
   (b) Spoof user-friendly name of device \( d_s \);
   (c) Spoof class of device \( d_s \);
Else if \( \text{Accept}(d_i) \ \text{or} \ \text{Accept}(d_j) = True \) then spoof device \( d_s \in \{d_i, d_j\} \) such that 
   \( \text{Accept}(d_s) = False \);
   (a) Spoof BD_ADDR address of device \( d_s \);
   (b) Spoof user-friendly name of device \( d_s \);
   (c) Spoof class of device \( d_s \);
Else Abort;
5. While (1): Send one SDP-Request to target Bluetooth device \( d_t \in \{d_i, d_j\} \setminus \{d_s\} \);

Attack Scenario 2 (Ping Connection Dump)

The second attack scenario is quite similar to the previous one but uses the echo-request 
and echo-reply protocol instead of SDP to disconnect legitimate Bluetooth devices. The 
echo-request and echo-reply L2CAP (Logical Link Control and Adaptation Protocol) layer 
protocol is used by Bluetooth devices to test the round-trip time of their ACL-connection 
with remote Bluetooth devices. To that end, it initiates a pairing-free ACL-connection
Algorithm 3 RFCOMM Connection Dump
1. Scan for nearby Bluetooth devices;
2. Identify a set of target Bluetooth devices $D \in \{d_1, \ldots, d_n\}$;
3. Assuming $\Gamma = True$, select a couple of Bluetooth devices $(d_i, d_j) \in D \times D$;
4. If $(Accept(d_i) \text{ and } Accept(d_j) = True)$ then spoof device $d_s \in \{d_i, d_j\}$;
   (a) Spoof BD_ADDR address of device $d_s$;
   (b) Spoof user-friendly name of device $d_s$;
   (c) Spoof class of device $d_s$;
Else if $(Accept(d_i) \text{ or } Accept(d_j) = True)$ then spoof device $d_s \in \{d_i, d_j\}$ such that $Accept(d_s) = False$;
   (a) Spoof BD_ADDR address of device $d_s$;
   (b) Spoof user-friendly name of device $d_s$;
   (c) Spoof class of device $d_s$;
Else Abort;
5. While (1):
   (a) Establish an RFCOMM connection with the target device $d_t \in \{d_i, d_j\} \setminus \{d_s\}$;
   (b) Abort spoofed connection before authentication.

with the remote Bluetooth device, then sends echo-requests over that connection. The remote device responds back with an echo-reply. Once the desired number of echo-requests is achieved, the connection is normally terminated by the device which initiated the connection. Similar to SDP connection dump, an attacker performs the steps described in Algorithm 1, where this time it uses Ping requests instead of SDP request.

Attack Scenario 3 (RFCOMM Connection Dump)

The third attack scenario is somehow different from the previous scenarios. It uses a pairing-based ACL-connection instead of a pairing-free ACL-connection. Basically, almost all application-based connections use the RFCOMM protocol as their transport protocol. Therefore, any application (e.g., obexftp or minicom) that uses any of the adopted protocols (e.g., OBEX or AT-Commands) built on top of the RFCOMM transport protocol can be
4.3. CLASSICAL BLUETOOTH COMPROMISE

used to generate this attack scenario. As illustrated in Algorithm 2, the attacker discovers
two paired Bluetooth devices where at least one of them accepts multiple ACL-connections
from the same source (Line 1-3). Then, it establishes a spoofed ACL-connection with the
device that accepts multiple connections, using an application that runs an adopted proto-
col over RFCOMM (Line 4-5a). For example, obexftp is used to transfer files. Finally, the
attacker aborts the connection just before the authentication procedure starts (Line 5b).
This will consequently disconnect the spoofed Bluetooth device.

4.3.3 Role Switching Attack

The role switching attack is actually a consequence of the previous attack scenarios. As we
mentioned in the beginning, Bluetooth adopts a master-salve communication mode. In this
mode, the Bluetooth device which initiates a connection is assigned the role of a master
whereas the device which accepts the connection is assigned the role of a slave. Therefore,
if an attacker manages to initiate a pairing-free ACL-connection with a remote Bluetooth
device that accepts multiple ACL-connections, the Bluetooth device roles may be switched
depending on the current device configurations. In fact, when the attacker discovers two
paired Bluetooth devices $d_i$ and $d_j$ satisfying assumption $\Gamma$, and spoofs one of the devices
$d_s \in \{d_i,\ldots,d_j\}$, it performs one time, a connection dumping to cause a disconnection.
Thus, if the spoofed device was a master device, i.e., $\text{Master}(d_s) = \text{True}$, the roles will
change and the master becomes the salve and vice-versa. Indeed, upon disconnection, the
slave will try to reconnect as a master, which will change the roles. Another scenario to
switch the roles consists of just spoofing the slave device and establishing a pairing-free
ACL-connection with the master device without terminating the connection.

4.3.4 Implementation and Experimental Evaluation

In this subsection, we present our implementation of the attack scenarios presented in
the previous section. We describe our testbed configuration with respect to the target
Bluetooth devices and Bluetooth attacker. Since all presented attack scenarios exploit the same vulnerability and aim for the same purpose, we only consider reporting the experience related to the SDP connection dump by providing the attack code and performing a dynamic analysis during the attack using hcidump\textsuperscript{17} and Wireshark\textsuperscript{18}.

**Evaluation Environment**

We consider a set of devices, namely, smartphones, laptops, handsfree, cars, and smartwatches. These devices run different types of operating systems or firmwares and implement different versions of Bluetooth protocol stack. Concretely, we consider Android, iOS, MacOS X, Windows, Linux, and other operating systems and Bluetooth versions from v2.0+EDR to 5. For the attacker profile, we use a laptop HP ProBook 6560b (Intel Core i5 CPU and 4GB of RAM) operating on Linux Ubuntu 16.04, Kernel 4.13.0-45-generic and running the Bluez 5.37 Linux Bluetooth host implementation. We also use an ORICO BTA-403 Bluetooth dongle (which costs around $4) and a set of software tools such as hciconfig, hcitool, sdptool, l2ping, spooftooph, obexftp, minicom and rfcomm. Finally, in order to follow what is happening during the attacks, we use some Linux software utilities such as hcidump and Wireshark. The hcidump tool allows us to capture all messages being sent and received by the attacker’s Bluetooth controller and store them into files in PCAP-format. These PCAP-files are then read using Wireshark to visualize and analyze the packets that were exchanged between the attacker and target devices.

**Attack Scenario Implementation**

We develop a Linux bash script in order to launch the presented attacks, namely, SDP connection dump, Ping connection dump, and RFCOMM connection dump\textsuperscript{19}. Figure 4.17

\textsuperscript{17}hcidump is a Linux utility which allows the monitoring of Bluetooth activity. It reads raw HCI data coming from and going to a Bluetooth device.

\textsuperscript{18}Wireshark is a free and open source packet analyzer: www.wireshark.org.

\textsuperscript{19}A demonstration of the attack can be watched at: https://www.youtube.com/watch?v=4-E6jcUa0-4&ab_channel=DzSec.
4.3. CLASSICAL BLUETOOTH COMPROMISE

1. #! /bin/bash
2. hciconfig -a
3. hciconfig hci0 up
4. hciconfig -i hci0 scan
5. spooftooph -i hci0 -a spoofed_device_BDADDR
6. hciconfig -i hci0 name 'spoofed_device_name'
7. SDP_Num=0
8. while $one
9. do
10. sdptool browse $target_BDADDR >> /dev/null
11. let SDP_Num++
12. echo $(date +"%T")': Disconnection'(''$SDP_Num')'
13. done

Figure 4.17: Bash script for SDP connection dumping attack.

illustrates a snippet of the bash script used to launch the SDP connection dump attack, where the target_BDADDR variable refers to the MAC address of the target device. To run the attack, the attacker performs the following steps: By executing Line 2, the attacker displays all available Bluetooth controllers physically connected to its device along with other related information such as the name of the controller, status (up or down), BD_ADDR address, device class, manufacturer, and Bluetooth version. In our case, the ORICO BTA-403 Bluetooth dongle has the pseudonym hci0. This interface is set up in Line 3. Next, the attacker scans for nearby available Bluetooth devices (Line 4). This inquires nearby devices and grab useful information about these devices such as devices BD_ADDR addresses, names, and classes. Then, considering $D = \{d_1,\ldots,d_n\}$, the set of discovered devices, the attacker is assumed to know that at least two devices are paired i.e., $\exists(d_i,d_j) \in D^2$ s.t. $\text{Paired}(d_i,d_j) = True$. If the attacker knows this information, he can try to impersonate one of the two paired devices, let us say $d_i$, by changing its Bluetooth controller’s user-friendly name as well as his Bluetooth controller’s BD_ADDR address into the $d_i$’s user-friendly name and BD_ADDR address (Line 5 and 6). Finally, the attacker sends SDP-requests to the target device $d_j$ (Line 8 to 13). This repeatedly initiates and terminates pairing-free ACL-connections with the target device $d_j$, which results in a permanent disconnection of the legitimate device $d_i$. 
Table 4.2: Connection dumping attack results with respect to our Bluetooth devices testbed (✓: vulnerable, ×: not vulnerable, and •: not concerned). We note that this table reports only 33 tested devices. A complete version of this table can be found at https://www.docdroid.net/OSAScDs/cdv-pdf#page=2, where we have tested 87 Bluetooth-capable devices.

<table>
<thead>
<tr>
<th>Devices Name</th>
<th>Devices Type</th>
<th>Firmwares or Operating Systems</th>
<th>Bluetooth Versions</th>
<th>Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hewlett-Packard</td>
<td>Laptop</td>
<td>Ubuntu 16.04.1</td>
<td>v2.1+EDR</td>
<td>×</td>
</tr>
<tr>
<td>ProBook 6500b</td>
<td>Laptop</td>
<td>Kernel 4.13.0-45</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lenovo Yoga 720</td>
<td>Laptop</td>
<td>Windows 10</td>
<td>v4.1+LE</td>
<td>✓</td>
</tr>
<tr>
<td>ACER Aspire E15</td>
<td>Laptop</td>
<td>Windows 10</td>
<td>v4.0+LE</td>
<td>✓</td>
</tr>
<tr>
<td>ASUS N43S</td>
<td>Laptop</td>
<td>Windows 7 SP1</td>
<td>v3.0+HS</td>
<td>✓</td>
</tr>
<tr>
<td>Apple MacBook Air</td>
<td>Laptop</td>
<td>MacOS</td>
<td>v4.0+LE</td>
<td>×</td>
</tr>
<tr>
<td>Apple iPad Air 2</td>
<td>Tablet</td>
<td>iOS 9.3.5</td>
<td>v4.2+LE</td>
<td>×</td>
</tr>
<tr>
<td>Apple iPhone 8</td>
<td>Smartphone</td>
<td>iOS 11.4</td>
<td>5</td>
<td>×</td>
</tr>
<tr>
<td>Apple iPhone 7</td>
<td>Smartphone</td>
<td>iOS 11.3</td>
<td>v4.2+LE</td>
<td>×</td>
</tr>
<tr>
<td>Apple iPhone 4</td>
<td>Smartphone</td>
<td>iOS 7.1.2</td>
<td>v2.1 plus EDR</td>
<td>×</td>
</tr>
<tr>
<td>Sony Xperia Z2</td>
<td>Smartphone</td>
<td>Android 6.0.1</td>
<td>v4.0+LE</td>
<td>×</td>
</tr>
<tr>
<td>Samsung Galaxy S8</td>
<td>Smartphone</td>
<td>Android 8.0</td>
<td>v4.2+LE</td>
<td>×</td>
</tr>
<tr>
<td>Samsung Galaxy S7</td>
<td>Smartphone</td>
<td>Android 8.0</td>
<td>v4.2+LE</td>
<td>×</td>
</tr>
<tr>
<td>Samsung Galaxy A5</td>
<td>Smartphone</td>
<td>Android 7.0</td>
<td>v4.2+LE</td>
<td>×</td>
</tr>
<tr>
<td>Samsung Galaxy J5</td>
<td>Smartphone</td>
<td>Android 7.1.1</td>
<td>v4.1+LE</td>
<td>✓</td>
</tr>
<tr>
<td>Samsung Galaxy J7</td>
<td>Smartphone</td>
<td>Android 7.0</td>
<td>v4.1+LE</td>
<td>✓</td>
</tr>
<tr>
<td>Samsung Galaxy S4</td>
<td>Smartphone</td>
<td>Android 5.0.1</td>
<td>v4.0+LE</td>
<td>✓</td>
</tr>
<tr>
<td>Samsung Grand Prime</td>
<td>Smartphone</td>
<td>Android 5.1.1</td>
<td>v4.0+LE</td>
<td>✓</td>
</tr>
<tr>
<td>LG Nexus 5X</td>
<td>Smartphone</td>
<td>Android 8.0.0</td>
<td>v4.2+LE</td>
<td>✓</td>
</tr>
<tr>
<td>Huawei Nexus 6P</td>
<td>Smartphone</td>
<td>Android 8.1.0</td>
<td>v4.2+LE</td>
<td>✓</td>
</tr>
<tr>
<td>HTC One M8</td>
<td>Smartphone</td>
<td>Android 4.4</td>
<td>v4.0+LE</td>
<td>✓</td>
</tr>
<tr>
<td>BLU Studio G2</td>
<td>Smartphone</td>
<td>Android 6.0</td>
<td>v4.0+LE</td>
<td>✓</td>
</tr>
<tr>
<td>Motorola Moto Z2</td>
<td>Smartphone</td>
<td>Android 7.1.1</td>
<td>v4.2+LE</td>
<td>✓</td>
</tr>
<tr>
<td>Xiaomi Redmi N3</td>
<td>Smartphone</td>
<td>Android 6.0</td>
<td>v4.1+LE</td>
<td>✓</td>
</tr>
<tr>
<td>Xiaomi Redmi N2</td>
<td>Smartphone</td>
<td>Android 5.0.2</td>
<td>v4.0+LE</td>
<td>✓</td>
</tr>
<tr>
<td>OnePlus X</td>
<td>Smartphone</td>
<td>Android 6.0.1</td>
<td>v4.0+LE</td>
<td>✓</td>
</tr>
<tr>
<td>EDIMAX EW-7611ULB</td>
<td>BL USB dongle</td>
<td>Linux Kernel 4.15.0-23-generic</td>
<td>v4.0+LE</td>
<td>✓</td>
</tr>
<tr>
<td>Mercedes-Benz C100 (2010)</td>
<td>Car Handfree</td>
<td>Unknown</td>
<td>v2.0+EDR</td>
<td>●</td>
</tr>
<tr>
<td>Kia Sedona (2010)</td>
<td>Car Handfree</td>
<td>Unknown</td>
<td>v2.0+EDR</td>
<td>●</td>
</tr>
<tr>
<td>Mini Cooper JCW (2017)</td>
<td>Car Handfree</td>
<td>Unknown</td>
<td>v2.0+EDR</td>
<td>●</td>
</tr>
</tbody>
</table>

**Experimentation**

To experiment the attack scenarios presented in Subsection 4.3.2 and 4.3.3, we launch the attack scenarios on various Bluetooth devices. These devices were made by different manufacturers and operate different Bluetooth versions and operating systems. We have
discovered that some modern Bluetooth devices are vulnerable to these attacks and can be easily disconnected from a Bluetooth network, whereas other devices seem to be “unintentionally secure” from the CDV. Table 4.2 shows different Bluetooth devices used in our testbed, with respect to their names, types, operating systems, Bluetooth versions, and whether they are affected by the CDV or not. Actually, we have identified three classes of Bluetooth devices:

**Class 1.** Bluetooth devices marked with (●) in Table 4.2 are not affected by this vulnerability. This is mainly because these Bluetooth devices are designed to establish one and only one connection at a time. Once a connection is established, these devices do not accept or respond to any other connection. In fact, when these devices are already paired and connected with other Bluetooth devices, our attacking Bluetooth device displays a “Host is down” message when we try to establish a new pairing-free connection with them. In the experiments, we have considered pairing the *Kirvos DZ09* smartwatch with the *Samsung J7*. Then, by impersonating the smartphone, we have sent a SDP-request to the smartwatch which did not respond back.

**Class 2.** Bluetooth devices marked with (✓) in Table 4.2 are not safe from the CDV. We have successfully conducted the attack scenarios on those devices. In fact, those Bluetooth devices accept more than one ACL-connection at a time from the same source. In the experiments, we have considered pairing the *Kirvos DZ09* smartwatch with the *Samsung J7* smartphone. Then, by impersonating the smartwatch, we have sent a SDP-request to the smartphone, captured the exchanged messages using *hcidump* tool and visualized the messages using *Wireshark*. By analyzing the exchanged packets, we notice that the new pairing-free ACL-connection was successfully initiated by the attacker with the smartphone followed by a certain number of packets related to the SDP. At the end, the connection was terminated by the attacker using a disconnect request. The smartphone replied back with a disconnect response before disconnecting. This resulted in a successful disconnection of
the legitimate smartwatch the Kirvos DZ09 after a short time.

For the role switching, we have performed the attack by first running the SDP dump attack scenario which caused a spontaneous disconnection of the smartphone. After its disconnection, the smartphone tried a re-connection which was successful. This made the smartphone to become the master and the smartwatch became the slave. In fact, by impersonating the smartwatch, which is in slave role and by initiating a pairing-free ACL-connection with the smartphone, the roles were changed. We have noticed that a “role change” packet has been exchanged during the attack. The packet indicated that the roles were changed upon the establishment of the spoofed ACL-connection.

Class 3. The third category marked with (×) in Table 4.2 seems to be “unintentionally secure” from the CDV. We have failed to successfully conduct the attack scenarios on those devices. In fact, the Bluetooth implementations in those devices seem to restrict the number of connections coming from the same remote device to one connection only. When conducting the attack scenario, we have captured the packets that were exchanged during the attack and visualized them using Wireshark. The number of packets captured when attacking this category of devices was considerably fewer (precisely 3 packets) compared to the previous category (55 packets) where the attack succeeded. By analyzing these three packets, we have noticed that those devices reply to the attacker with an “ACL Connection Already Exists (0x0b)” message when the attacker tries to establish a new spoofed connection with those devices. This message indicates that those devices do not accept a new connection from a Bluetooth device as long as a current connection is running with the same Bluetooth device. In the experiments, we have paired the smartwatch Kirvos DZ09 with the Apple iPad Air 2, then spoofed the smartwatch to launch the SDP dumping attack scenario which failed.

As a first hypothesis, we have assumed that the third category of Bluetooth devices seems to be unintentionally secure. Yet, we have confirmed our assumption to be true after
running other experiments. In fact, we have discovered that those Bluetooth devices are vulnerable to an other type of attack which also affect the Bluetooth availability if the CDV vulnerability is exploited in a different way. In the case of the connection dumping attacks, those devices reply with an “ACL Connection Already Exists (0x0b)” message. By observing this indication of an ongoing connection, we have asked ourselves the following question: “What could happen if the attacker establishes a spoofed pairing-free ACL-connection with those devices before the spoofed device does?” To answer this question, we have conducted two experiments. In the first experiment, we have spoofed the smartwatch Kirvos DZ09 and established a pairing-free connection with two of the those devices (Samsung S8 and Apple iPhone 8). Then, we have tried to connect the legitimate smartwatch to the smartphones, which resulted in a complete connection failure. The smartwatch was not able to connect to the smartphones since the attacker has already established a connection. This constitutes another threat on availability w.r.t. this category of devices. The second experiment was conducted on the devices of the first category, specifically the handfree and the smartwatch. We have spoofed the smartphone Samsung J7 and established a pairing-free connection with those devices. Then, we have tried to connect the legitimate smartphone Samsung J7 with those devices. This resulted in a complete connection failure since the attacker is already connected to them.

4.3.5 Countermeasures

According to the experimental results, we believe that this security flaw is strongly related to following two causes: (1) The Bluetooth security mode 4. (2) The Bluetooth host implementation by different manufacturers. In security mode 4, security procedures are initiated after the physical and logical link establishment. This allows to establish two types of connections, namely, the pairing-based and the pairing-free connection. Also, we have seen that two devices, from different manufacturers, running the same version of the operating system and Bluetooth stack, do not behave in the same way against the attacks.
4.4. BLUETOOTH LOW ENERGY COMPROMISE

This is mainly due to the different implementations of the Bluetooth stack by various device manufacturers. Thus, we recommend that connection handling mechanism in Bluetooth, should be seriously reviewed in the specification and should be correctly implemented by the manufacturers. We recommend a combination of two solutions to mitigate against the CDV vulnerability: (1) require an authentication before physical and logical connection as it is performed in security mode 3 used in Bluetooth versions v2.0+EDR and earlier, and (2) restrict the number of connections coming from the same remote device to one connection at a time. In this way, we believe that we can thwart the presented attacks.

4.3.6 Discussion

In the previous subsections, we have presented a new Bluetooth vulnerability that we have discovered in the Bluetooth security mode 4, called connection dumping vulnerability (CDV). This vulnerability can be exploited to generate attacks that compromise Bluetooth availability. We have generated three attack scenarios and demonstrated the existence of the vulnerability on modern Bluetooth devices. We have assumed that the target Bluetooth devices are set on discoverable mode and can be easily spoofed. We have also demonstrated that by exploiting the CVD, an attacker is able to perform Bluetooth connection deprivation and role switching. We claim that this vulnerability is due to both Bluetooth host implementation by different manufacturers and the Bluetooth specification. We also claim that this vulnerability can be exploited further to launch more sophisticated attacks. Finally, we highlight that the implementation of connection handling must be seriously reviewed. Finally, we have discussed possible countermeasures to address the CDV.

4.4 Bluetooth Low Energy Compromise

For the past five years, BLE (Bluetooth Low Energy) has become more and more popular and adopted in many IoT devices in a variety of fields. Nowadays, it can be found in
high-end smartphones, sport devices, home appliances, vehicles, and medical devices. It has transformed the classic Bluetooth technology into a technology that can be embedded into resource-constrained devices that run on a cell coin battery for months or years. These devices adopt lightweight authentication and encryption protocols to establish secure connections. For example, to connect a Bluetooth smart device, such as a blood sugar monitor, to a smartphone in order to get a notification when the blood sugar level is up so that insulin can be taken, the smartphone and the blood sugar monitor must be paired, i.e., connected and authenticated. Nevertheless, most, if not all, Bluetooth smart devices that are available in the market, use a pairing scheme called Just Works. This scheme allows any device to connect to another device without any “secret-based” authentication. This makes these smart devices vulnerable to a variety of attacks, which threaten the security, privacy, and safety of users.

In this section, we present a vulnerability in the BLE Just Works pairing mode. Then through a practical case study, we evaluate the security of three Bluetooth smart devices: a bikelock, a lightbulb, and a deadbolt. These devices are made from different manufacturers and implement security in a different manner. We show how the Just Works pairing mode makes those devices vulnerable to different types of attacks. We propose a solution to fix the identified vulnerability.

4.4.1 Bluetooth Low Energy

In 2010, the Bluetooth SIG (Special Interest Group) released Bluetooth v4.0+LE (Low Energy), or simply BLE [Blu18a]. This new technology includes two sub-specifications: Bluetooth smart, also known as BLE single mode; and the Bluetooth smart ready, also known as BLE dual mode. These two sub-specifications have a completely different physical and link layers. Thus, there exist two different protocol stacks: the BLE dual mode and the BLE single mode as illustrated in Fig. 4.18. BLE dual mode implements both protocol stacks, the classic Bluetooth stack, which is used in Bluetooth v1.1 to Bluetooth v3.0+HS
4.4. BLUETOOTH LOW ENERGY COMPROMISE

(High Speed); and the Bluetooth smart stack. Bluetooth devices which implement only the Bluetooth smart stack, i.e., single mode, are not retro-compatible with classic Bluetooth devices [Blu18a].

Bluetooth low energy divides the spectrum into forty channels each of 2MHz bandwidth. Three channels out of the forty (37, 38, and 39) are used for advertisement and connection establishment. The remaining channels are used for data exchange during a communication.

BLE stack can be divided into three layers: controller, host, and application. The controller layer deals with transmission and reception of radio signals. The host layer encompasses all the modules needed to provide applications with diverse APIs (Application Programming Interfaces) to communicate with remote Bluetooth devices through the radio. Finally, the application layer groups all Bluetooth smart applications. The host layer is composed of multiple modules. The L2CAP (Logical Link Control and Adaptation Protocol) is used for protocol multiplexing, packets segmentation and reassembly, quality of service, and group abstractions. The SM (Security Manager) defines all cryptographic functions needed to provide security services. The GATT (Generic Attribute Profile) defines a set of standard profiles that specify how messages are constructed, formatted, and exchanged between two Bluetooth devices. The GAP (Generic Access Profile) defines four BLE roles in which a device can operate: observer (can only receive traffic), broadcaster (can only advertise),

![Diagram of Bluetooth stacks](image-url)
peripheral (can accept connections from other devices), and the central (can initiate connections with peripherals) [Blu18a]. The peripheral and the central are classically known by slave and master, respectively.

The set of profiles that are defined by the GATT module are used to develop smart applications. Each profile defines a set of services. Each service is identified by a unique identifier UUID (Universal Unique Identifier). A service includes one or more characteristics. Each characteristic is identified by a value known as handle. A characteristic contains one or more properties, one value, and one or more descriptions. For example, a smart application running on a central device, such as a smartphone, can use the **Battery service** to monitor the level of the battery in a remote Bluetooth peripheral device.

In order to communicate, Bluetooth devices have to be connected and authenticated to each other. This is performed during an authentication procedure called pairing. The pairing procedure allows two Bluetooth devices to authenticate each other and negotiate on a set of security parameters to derive a master key called link key, or LTK (Long Term Key) in BLE. This key is stored and used to encrypt all future communications between a pair of Bluetooth devices. BLE defines two main pairing modes: legacy pairing and SC (Secure Connections). The legacy pairing applies the SSP (Secure Simple Pairing), which is used in classic Bluetooth (from Bluetooth v2.1+EDR to v4.1+LE), but without ECDH (Elliptic curve Diffie-Hellman) [DH76]. The Secure Connections upgrades SSP since Bluetooth v4.2+LE (and Bluetooth 5.2). It uses ECDH in BLE, longer keys, and provides data integrity. In BLE legacy pairing, only three association modes are possible: Just Works, Passkey Entry, and Out of Band. In BLE Secure Connections, a fourth mode called Numeric Comparison, is added. Besides Numeric Comparison, none of the previous association modes provides protection against passive eavesdroppers. BLE employs AES\textsuperscript{20} 128-bit in CCM\textsuperscript{21} mode for data encryption. In what follows, we only consider legacy

\textsuperscript{20}AES: Advanced Encryption Standard.
\textsuperscript{21}CCM: Counter with Cipher block chaining - Message authentication code.
pairing, i.e., Bluetooth v4.0+LE and v4.1+LE.

![Attack Tree for Attacking Bluetooth Smart Devices](image)

Figure 4.19: An attack-tree for attacking Bluetooth smart devices, where ○ represents an attack and ○–○ indicates an attack refinement. A refinement can be a conjunction (And), disjunction (Or), or a sequential conjunction (Then).

### 4.4.2 Attacking Bluetooth Low Energy

BLE (Bluetooth Low Energy) has been a hot subject of security attacks. Many vulnerabilities have been discovered and various attacks have been conducted and reported in the literature as well as in the hacking conferences. For example, in [Spi12], the authors showed how BLE can be easily sniffed using affordable hardware, such as Ubertooth One [Gre09]. In [Rya12], the authors demonstrated fundamental weaknesses in the key exchange protocol that is adopted in BLE. Ray et al., [RRO+18] and Lonzetta et al., [LCJ+18] discussed general attacks on BLE. With respect to Bluetooth smart devices, Ryan [Rya12] demonstrated attacks on heart rate monitors. Rose et al., [RR16] have discussed how they hacked twelve smart locks out of sixteen. Cauquil [Cau16] developed a MITM (Man In The Middle) framework, called Btlejuice, to conduct MITM attacks on smart padlocks, a robot, and a blood GMS (Glucose Monitoring System). Jasek et al., [Jas17] demonstrated through different attacks, such as MITM attack, replay attacks, and reverse engineering of Bluetooth mobile applications, how they have managed to hack and unlock different types of
smart padlocks. Tan et al., [Tan18] presented how they have managed to hack bicycle locks used for bike sharing applications in Singapore. Zhang et al., [ZL17] conducted attacks on wearable devices, in particular, wristbands, and have shown how vulnerable these devices are. Similar work in [CHM+14; AWH+15], conducted attacks on the same types of devices, i.e., wristbands, by reverse-engineering the mobile applications that are used to connect and control the devices. Also, many vulnerabilities as well as privacy issues have been discussed in [CFC16; HPK16], where authors have shown that some smart devices, such as keyboards, fitness trackers, and heart rate monitors, do not employ encryption at all and all confidential and private information are sent in plaintext over the air. Gullberg [Gul16] have presented DoS attacks on BLE.

In the following paragraph, we discuss the attack anatomy that most attackers follow to generate attacks against BLE devices.

To conduct an attack on BLE devices, an attacker usually follows the attack steps described by the attack-tree\(^{22}\) of Fig. 4.19, where \(\bigcirc\) can be a final goal, a sub-goal, or a basic attack, depending at which level the node \(\bigcirc\) is situated. \(\land\) refers to conjunctive refinement (And) or sequential conjunctive refinement if there is an arrow (Then), and \(\lor\) refers to disjunctive refinement (Or). The steps are as follows:

**Step 1.** The attacker starts by capturing BLE traffic, and more interestingly, capturing packets related to a target communication. Capturing a BLE communication requires the attacker to capture the `connect_request` packet that is sent during a connection establishment. This packet contains all synchronization parameters, such as the hop increment, the hop interval, timeout, window size, and the access address, needed to generate the hop sequence [Blu18a]. Knowing the hop sequence, an attacker can follow up a complete ongoing BLE communication. If the attacker misses this packet, it can apply other techniques, such as the one discussed in [Rya12].

---

\(^{22}\)Attack-trees [Sch99] are graphical security models used to represent possible attack scenarios on a given information system in a user-friendly way.
Step 2. Once the attacker has captured enough packets, it connects to the target device. As most Bluetooth smart devices employ the Just Works pairing mode, the attacker will easily succeed this step.

Step 3. Now that the attacker is connected, it can start replaying previously captured packets or generating new packets based on the captured ones. The attacker just needs to know which service (i.e., UUID) and characteristic (i.e., handle) it targets. As part of the BLE protocol, the attacker can request the smart device to reveal all services and characteristics that the device hosts. Once the attacker identifies the targeted service and characteristic, it generates read and write commands to perform unauthorized readings and writings on the smart Bluetooth device.

4.4.3 Just Works Vulnerability in BLE

In the following paragraphs, we discuss the features of the Just Works pairing mode in BLE and how attackers can take advantage of these features to generate attacks. We present a vulnerability that can be exploited to abuse Bluetooth smart devices functionality. The Just Works pairing mode is a pairing procedure that is used when at least one of the Bluetooth devices to be paired with, does not have any input and output capability. In BLE, it is the most adopted pairing mode, as almost all Bluetooth smart devices, do not have any input and output capability such as lightbulbs, smart locks, fridges, keyfobs, heart rate monitors, and other devices. The main feature of this pairing mode is that it does not require any authentication to complete a pairing procedure. Technically, there is a kind of meaningless authentication, where both devices generate a shared STK (Short Term Key) by proving to each other the possession of a shared TK (Term Key), where this TK is by default set to \textit{0x00} in this pairing mode. The STK is then used to exchange the LTK (Long Term Key) \cite{Blu18a}. In few words, anybody can connect to any BLE device that uses Just Works. To provide security, Bluetooth application developers along with Bluetooth
smart device vendors, use the BLE security specifications and follow the Bluetooth SIG recommendations to design and implement a proprietary high-level security protocol to secure their applications and Bluetooth smart devices. Thus, the authentication procedure of any application that is developed to connect and control a Bluetooth smart device, usually goes through two phases: (1) The standard Just Works phase. (2) The application-based authentication phase. The latter phase is not always implemented. Some vendors invest some money to implement the second phase and make their applications and BLE devices secure, whereas other vendors leave the pairing as is, i.e., not secure.

Another feature in BLE is that most, if not all, Bluetooth smart devices, which operate as peripherals, are restricted to establish one and only one connection at a time with a central device. This can be interpreted as a security vulnerability that can affect Bluetooth availability. As anybody can connect to these devices, and as these devices accept only one connection at a time, an attacker can establish a connection with these devices and deprive legitimate users from connecting to them and using their services as illustrated in
4.4. BLUETOOTH LOW ENERGY COMPROMISE

the MSC\textsuperscript{23} of Fig. 4.20.

4.4.4 BLE Device Security Case Study

To study the security of BLE devices and to demonstrate the impact of the vulnerability that we have presented in the previous section, we consider three use cases. The first use case consists of a Bluetooth \textit{bikelock}, where the owner of a bike uses a mobile application installed on its smartphone to connect to the Bluetooth lock and unlock it. The second use case consists of a smart Bluetooth \textit{lightbulb}, where the owner uses a mobile application to switch the \textit{lightbulb} ON and OFF, and adjust its brightness. Finally, the third use case consists of a home Bluetooth \textit{deadbolt}. The owner uses a mobile application to connect to the \textit{deadbolt}, unlock it to open the door, and gets into the house.

\textbf{Attacker Environment.} To generate attacks on BLE devices, we use different types of tools (viz., Fig. 4.21). We use three hardware components: an \textit{Ubertooth One} \cite{Gree09}, a Bluetooth dongle, and a high-gain antenna. \textit{Ubertooth One} is a hardware component designed to perform Bluetooth experimentation, in particular, on BLE. The Bluetooth dongle is a dual band dongle (\textit{LM Technologies LM1010}), which allows us to establish connections with Bluetooth smart devices. The antenna is a 2.4GHz \textit{Yagi} antenna with a high gain of 25dBi. For software components, we use three software tools: \textit{Ubertooth btle}, \textit{hcitool}, and \textit{gatttool}. The \textit{Ubertooth btle} is a software utility that is used along with \textit{Ubertooth One} to capture BLE traffic and follow an ongoing BLE communication. \textit{Hcitool} is part of the \textit{Linux BlueZ} Bluetooth package. It is used to communicate with the local Bluetooth controller of the computer and perform a multitude of operations, such as scanning for nearby Bluetooth devices. Finally, the \textit{gatttool} software utility, which is used to establish connections with Bluetooth smart devices and send read and write commands to the devices to read and write the characteristics (values) of a given target service.

\textsuperscript{23}MSC (Message Sequence Chart) is a graphical language for the description of the interaction between different components of a system. This language is standardized by the ITU (International Telecommunication Union).
In the next subsection, we conduct attacks on the three Bluetooth smart devices, following the attack tree of Fig. 4.19. We perform attacks against data confidentiality, user authenticity, data integrity, and device availability. We present these attacks according to the classification that we have proposed in Chapter 3: Interception attacks cover attacks on data confidentiality, fabrication attacks contain attacks on user authenticity, modification attacks cover attacks on data integrity, interruption, and domination attacks include attacks on device availability. In this way, we cover all four fundamental security services, also known as information assurance pillars, defined in the DoD Information Assurance Certification and Accreditation Process [DoD02].
4.4. BLUETOOTH LOW ENERGY COMPROMISE

Bicycle Lock

In this first use case, we consider the Bluetooth NULOCK bikelock from NuVending\textsuperscript{24}. This lock is equipped with a braided steel cable of 47 inches, a 110dB alarm protection, and costs around $40. It runs Bluetooth v4.0+LE (Low Energy). It is controlled through a mobile application called Nulock (had 1,000+ downloads on Play Store in June 2019), which allows the owner of the bike to lock and unlock the bikelock using BLE and by just standing close to the lock.

Interception Attack. To intercept specific BLE traffic, we first start by detecting nearby Bluetooth smart devices, in particular, the bikelock. To this end, we use a Bluetooth dongle (LM Technologies LM1010) plugged in a laptop (HP Probook 6065b) that runs Linux (Ubuntu 16.04 LTS). Then, we use the hcitool software utility to communicate with the dongle and perform operations, such as scanning for nearby Bluetooth devices. We detect the bikelock and obtain both its address (f8:36:9b:48:09:d9) and user-friendly name (smartlock). The hcitool command sends scan_requests to detect Bluetooth smart devices, which reply back by a scan_response. Next, we use Ubertooth One along with Ubertooth-btle software utility to eavesdrop any communication that involves the bikelock. Last, to display the captured packets and read their contents, we use Wireshark with BLE packet dissector. We have activated the Bluetooth interface of the smartphone and have stood next to the lock. The connection between the smartphone and the bikelock has taken place and has been successfully intercepted by Ubertooth One, which has managed to intercept the connect_request packet, compute the hop sequence, and follow up the communication.

We have executed some commands, such as changing the lock configuration using the mobile application. This has generated write-command packets sent from the smartphone to the bikelock. Ubertooth One has managed to follow up and intercept those packets.

\textsuperscript{24}NuVending of NUNET is an American company specialized in designing and selling innovative IoT products.
Interestingly, we have tried to interpret the content of one of the packets and found a value (`a171727374313301`) being transmitted as a value for a characteristic which handle is `0x0025`. After decoding that value, it turns out that part of that value (underlined) corresponds to the ASCII code for the password `qrst13` which the user has initially set up. Now that the attacker knows the password, it can use it to bypass the authentication mechanism that is used in the mobile application and unlock the lock. The attacker just has to download the application on its smartphone, set up the password `qrst13`, stand near the bike, unlock the lock, and cycle the bike away. Also, the attacker can adopt a more sophisticated approach to eavesdrop the communication between the smartphone and the `bikelock`. For example, it can use a drone to fly over the bike while the owner is unlocking it. The drone intercepts the packets and flies back to the attacker. Notwithstanding, as drones make noises, another passive alternative consists of using a high gain directional antenna, such as the Bluegun [Her04b], to increase the interception range. The attacker can stand far away from its victim while the latter is unlocking its bike and capture the packets without being noticed. We have actually performed this attack using a 5dBi omnidirectional antenna by standing **100 meters** away from the `bikelock` at Queen’s university campus. Note that we have observed that this `bikelock` uses encryption only at the beginning when the owner sets up the initial password (probably to send the password to the `bikelock`). After that, all communications are in plaintext, including the password.

**Fabrication Attack.** Knowing the password and the Bluetooth device address, we have tried to connect and interact with the `bikelock` using the `gatttool` software utility that runs on a Linux system (Ubuntu 16.04 LTS) of a laptop. We have observed that the laptop gets paired with the `bikelock` using Bluetooth Just Works mode and gets disconnected after few seconds. This is mainly due to the application protocol that runs on the `bikelock`. The `bikelock`, as a BLE peripheral, expects the laptop, as a BLE central, to send some information to ensure that it is talking to the mobile application that was developed for communicating with the `bikelock`. This brings us to analyze the packets
that have been captured by Ubertooth One and to find a pattern that would allow as to convince the bikelock that it is talking to the Nulock mobile application. Interestingly, we have found that the mobile application sends a write request that contains a 20-byte value (a1373431363839099278026b1364b0770d6e237e). We have observed that the first most significant eight bytes (underlined) are fixed for each connection, whereas the remaining bytes vary by changing the password. Thus, we have tried to connect again to the bikelock and send a write command to the handle 0x0025 with the same twenty-byte value. We have observed that in this time the bikelock has not dropped out the connection. This means that the twenty-byte sequence is the right value to send in order to keep the connection with the bikelock running. Next, we have sent a write command to the handle 0x0025 with the learned password and Poof! the bikelock got unlocked.

![Image](image_url)

**Figure 4.22:** Mobile application interfaces after BLE attacks (from left to right): (a) Nulock for the bikelock and (b) Pulse for the lightbulb.

**Modification Attack.** We have thought of the possibility of illegally changing the configurations of the bikelock. One important configuration is the password. We have set up Ubertooth One to follow the connection between the bikelock and the legitimate user while changing the password using the mobile application. After we have changed the
password, we have analyzed the packets and found a write command that contains both passwords, the old one and the new one, separated by one byte (0x07). Thus, we have paired the laptop again with the bikelock, sent the twenty-byte sequence to hold the connection, and sent a slightly modified write command to change the password. We have just swapped the old password with the new one in such a way so that the new password becomes qrst13. The bikelock has replied with a message indicating that the command was successfully executed. This means that the integrity of the stored data in the device has been successfully corrupted.

**Interruption Attack.** In order to check the impact of the write command that we have sent during the modification attack, we have used the mobile application and tried to legally connect the bikelock and unlock it. The mobile application has popped up a message informing that the password has been changed and that the bikelock has to be removed from the list as illustrated in Fig. 4.22.(a). In this way, the legitimate user has been locked out from its bike. If the user forces the lock, the lock will trigger its 110dB alarm. The alarm will not stop until it runs out of battery, which will take hours. At this point, the bike lock requests the user to input the new password which is only known by the attacker. This approach can be used to set up a ransomware that performs the same attack and locks users out and forces them to pay a certain amount of money to unlock their items.

Furthermore, to evaluate the impact of connection deprivation discussed in Subsection 4.4.3, we have established a connection with the bikelock and used that 20-byte value to hold the connection. Next, using the legitimate application, we have tried to connect to the bikelock which failed as shown in Fig. 4.23.(a). An attacker standing far away from the bike, can perform this attack using a high-gain directional antenna and deprive a legitimate user from unlocking its bike.

**Domination Attack.** As the password constitutes the security single-point of failure in the bikelock, we can consider that we have dominated the security of this system as we
have managed to disclose the password during the interception attack.

**Smart Lightbulb**

In this second case, we consider the Bluetooth *pulse solo lightbulb* from Sengled. The *lightbulb* comes with an integrated JBL speaker to allow users to broadcast audio data on the *lightbulb* from their smartphones. It costs around $70. It runs Bluetooth 4.0+LE (Low Energy). Users can download and install a mobile application called *Pulse* (had 100,000+ downloads on *Play Store* in June 2019) to connect their smartphones to the *lightbulb* and control it remotely. They can use the application to switch the *lightbulb* ON and OFF, and adjust its brightness.

**Interception Attack.** Similar to the *bikelock*, we have started by detecting the *lightbulb* and learning its Bluetooth device address (08:7c:be:36:e0:49) as well as its user-friendly name (C01-A66_Pulse Solo) using *hcitool* utility. The *lightbulb* periodically broadcasts advertisement packets on channel 37 to indicate its presence in the neighborhood. The mobile application, i.e., *Pulse*, does not require any password from the user. Thus, this time we do not have to intercept any confidential information, such as a password. Still, we are interested in intercepting **write-commands** that the application sends to change the status and configurations of the bulb such as the commands that modify the brightness of the *lightbulb* and the ones that turn the light ON and OFF. We have generated commands to switch the light ON and OFF and to modify the *lightbulb* brightness. At the same time, we have set up *Ubertooth One* along with *Ubertooth-btle* software to intercept the traffic that is generated by the commands that are executed from the smartphone. We have captured the packets that were used to modify the brightness of the *lightbulb*. Those packets carry **write-commands** to the handle 0x0017 with a 168-bit value (7e800000000001000100000000644000007e). The byte that contains the value 64 (underlined) represent the brightness, which is equal to 100% in this case. This is the

---

25 Sengled is an international company specialized in smart lighting products.
same write-command that is used to switch the light ON. Therefore, when switching the lightbulb OFF, there will be a write-command to the handle 0x0017 with a value equal to 7effffff0000000010001000000ff00400007e.

**Fabrication Attack.** The Pulse application does not implement any authentication mechanism. Anybody can install the application and connect to the lightbulb and remotely switch it ON and OFF. An attacker can take over the lightbulb by connecting to it and illegally executing commands, such as switching the light ON and OFF and changing the light brightness.

**Modification Attack.** To breach the integrity of the lightbulb, we have tried to illegally change the configurations of the lightbulb. Knowing the value to be written on the handle 0x0017, we have connected the laptop to the lightbulb and have executed write-commands on the bulb. By setting the brightness byte to 0x00, we have managed to turn the lightbulb OFF, and by setting it to 0x64 we have turned the lightbulb ON with 100% brightness. Interestingly, since the highest value to be represented on one byte is 255, we have tried to execute a write-command with a value set to 0xff. The command was executed with success. Then, to check the impact of the last command, we have used the mobile application to connect to the lightbulb and found out that the current displayed brightness is 255% as illustrated in Fig. 4.22.(b). Hopefully, nothing dangerous has happened after we have boosted the brightness beyond its maximum for few seconds. However, we were pretty confident that if we repeatedly and rapidly switched the light ON and OFF with that brightness for a longer time, the light may become faulty.

**Interruption Attack.** In this scenario, there are three ways to cause a DoS on the lightbulb: (1) Sending a steady-stream of write-commands to the handle 0x0017 with a value set to 0x00 to switch OFF the light definitely. (2) Sending a steady-stream of write-commands to the handle 0x0017 with a value set to 0xff and see what will happen
after sometime. (3) Establish an illegal connection with the lightbulb and deprive legitimate users from connecting to it and using it. We have performed this third attack option and tried to connect using the application as legitimate users, but the application popped up the message shown in Fig. 4.23.(b). We have installed the lightbulb inside an office in the 6th floor of a building at Queen’s university. By staying 85 meters away outside the building inside a car while it was raining, we have managed to take over the lightbulb. We note that the factory operational range of the lightbulb is set to 10 meters.

**Domination Attack.** Based on the previous attacks, this lightbulb does not have security at all. It’s security can be considered as dominated with a minimal effort.

**Smart Deadbolt**

In this last case, we consider the Bluetooth Sense deadbolt from Schlage\(^\text{26}\). The deadbolt is embedded with Wi-Fi as well as Bluetooth smart technology. It costs around $250. It runs Bluetooth 4.1+LE (Low Energy). Users can download and install a mobile application called SchlageHome (had 50,000+ downloads on Play Store in June 2019) to connect their smartphones to the deadbolt and unlock it. The deadbolt can also be connected to the Internet through its Wi-Fi interface and get unlocked from anywhere through Internet.

**Interception Attack.** We have captured the communication between this home lock and the smartphone using Ubertooth One. Then, using Wireshark we have displayed the packets that have been intercepted and tried to interpret them. It turns out that this deadbolt uses AES-encryption with data freshness. This means that the packets can neither be read (decrypted) nor replayed. We could not apply simple attacking tools to disclose any confidential information. We have also tried to capture the pairing messages in an attempt to crack the LTK key, but have observed that the pairing is actually not happening over BLE. We have noticed that the application requires the Wi-Fi interface to remain active

\(^{26}\)Schlage is an American lock manufacturer founded in 1920. It produces high-security key and cylinder lines, Primus, Everest, and Everest Primus XP.
4.4. BLUETOOTH LOW ENERGY COMPROMISE

while the pairing is performed. We have concluded that this **deadbolt** uses the Out-of-Band pairing mode to accomplish a secure pairing. We have also reverse-engineered the mobile application to understand its code, but the code was too complex and appeared to be well designed and implemented. Hence, this **deadbolt** requires more sophisticated approaches that we leave for future work.

**Fabrication Attack.** We have tried to replay the packets that we have captured and observed that the **deadbolt** cut off the connection right after receiving our replayed packets. This means that the **deadbolt** is implementing a kind of replay protection to discard any replayed or unexpected packets.

**Modification Attack.** We have observed that the **deadbolt** discards most of the packets that we have replayed and injected. However, we have managed to successfully change the values of some characteristics, such as the one identified by the handle `0x0023`. The impact of changing the value of the characteristic `0x0023` is discussed in the next paragraph.

**Interruption Attack.** Although this **deadbolt** has proven to have security implemented, it does not resist against the connection deprivation attack discussed in Subsection 4.4.3. As the **deadbolt** accepts only one connection at a time, an attacker can easily occupy the connection and take over the **deadbolt**. Hence, any legitimate user who tries to connect and unlock the **deadbolt** to get home, will be locked out. We have generated this attack and tried to connect using the legitimate application. The latter has displayed the screenshot shown in Figure 4.23.(c). Nevertheless, we have observed that the connection is dropped after 60 seconds. We have tried to understand the reason behind that and have found that the **deadbolt** periodically disconnects from the application after each 60 seconds and then reconnects. We have considered that as a security measure against eavesdroppers since re-establishing a new connection results in a new hop sequence. Thus, to perform the connection deprivation attack, the attacker needs to reconnect after each 60 seconds. However, by applying some fuzzing, we have found that we can successfully change the value
of the handle 0x0023, and that if we change that value to 0xff few seconds before those 60 seconds elapse, we can keep the connection for another minute. Hence, the attacker just has to send a write command to the handle 0x0023 with a value 0xff, say each 55 seconds, to not get disconnected and indefinitely deprive legitimate users from unlocking the deadbolt. We have used our high-gain antenna and managed to successfully perform this attack from inside a car that we have parked 425 meters away from the deadbolt. We emphasize that it was raining when we performed this attack from that distance.

![Mobile application interfaces after connection deprivation (left to right): (a) Nulock for bikeclock, (b) Pulse for lightbulb, and (c) SchlageHome for deadbolt.](image)

**Domination Attack.** We have failed to crack the password for this deadbolt. We can claim that its security is good and the device can be trusted.

**Mitigating BLE Deprivation Attacks**

As a mitigation to the connection deprivation attack, Bluetooth smart devices should be designed to accept multiple connections at a time. This will for example allow two or more users to control a device at the same time. Notwithstanding, we boldly forewarn that connection handling must be implemented correctly to avoid falling back into the connection
dumping vulnerability (cf., Subsection 4.3.1).

4.4.5 Discussion

In this section, we have discussed BLE security, in particular, when the Just Works is used. We have shown through a practical case study of three different Bluetooth smart devices, how vulnerable certain devices are. We have also discussed the connection deprivation vulnerability that can be exploited to deny legitimate users from using their own devices and proposed a mitigation technique to this attack as well.

We cannot force Bluetooth smart device vendors to invest for security. Still, we can warn users about the possible risk of buying insecure devices and choosing convenience over security, privacy, and safety. BLE devices are predicted to be nearly one-third of the forty-eight billion IoT devices in 2021. If BLE security is not considered seriously, by 2021 we will have billions of vulnerable devices in the market. Those devices will be used in IoT applications and make IoT vulnerable as well.

4.5 RFID Compromise

RFID passive keyless entry and start (PKES) systems have been widely deployed in modern cars. They allow drivers to unlock their cars or start their engines by just standing within the proximity of their cars or being inside the car along with an RFID keyfob in their pockets. Although these systems have many advantages over their predecessors and are considered more secure, they are nowadays subject to a new type of attack, known as a relay attack (cf., Subsection 3.6.1 of Chapter 3). In a relay attack, an attacker tries to fool a car into believing that its associated keyfob is located in its proximity, where in reality it is not the case. This is performed by relaying signals from the keyfob to the car and vice-versa to unlock and start the car. Due to this attack, hundreds of cars have been stolen in many countries. Since then, car manufacturers as well as insurance companies have been
experiencing an endless nightmare. Researchers have also been working hard into proposing solutions, mainly based on distance-bounding protocols and sensing technologies, to counter relay attacks, but none of the solutions offered with a fundamental mitigation.

In this section, we propose the application of FHSS (Frequency Hopping Spread Spectrum) transmission technique as a physical-layer countermeasure to mitigate relay attacks. By hopping from one frequency to another, within a wide bandwidth, and following a per-session secret-shared frequency hopping sequence, we believe that the communication between a car and its associated keyfob can be hidden from the attackers as long as the attackers are not aware of the hopping sequence.

4.5.1 Car’s Entry Systems Overview

To prevent car theft, cars have been traditionally equipped with an authorization system that allows a driver to unlock/lock a car and start/stop its engine. The system relied on a metallic or alloy key (viz., Fig. 4.24 (a)) that the driver had to physically insert into the door lock (and the car’s ignition switch) to unlock/lock the car (and to start/stop the car engine) by turning the key clockwise/anticlockwise. Such a system was vulnerable to key copying attacks. The attacker needed to obtain a key for a short time to make a copy at a locksmith. Also, certain car models used to have the same lock for the car doors as well as for the petrol tank cover. An attacker just had to take the cover and get the lock structure to forge a copy of the car’s key.

Later, some car models started using a key that was embedded with an immobilizer chip (viz., Fig. 4.24 (b)) to prevent key copying as well as physical lock bypassing. In early eighties, car manufacturers started thinking about creating a more convenient technology that allows easy access to cars and prevents known classical auto theft attacks. This led

---

27 Breaching car security is not limited to stealing the car itself but also includes breaking into the car to grab anything valuable inside, placing a remote controllable ODB (On-board diagnostics) adapter on the car’s ODB port, or taking off any part needed by the thief, such as car’s stereo, documents, doors, bonnet, or any expensive engine’s parts.
4.5. RFID COMPROMISE

Figure 4.24: Car key models: (a) Classical metallic or alloy key, (b) Car key with immobilizer. (c) RF remote controller key, (d) RFID keyfob.

Figure 4.25: Door handle button (left), the “Start Engine” button (center), and the car door-keypad used on certain American car makes such as Ford, Cadillac, Lincoln, Mercury, and Chevrolet, since the early eighties (right).

to the emergence of keyless entry systems, where the classical metallic key is augmented with a remote controller. Drivers just have to push a button on the remote controller to unlock/lock the car’s locking system. Certain remote controllers have the ability to pop up the trunk, whereas others can get the car’s engine remotely started (viz., Fig. 4.24 (c)).

Nowadays, keyless entry systems are being widely deployed on most, if not all, car models. The current generation of such systems allow drivers to automatically unlock their cars by just standing few feet away from the car and by carrying an RFID (Radio Frequency IDentification) keyfob in their pockets (viz., Fig. 4.24 (d)). The car gets locked as soon as the driver walks away from the car with the keyfob\textsuperscript{28} or after pressing a button on the door handle (viz., Fig. 4.25 (left)). In addition, drivers can start/stop their car’s engines

\textsuperscript{28}For security purpose, certain car models get automatically locked after some seconds, if the driver walks away without locking the car.
by pushing a “Start Engine” button (viz., Fig. 4.25 (center)) and having the keyfob present anywhere near the steering wheel. Such systems are commonly known as Passive Keyless Entry and Start systems, or PKES systems for short. They are deployed on most high-end car models. However, this considerable advancement in keyless entry systems has given birth to more advanced auto-theft techniques. Among these techniques, applying relay attack for auto-theft has been a recent and serious matter of concern for many car manufacturers, insurance companies, and car owners. In a relay attack, attackers collaborate to relay the signals between a car and its associated keyfob by boosting the signals while the car and the keyfob are not in close proximity. In this way, the attackers fool the car into believing that the keyfob is in its proximity and gets unlocked and started.

Due to relay attacks, hundreds of cars, equipped with PKES systems, were stolen. Tracker (a UK vehicle tracking company) reported that 4/5 of all vehicles stolen and recovered by the firm in 2017 were stolen without using the owner’s keys. CCTV footages in the UK showed how prestigious cars (Land Rover [Dai19], Tesla [New19c], Mercedes-Benz [Bir18; Exp18; Exp17], BMW [Dai18], Audi [New19b], and Jeep [Cox20]) were stolen from driveways using relay attacks in less than a minute. In December 2019, CBC News (Canadian Broadcasting Corporation) reported that hundreds of high-end vehicles across Ottawa region (Canada) were stolen since April 2019 [New19a]. Furthermore, a German automobile club (ADAC) tested relay attack on 237 car models from 30 different manufacturers. They found that only 7 cars could not be either unlocked or started. Interestingly, the use of relay attacks for auto-theft has been studied and investigated in the literature and its feasibility has been demonstrated as well. In 2011, researchers [FDC10] managed to unlock and start the engine of ten car models from eight manufacturers using relay attacks. In 2017, a security research team, called UnicornTream, developed a cheap hardware (around $22) to realize a relay attack. The team demonstrated the feasibility of a relay

---

29 Toyota 4Runners, Highlanders, Tacoma pickup trucks and Lexus GX460.
30 See ADAC: https://www.youtube.com/watch?v=0AHSDy6A1VO.
attack from 300 meters [ZYL17]. Furthermore, RFID location-based attacks (which include relay attacks [HMM09; DM07; HPJ07; Han06; KW05a]) were discussed in the literature since early nineties [DGB87; Des88; CLP05b].

Thus far, there is no fundamental countermeasure to completely mitigate relay attacks. Existing countermeasures are either mechanical or technological. Mechanical countermeasures are not practical, not flexible, sometimes not feasible, and not automatic. Technological countermeasures however, are of a great concern in research. This has turned our attention to investigate the situation and propose a fundamental countermeasure to mitigate relay attack.

4.5.2 Car Keyless Entry Systems Security

Vehicles in general and cars in particular have been equipped with keyless entry technology since early eighties. These systems have evolved from typing a PIN code on a door-keypad (first time used in the 1980 Ford Thunderbird, viz., Fig. 4.25 (right)), into remotely pressing a button on a remote controller (first time used in the 1982 Renault Fuego), and nowadays, by standing in the car’s proximity and carrying a wireless keyfob (first time used in the 1993 Chevrolet Corvette). Each of these mechanisms has brought advantages over its predecessor in terms of security, reliability, and flexibility. At the same time, each mechanism is vulnerable to a certain type of attacks. In the next paragraphs, we present each car’s entry mechanism and discuss its germane attacks and possible countermeasures.

**Static Remote Controller Keys.** This mechanism consists of remotely (≤ 35m away from the car) pressing a button on a remote controller (viz., Fig. 4.24 (c) and Fig. 4.24 (d)) to send a signal that carries a factory fixed-code to the car over the UHF radio frequency 315 MHz or 433.92 MHz. The car receives and interprets the signal to execute the requested command, e.g., unlock car, after verifying the correctness of the code as illustrated by the

---

31 Certain car manufacturers, such as Ford and Lincoln, have started adopting the UHF frequency band between 902.375 MHz and 903.425 MHz.
4.5. RFID COMPROMISE

MSC\(^{32}\) of Fig. 4.26. This mechanism is vulnerable to interception and replay attacks. An attacker can intercept the signal using a dedicated hardware (e.g., HackRF One\(^{33}\)) when the driver is unlocking the car and then replay the signal later on to unlock it.

\[\text{Prover} P \quad \text{Verifier} V \quad \text{Shared fixed code} \]

\[\text{Request}\left(\text{fixed code, cmd}\right)_{P \rightarrow V} \quad \text{Verify fixed code} \quad \text{Execute cmd}\]

Figure 4.26: Fixed code-based authentication for car remote entry systems. The notation \(M_{P \rightarrow V}\) indicates a message \(M\) sent from the prover, denoted as \(P\), to the verifier, denoted as \(V\). The \(cmd\) denotes the command to execute, e.g., unlock/lock doors, remote start engine, or pop up trunk.

**Dynamic Remote Controller Keys.** The next generation of remote controller keys focused on essentially mitigating replay attacks. Remote controller keys started adopting the rolling code (hopping code) approach where the remote controller sends a different code sequence each time a command, e.g., unlock doors, is sent to the car. The car and the remote controller are somehow synchronized over a sequence of codes in the sense where the next codes (in case the car has missed some transmitted keypresses) to be used is known by both car and remote controller (viz., MSC\(^{32}\) of Fig. 4.27). This will mainly prevent replay attacks that are successful on static remote controller keys. However, the rolling code has been demonstrated to be vulnerable to rolljaming [Kam15] where the attacker prevents the car driver from unlocking its car by jamming the signal at each attempt. The attacker

---

\(^{32}\)MSC (Message Sequence Chart) is a graphical language for the description of the interaction between different components of a system. This language is standardized by the ITU (International Telecommunication Union).

\(^{33}\)HackRF One is a transmit and receive capable SDR. It has a 10 MHz to 6 GHz operating range and up to 20 MHz of bandwidth. It costs around $300.
records the rolling code signals (i.e., the rolling codes) sent each time while performing jamming. The attacker then unlocks the car using the first rolling code signal making the driver think that it has unlocked the car by keypressing. The attacker uses the next rolling codes to unlock the car later on. Also, it has been recently demonstrated that by replaying previously captured codes, certain car models tend to behave in an incorrect way where their keyfob gets disabled locking out their drivers\textsuperscript{34}. In addition to disabling the keyfob, in some car models, e.g., 2019 Ford Expedition, the rolling code is reset to zero which would allow the attacker to replay the previously captured codes to unclock and lock the car as long as those codes are replayed in their correct order.

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{fig427.png}
\caption{Rolling code-based authentication for cars remote entry systems. The notation \(M_{P\rightarrow V}\) indicates a message \(M\) that is sent from the prover, denoted as \(P\), to the verifier, denoted as \(V\). Also, the notation \(c(t_j)\) indicates the code \(c_{i[n+1]}\) that is sent at time instant \(t = t_j\), and \(cmd\) denotes the command, e.g., unlock car’s door.}
\end{figure}

The fixed-code as well as the rolling-code mechanism are both vulnerable to signal blocking attack (a.k.a., remote interference). In this attack, the attacker jams the signal that is transmitted by the car’s remote controller when locking the car doors. The car remains unlocked if the driver does not physically check whether the car is locked or not.

\textsuperscript{34}See Hack5: https://www.youtube.com/watch?v=k8rN9Q3mBZQ4&ab_channel=Hak5.
Drivers in a hurry tend not to check their cars. The attacker gets into the car once the driver is away and may grab something valuable or plug any adapter into the OBD-II (On-Board Diagnostics II) port for car remote controlling, car tracking, or key reprogramming [Col19].

For a better illustration of this attack, we have used a 2010 Dodge Caravan remote controller and a garage door remote controller, both operating at the same frequency (around 433.90 MHz). The car’s locking-signal is sent at the frequency of 433.92 MHz as illustrated in Fig. 4.28 (left). By holding the lock button on the garage door remote controller while trying to lock the car, the car’s locking-signal gets completely jammed as illustrated in Fig. 4.28 (right). As a consequence, the car remains unlocked. The same experiment has been conducted on a 2010 Kia Sedona (operating on 315 MHz) using a 315 MHz garage door remote controller, viz., Fig. 4.29.

Figure 4.28: The locking-signal sent from a Dodge Caravan’s remote controller at 433.92 MHz (left) and the jammed signal at the same frequency (right).

Figure 4.29: The locking-signal sent from a Kia Sedona’s remote controller at 315.02 MHz (left) and the jammed signal at near frequency (right).
Nevertheless, this attack targets only car models that are not equipped with RFID keyless system, which we present in the next paragraph. In fact, the newest car keyless entry systems have not literally replaced their predecessor systems but rather augmented them with a newer technology providing a higher availability. For example, if a car that uses RFID keyless system has been subject to jamming, the car would automatically get locked as soon as the driver walks away. Also, if a thief tries to lock out the driver from unlocking its car through jamming, the driver would be able to unlock its car by just approaching it and touching the door handle or by pressing a button on the door handle as long as the used RFID frequency is different from the one used for jamming. In the worst case, the driver can pull the metallic key from the keyfob and unlock its car just like in the old time.

**RFID Keyfobs.** To add more security and convenience, hands-free keys appeared in early nineties and started to become more popular in early 2000. In this generation, car keys are transformed into (or augmented with) an RFID-tag (becoming a keyfob, viz., Fig. 4.24 (d)). The car and the keyfob communicate over LF (Low Frequency) RFID radio frequency band which operates between 125 kHz and 137 kHz\(^{35}\). In this system, the keyfob plays the role of a semi-passive RFID-tag, whereas the car plays the role of an RFID-reader. Thus, when the keyfob is approached to the car\(^{36}\) (≤ 35cm around the car), the keyfob circuit gets power-supplied by the car through induction-coupling (or electromagnetic coupling on HF) phenomenon and an optional lightweight authentication protocol is executed\(^{37}\). The car verifies that it is communicating with the correct keyfob and gets unlocked.

Nonetheless, researchers [FDC10; ZYL17] demonstrated through relay attacks that it is possible to fool a car into believing that its associated keyfob is located in its proximity.

---

\(^{35}\)In some keyless entry systems, the RFID communication is asymmetric. The communication from the car to the keyfob is performed over LF 125 kHz band (shorter range), whereas the communication from the keyfob to the car is over UHF band with a frequency of 315 MHz, 433.92 MHz, 868 MHz, or 915 MHz for a longer range.

\(^{36}\)In some car models, the driver has to either press a button or touch a motion sensor on the door handle which lets the car know that the keyfob is around. The car starts broadcasting the signal that supplies power to the keyfob.

\(^{37}\)The mechanism that a passive RFID-tag uses to respond to an RFID-reader by using the reader’s carrier as a power-supplying source is called backscattering.
so that it gets foolishly unlocked although the keyfob is located far away from the car. This happens as the adopted RFID protocol automatically assumes that a keyfob is in close proximity to the car if the car can communicate with the correct keyfob. This has led to the development of RFID distance-bounding protocols [BC93]. These protocols are authentication protocols that in addition to proving the identity of a given party to another, e.g., keyfob to a car, the protocol also verifies the distance that separates both authenticating parties, by measuring the challenge-response time, which is also known as the round-trip time (RTT) (viz., MSC32 of Fig. 4.30). If the estimated distance is $d$ and the round-trip time is $\Delta t$, then it must hold that $d \leq \frac{1}{2}\Delta t \cdot c$, where $c$ is the speed of light. This type of
protocols have mitigated a certain type of relay attacks. Notwithstanding, most cars keyless entry systems used nowadays, implement the authentication protocol in a way where relay attacks are still possible. In present-days, attackers employ signal-amplification relay attack to steal cars. In this type of attack, attackers relay signals from the keyfob to the car and vice-versa by simply amplifying them (viz., MSC of Fig. 4.31).

Figure 4.31: A relay attack on a passive keyless entry system. \( m_i = M_{P \rightarrow V} \) indicates that a message \( m_i \) is sent from the prover, denoted as \( P \), to the verifier, denoted as \( V \). The symbols \( n_V \) and \( n_P \) denote nonces. The symbol \( Sk \) denotes a shared secret between \( P \) and \( V \), where \( \mathcal{E} \) denotes the encryption function (hash function), and \( \delta t \) the empirical challenge-response travel time. The computed round trip time is denoted by \( \Delta t \).

4.5.3 Countermeasures for Relay Attacks in PKESs

With respect to mitigating relay attacks for auto theft, the existing solutions can either be mechanical or technical. Mechanical solutions include keeping the keyfob inside a metallic..
4.5. RFID COMPROMISE

box, freezer, or microwave, covering the keyfob with a foldable-aluminum sleeve (a.k.a., Faraday cage), keeping the keyfob away from the car, disabling the keyfob, parking the car inside a garage, or parking a worthless old car next to the expensive one to block the latter from moving. These solutions are clearly neither practical nor flexible. Technical solutions include either the use of round-trip time-based solution (distance-bounding protocols [BC93]) or sensing technologies. Although distance-bounding protocols are being adopted and recommended as the unique physical-layer countermeasure for relay-attacks, this solution suffers from the following concerns: (1) Most keyless entry systems do not implement a secure distance-bounding algorithm and (2). If they do, an overestimation of the fixed delay would make the keyless entry system vulnerable to reduced-delay relay-attacks.

Sensing-based solutions consists of measuring the temperature, sound, GPS-position, radio-environment (e.g., Wi-Fi and Bluetooth), signal strength, i.e., RSSI (Received Signal Strength Indicator), light brightness, or motion detection. These solutions tend to make the entry system costly as additional sensors are needed. In addition, sensing-based solutions are not always reliable [CKK15; WLZ19c; AC17; CSL18; KP15]. For example, in a temperature-based solution, the temperature around a keyfob that is inside the driver’s pocket is totally different from the temperature that is measured by the car. Also, radio-signals, such as the ones used for GPS-positioning, are not always available (e.g., in underground parking). Moreover, most sensing capabilities can easily be bypassed. The GPS-signal for instance can be spoofed using some SDR-hardware (Software Defined Radio-hardware) allowing attackers to generate fake location signals in the neighborhood and hence fooling any receiver trying to geolocalize itself. Other solutions are based on combining sensing features along with distance estimation or involving other technologies [CSL18; KP15; SC10; WLZ19a].

The idea of changing the frequency during a communication to counter relay attacks was proposed in a patent by the Texas Instruments Inc. [KDR+15]. Their idea consists of using different frequencies during a challenge-response protocol execution between the car and its keyfob. The idea was very generic irrespective of the frequency band on which it
will be used and on which technology it will be adopted. In contrary, we will demonstrate that applying FHSS within the frequency band that is currently used by car PKESs is not that useful. Also, we will show in the next subsection that applying FHSS is worthwhile on certain frequency bands only, hence can only be adopted by some contactless applications. Another patent [C S13] has proposed to reduce the transmission power of the keyfob in asymmetric RFID keyless entry system\textsuperscript{38}. The idea consists of reducing the range of the signal that is sent from the keyfob to the car (i.e., UHF-signal) that does not need to be amplified by the thieves. However, this does not entirely mitigate the attack, as it will fail to mitigate the attack if the thief with an amplifier is within that reduced range, which will allow the thief to boost the keyfob signal to the other thief which is beside the car.

**Frequency-Hopping Spread Spectrum**

Frequency-Hopping Spread Spectrum (FHSS) is a technique for transmitting radio signals by rapidly varying the frequency (channel) of the carrier following a predetermined pseudo-random sequence of frequencies in a given radio band. In this case, the transmitter as well as the receiver will have to synchronize over the sequence of frequencies to be able to know on which frequency they should be tuned on to send and correctly receive the signals. This technique avoids interference and jamming attacks. It also provides a physical-layer encryption to eavesdroppers as long as the sequence of frequencies is kept secret between the transmitter and the receiver. Military services have been using this transmission technique since World War I as a physical-layer security mechanism. In FHSS, the available frequency band is divided into channels. The transmitter as well as the receiver hop from one channel to another at a predefined hopping-rate.

**4.5.4 Mitigating Relay Attacks using FHSS**

In this subsection, we propose the use of FHSS (Frequency Hopping Spread Spectrum) transmission technique on passive keyless entry systems as a physical-layer countermeasure
to mitigate relay attacks. The intuition behind using FHSS to mitigate relay attack is to move and switch (hide) the radio domain of the communication between the car and its keyfob over time with respect to the attacker. In fact, for a successful relay attack, the following assumptions must hold: (i) The car-keyfob and the attacker’s time domain must be the same, which means the attacker needs to be relaying the signals at the same time when the communication between the car and its keyfob takes place (no delays). (ii) The car-keyfob and the attacker’s location domain must be the same, which means the attacker has to be physically near the car and keyfob when the car-keyfob communication happens. (iii) The car-keyfob and the attacker’s radio domain must be the same, which means the attacker should be tuned on the same frequency over which the car and its keyfob operate (as the attackers need to amplify the signals that are sent from the car to the keyfob and optionally the ones sent from the keyfob to the car). In addition to that, the hardware that attackers use (i.e., amplifiers) have some hardware characteristics, such as the operational frequencies, the acquisition bandwidth, the gain, the sensitivity, the related-delay, the physical size, and the price. Essentially, the hardware has to be operational on the car-keyfob frequency, has a wide acquisition bandwidth to capture the signals on a wideband, a high gain and good sensitivity to minimize the noises, and finally be portable and cheap. We note that these amplifiers are available on the black market and generally expensive, but not compared to the price gained after selling multiple stolen high-end cars. Hackers use cheap devices (usually less than $100) and turn them into extremely expensive tools for thieves to conduct relay attacks.

Therefore, we apply FHSS to enforce the communication (including the authentication protocol) between the car and the keyfob to occur on multiple channels (radio domain). At a given time, the car and the keyfob are tuned over a frequency for a very short time before hopping to another frequency following a previously shared secret pseudo-random sequence

---

38In asymmetric PKESs, where the car-keyfob frequency is not the same in both directions, attackers relay the LF signals that are sent from the car to the keyfob (short range signal) and leave the UHF signals that are sent from the keyfob to the car (long range signal $\approx 100m$) [FDC10].
4.5. RFID COMPROMISE

Figure 4.32: Set of hardware materials used for the experiments (from left to right, from top to bottom): The NooElec Ham it up module (the black rectangle module), an LF RFID reader (125 kHz), a UHF RFID reader (865 MHz - 928 MHz), a 433 MHz keyfob, a 315 MHz keyfob, an LF RFID tags (card), UHF RFID tag (card), UHF RFID tag (patches), NooElec smart SDR, and two antennas compatible with the SDR.

of frequencies. In this way, an attacker who tries to conduct a relay attack needs to know on which frequency the car and the key are communicating in order to boost the signals. As the attacker does not know the sequence of frequencies, it will be able to boost only some signals. If the car and the keyfob are not in proximity, failing to boost a single signal at a given time would result in a disconnection, which brakes the relay attack. Thus, as long as the sequence of frequencies is kept secret, an attacker will not be able to tune its hardware to receive the signals and amplify them so that the relay attack succeeds. Furthermore, as we believe that relay attack is a physical-layer attack, proposing a fundamental mitigation solution would only be possible through a physical-layer countermeasure.

Frequency-Hopping Spread Spectrum in RFID

Many RFID standards including EPC\textsuperscript{39} Gen2 have incorporated FHSS as part of their standards [EPC08; EPC13], essentially to prevent two UHF RFID-readers from interfering

\textsuperscript{39}EPCGlobal industry association defines four classes of UHF RFID tags. Class 1 is for passive tags, Class 2 enriches Class 1 with more memory and add cryptography, Class 3 is semi-passive tags, and Class 4 is for active tags.
on each other. The standard EPC Gen2 (passive UHF RFID) specifies the operational frequency for the tags to be between 860 MHz and 960 MHz. Usually, depending on the radio regulatory agency of each country, a smaller frequency range between 860 MHz and 960 MHz is used. For example, the allocated frequency range (bandwidth) for UHF-RFID is 902.0 MHz to 928.0 MHz in the US (865.6 MHz to 867.6 MHz in Europe) [BPT+17]. Thus the use of FHSS should only be within the regulated radio bandwidth. Notably, the use of FHSS by UHF-tags makes sense as the available bandwidth is large enough [BPT+17].

Current PKES systems commonly operate at a lower frequency (LF RFID from 125 kHz to 137 kHz). The available bandwidth is not that wide making it not suitable for FHSS application. In fact, the RF-hardware that an attacker uses may have an acquisition bandwidth which is larger than the hopping bandwidth. For example, the HackRF One SDR tool has an acquisition bandwidth of 20 MHz. RFID signals sent by hopping within a 12 kHz-bandwidth (|137 kHz - 125kHz|) will be certainly intercepted by the attacker’s hardware. Moreover, when a low frequency \(f_0\) is used, its harmonics, i.e., \(2 \cdot f_0\), \(3 \cdot f_0\), \(4 \cdot f_0\), ..., \(n \cdot f_0\) are very close to each other with respect to an interception hardware with a wide acquisition bandwidth. This allows an attacker to detect the signal even though its RF-hardware filter is not sharply tuned on the frequency \(f_0\). To demonstrate that, we have used an LF RFID reader (viz., Fig. 4.32) that operates on the 125 kHz low frequency band and have observed the signal spectrum using the NooElec smart SDR along with Ham it up module (viz., Fig. 4.32) and the SDR\# software. Fig. 4.33 (top left) shows the RFID-reader’s signal broadcasted at \(f_0=123.67\) kHz (main harmonic). Interestingly, we have also observed the other harmonics at \(2 \cdot f_0=248.65\) kHz (top right), \(3 \cdot f_0\), \(4 \cdot f_0\), \(5 \cdot f_0\), ..., and also at 1.998 MHz (bottom left) and 13.625 MHz (bottom right).

Considering the results in Fig. 4.33, using FHSS in RFID systems should be on higher-frequency bands. This implies that if we plan to use FHSS on car passive keyless entry and start systems, future keyfobs would be embedded with a UHF RFID-tag instead of an LF RFID-tag, which should not be an issue with respect to the cost. Nonetheless, moving to
4.5. RFID COMPROMISE

Figure 4.33: From top to bottom and from left to right: The main harmonic at 123.67 kHz, the 2\textsuperscript{nd} harmonic at 248.65 kHz, the 16\textsuperscript{th} harmonic at 1.998 MHz, and the 110\textsuperscript{th} harmonic at 13.625 MHz, of the 125 kHz RFID-reader signal (carrier).

the UHF-RFID band which has a wider bandwidth than LF band, will not entirely solve the problem of using FHSS in PKES. In fact, each country radio regulatory agency defines the bandwidth to be used in its country. In Europe, the total UHF-RFID bandwidth is 2 MHz divided into 10 channels of 200 kHz-wide each, whereas in the US, the total bandwidth is 26 MHz divided into 50 channels of 500 kHz-wide each. Also, it is important to note that existing RFID-readers that support frequency hopping, only apply FHSS in order to prevent interference with other RFID-readers. In fact, as soon as an RFID-tag is approached to the reader, the frequency is fixed and the communication occurs over the current frequency.

To illustrate that, we have used a UHF-RFID reader/writer that supports FHSS (viz., Fig. 4.32). We have configured the reader in such a way so that it hops over four channels after each second (ch\textsubscript{1}=913.80 MHz, ch\textsubscript{2}=914.20 MHz, ch\textsubscript{3}=914.60 MHz, and ch\textsubscript{4}=915.00
We have observed the carrier moving within the spectrum as shown in Fig. 4.34, where the image on the top left, top right, bottom left, and bottom right, shows the reader’s carrier intercepted on channel 913.80 MHz, 914.20 MHz, 914.60 MHz, and 915.00 MHz, respectively, while hopping from one channel to another. At a given time, we have approached the UHF RFID-tag (viz., patch-tag in Fig. 4.32) to the reader and have observed that the reader’s carrier has stopped on the current channel and the communication has taken place. Then, we have moved away the tag from the reader and have observed that the reader’s carrier has moved forward as if it has continued hopping during the tag-reading operation. It is obvious that hopping over four channels is not a perfect way of applying FHSS. A simple hardware with only 1.20 MHz of acquisition bandwidth can intercept all the signals over all four channels. Notwithstanding, we have used only four channels to make it simple for the illustration.

Based on the previous results, the RFID-system has to be implemented in such a way so that the data to be exchanged between the reader and the tag should be sent over different frequencies separately. Moreover, hopping over 50 UHF channels (902.60 MHz to 928.00 MHz) seems to be fine to skip eavesdroppers when the latter is using a narrow acquisition bandwidth receptor. In fact, we have configured the RFID-reader to hop over all 50 channels sequentially. Then using the NooElec smart SDR (with Ham it up module) and the SDR♯ software tool, we have tried to intercept the carrier. The NooElec smart SDR hardware has only 2.4 MHz of acquisition bandwidth. Thus, we have only managed to intercept the carrier on four channels. However, if the hopping sequence is randomly generated, the chance of intercepting the carrier will be considerably reduced. Indeed, if we denote the eavesdropper acquisition bandwidth by $w^e$ (where $w^e = |f_{\text{max}}^e - f_{\text{min}}^e|$) and the system’s bandwidth by $w^s$ (where $w^s = |f_{\text{max}}^s - f_{\text{min}}^s|$), the probability of the eavesdropper to successfully intercept a signal on a given channel is expressed as $P^e = w^e / w^s$, such that $\lim_{w^s \to \infty} P^e = 0$. From an attacker viewpoint, the success probability $P^e$ of the attacker could also converge to 1 when $|w^e - w^s| < \epsilon$. In fact, nowadays fabricating a UHF transceiver with 26 MHz of
acquisition bandwidth is not an impossible project. The current market actually provides public access to devices with such acquisition bandwidth. For instance, the Hack RF One\textsuperscript{33}, is a transceiver that has 20 MHz of acquisition bandwidth and is commonly used for hacking. Such a device can be used to capture the signals that are exchanged within the UHF RFID-band, which is defined by the FCC\textsuperscript{40}. Therefore, relying on the currently defined UHF RFID band to apply FHSS for PKES is not a prominent mitigation for relay attack. We need to augment the operational frequency to obtain a wider bandwidth and thus a larger hopping set. This would make the fabrication task of a wide acquisition bandwidth transceiver very hard and expensive for the attacker. Moreover, amplifying an entire wide band involves the amplification of both signals and noises. The attacker needs to perform additional signal

\textsuperscript{33}Hack RF One

\textsuperscript{40}FCC (Federal Communications Commission) is the radio frequency regulation agency in the United-States of America.
4.5. RFID COMPROMISE

processing to make the right signal reaches its destination in such a way so that it can be correctly interpreted by the car and/or the keyfob.

**Applying FHSS on PKESs**

Based on the observations and conclusions from the previous subsections, in this subsection we provide solutions that can be followed to implement future PKES systems which will be resilient against relay attacks through the use of FHSS:

- The operational frequency has to be high enough to provide the system with a wide hopping bandwidth. If the EHF/SHF (Extremely/Super High Frequencies) can be adopted, it will be a perfect frequency band to operate on. A wide bandwidth makes the task very hard for the attacker as the latter will have to amplify a wide band which includes other signals as well as many noises. Nonetheless, the selected frequency band has to be internationally accepted. Otherwise, interoperability issues may arise. For example, a PKES manufactured in the US will not be able to operate legally with respect to the frequency regulations in another country.

- The IEEE V-band (40 – 75 GHz) is a very suitable frequency band to be adopted in car PKES systems. In fact, within this band, certain frequencies, such as the 24 GHz and 60 GHz, have a higher atmospheric absorption, in particular, from humidity (H₂O) and oxygen in the air (O₂). This makes them not suitable for communications, but suitable for securing very short-range communications (≤ 1m) such as in car PKES systems. Attackers need extremely high power to amplify signals on that band and relay them to a longer distance. Also, the V-band provides a very wide bandwidth, which makes it a perfect band to apply FHSS. For example, if we simply consider a bandwidth of 2 GHz with \( f_0 = 60 \) GHz being the central frequency of the band and consider radio channels of 500 kHz width, we would have a total of 4000 channels to hop in, which is perfect for applying FHSS and changing the radio domain.
The hopping sequence has to be shared between the keyfob and the car. It can be provided as a per-car unique list of possible hopping frequencies (shared secret key). Each entry of the list is identified by an Id. The Id is exchanged between the car and its keyfob to agree about the hopping sequence to be used in a given session. Also, the hopping sequences have to be generated in such a way so that designing an RF-hardware with multiple receptors and antennas with wider acquisition bandwidth will be unpractical (very hard) and very expensive to predict the sequences.

The information that needs to be exchanged between the RFID-reader (car) and the RFID-tag (keyfob) has to be organized in such a way so that it can be transmitted over different channels. That is to say, we do not want the data rate to be very high so that the entire communication (e.g., the execution of the authentication protocol) happens on a single frequency. However, we do want the system to hop from one channel to another at a very high rate.

**PKES Design**

Let us consider a passive keyless entry and start system (PKES) implementing a distance-bounding protocol following the Brands and Chaum’s distance-bounding protocol model [BC93]. In such a protocol, the authentication occurs over three phases. The first or the initial phase, which is also known as the setup phase or the slow phase, allows the car (verifier) and its associated keyfob (prover) to share the nonces and agree on the security parameters. The second phase, called the critical or the timed phase, consists of multiple rounds in which the car sends a challenge to the keyfob which replies back with a corresponding response. Thus for \( k \) rounds, there would be \( k \) exchanged challenge and response tuples. During this phase, the car measures the RTTs (Round Trip Times) induced by each round execution. Finally, the last phase, which is also known as the authentication phase, allows the car (verifier) to conclude on a decision of whether to unlock the car or not by checking the computed
4.5. RFID COMPROMISE

\[Sk, \delta t, \text{ and } S = [S(0), \ldots, S(n)]\]

\[Sk \text{ and } S = [S(0), \ldots, S(n)]\]

\[\text{Generate } n_P \rightarrow V\]

\[\text{Generate } n_V \rightarrow P\]

\[\text{Select } S(id) \rightarrow P\]

\[\text{Select } S(id) \rightarrow V\]

\[\text{Tune RF on } f_{j \in S(id)}\]

\[\text{Tune RF on } f_{j \in S(id)}\]

\[\text{Generate } c_j \rightarrow V\]

\[\text{Generate } c_j \rightarrow P\]

\[\text{Compute } r_j \leftarrow m_0[j]\]

\[\text{Compute } r_j \leftarrow m_1[j]\]

\[\text{Hop to } f_j \in S(id) \mid f_j \neq f_{j-1}\]

\[\text{Hop to } f_j \in S(id) \mid f_j \neq f_{j-1}\]

\[\text{Verify all } r_j \text{ and all } \Delta t_j\]

\[\text{Execute } cmd\]

---

**Figure 4.35:** A distance-bounding protocol adopting FHSS for relay attack mitigation. During the fast phase, the car (verifier) and the keyfob (prover) hop from one frequency \(f_j\) to another at each round \(j\). The number of rounds is a security parameter mutually fixed during the first phase.

RTTs as well as the responses that the keyfob provided during the second phase. In the positive scenario, the car unlocks the doors. Also, for the same execution of the protocol, it will eventually start the car engine after the driver pushes on the “Start Engine” button.
Therefore, to make this system resilient against relay attacks, we augment the protocol with FHSS. We can consider changing the operational frequency at each phase as well as at each round during the critical phase (i.e., second phase). The resulting protocol would look like the one illustrated in Fig. 4.35. In this protocol, and during the first phase, the car and the keyfob exchange nonces and agree on an \( id \in [0 \ldots n] \) to select a sequence of frequencies \( S(id) \) among a shared list to be used during the second phase. During the fast phase, at each round \( j \in [0 \ldots n'] \), a frequency \( f_j \in S(id) \) is fixed for the entire round. The hoping rate is in this case assumed to be equal to the time of one round. In the extreme case, if an eavesdropper misses to intercept (relay) one single round, the whole protocol execution fails. Otherwise, in the general scenario, if we tolerate the success of \( \alpha \) rounds out of \( n' + 1 \) rounds to unlock the car, the attacker needs to relay at least \( k \) rounds (\( k \geq \alpha \)) to bypass the mechanism. This can be made hard by increasing the value of the parameter \( \alpha \) and using a wider hopping bandwidth \( w^s \) as we have discussed earlier.

### 4.5.5 Discussion

As a first step toward relay attack mitigation, we have proposed the use of FHSS (Frequency Hopping Spread Spectrum) as a physical-layer countermeasure. The intuition behind using this spectrum spreading technique is to hide the frequency on which a car and its keyfob communicate. Therefore, if the car and the keyfob secretly change their communication frequency, the attacker will have a hard time synchronizing the communication to conduct a successful relay attack.

To correctly apply FHSS on PKES systems, we have first studied the application of FHSS on RFID systems, in general, and then on PKES systems, in particular, and have proposed recommendations for future PKES implementations. We claim that by following the provided guidelines, future PKES systems would be more resilient not only to relay attacks but also to jamming attacks. Designing hardware to bypass the proposed solution would be very hard and expensive for attackers, if not impossible. Finally, we plan to
implement the proposed solution and experimentally evaluate it with respect to security and performance in a future work.

4.6 Conclusion

It has gone noticeable that exploiting vulnerabilities in authentication protocols to generate attacks on availability has become a common attack methodology for cybercriminals. Such attacks usually require a small cost, ordinary skills, and can cause serious damage and loses. In this chapter, we have devoted the first three sections to exploit vulnerabilities in authentication protocols of Wi-Fi, classical Bluetooth, and BLE, to generate DoS attacks. In fact, in Section 4.2, we have exploited a race-condition vulnerability as well as an implementation flaw in WPA2-PSK and WPA3-SAE authentication protocols and generated various DoS attacks on Wi-Fi. We have also discussed possible countermeasures to mitigate the presented attacks. In Section 4.3 and 4.4, we have focused Bluetooth technology. We have dedicated Section 4.3 to classical Bluetooth and Section 4.4 to Bluetooth low energy (BLE). In each Bluetooth version we have presented a vulnerability on the authentication protocol. We have exploited the vulnerabilities and demonstrated the feasibility of multiple attacks on different types of Bluetooth devices. Also, we have proposed countermeasures to fix the vulnerabilities.

Furthermore, in the last section, we have focused on RFID technology, in particular, on relay attacks on PKESs (Passive Keyless Entry and Start Systems). Relay attacks for auto-theft exploit vulnerabilities in PKES authentication protocol to breach the availability of RFID by stealing cars. In fact, in the context of IoT, a car can be perceived as more complex Thing, possibly a distributed system with four wheels [Sch18]. Thus, we have studied the possibility of applying FHSS (Frequency Hopping Spread Spectrum) as a fundamental physical-layer security measure to counter relay attacks.
Part II

Thing-to-Thing Authentication for IoT
Chapter 5

PUF-based Authentication Protocols

In this chapter, we review the security of recent PUF-based authentication protocols. We analyze their security and present possible attacks on the protocols. Also, we draw security lessons to be considered by future authentication protocol designers.

5.1 Introduction

The service of authentication constitutes the spine of all security properties. It is the phase where entities prove their identities to each other and generally establish and derive cryptographic keys to provide security services. Due to the heterogeneity and the particular security requirements of IoT (Internet of Things), developing secure, low-cost, and lightweight authentication protocols has become a serious challenge.

Interestingly, during the past five years, there has been a remarkable attraction and convergence from the research community and the industry to adopt PUFs (Physical Unclonable Functions) as a prominent physical security technology. Important industrial cores have already implemented the technology to develop secure integrated circuits [NXP18; Mic16; Int15; Cis16]. In the meantime, researchers have turned their attention to PUF technology to develop lightweight and security protocols for IoT.

PUFs are physical one-way functions constructed from the unique nanoscopic-structure of physical objects (e.g., integrated circuits, crystals, magnets, lens, solar cells, or papers).
and their reaction to random events. This intrinsic uniqueness in the structure and reaction is due to the idiosyncrasies in the manufacturing process of the objects. It allows not only the unique identification of an object but also its authentication. Also, it is assumed to be impossible to clone the PUF of an object (and hence the object), which can be perceived as a security-by-design that will prevent any possible impersonation and cloning attacks. Thus, PUFs are considered as a novel, reliable, and prominent physical security technology to develop secure integrated circuits and lightweight authentication protocols for IoT.

Furthermore, the rapid convergence of adopting PUFs for authentication by the research community has resulted in many subsequent research work on lightweight, low-cost, and secure-by-design authentication protocols [BMP+20; BNS+20; QM20; YMK+19; NY19b; CGS+19; LSL+19; KYK+19; MSM+18; MLM+18; BBM+19; BBM18; FYZ+18; YGH18; IB17; CZ16]. This has turned our attention to devote this chapter to investigating the security of the most recent PUF-based authentication protocols.

Therefore, we devote this chapter to analyze the security of recent PUF-based authentication protocols. We first provide the necessary background on PUFs, their types, and related attacks, as well as discuss how PUFs are used for authentication. Then, we analyze the security of existing PUF-based authentication protocols to identify possible attacks. Based on the identified attacks, we discuss possible security lessons to be considered by future authentication protocol designers.

5.2 PUFs (Physical Unclonable Functions)

5.2.1 PUFs Overview

PUFs (Physical Unclonable Functions), also known as POWFs (Physical One-Way Functions) [Rav01] or PRFs (Physical Random Functions) [GCvD+02], are physical one-way functions constructed from the unique nanoscopic-structure of physical objects (e.g., integrated circuits, crystals, magnets, lens, solar cells, or papers) and their reaction to random
events. In fact, when a PUF is excited with a random event, called stimulus, the function returns a unique, unpredictable, and reproducible output. This unique output represents the actual fingerprint of a particular PUF and allows its distinction among others. In the field of information technology security (ITS), the stimulus is known as the challenge, whereas the output is known as the response. The set of all possible challenges and their corresponding responses is often referred to as the CRPs (Challenge-Response Pairs).

In the electronics and nanotechnology disciplines, PUFs\(^1\) are constructed from the idiosyncrasies in the manufacturing process of semiconductors [BC96; BDM02; Nas01] (e.g., inherent delay characteristics of wires and transistors [MP15]) to assign uniquely distinct fingerprints to semiconductor pieces [HBG\(^+\)17]. Thus, physical cloning of a piece of semiconductor becomes very difficult (assumed to be impossible by many authors). This allows distinguishing a genuine integrated circuit from a counterfeit one (including exact copies produced during the same manufacturing process [HBG\(^+\)17]). Nowadays, PUFs are considered as a low-cost and a more sophisticated integrated-circuits-tagging technology than hard-printed serial numbers, bar codes, and holograms, which are readily reproducible.

Formally, we can define a physical unclonable function as a function \(\Psi: C \rightarrow R\), where \(C\) is the set of challenges and \(R\) is the set of the corresponding responses. Also, the function \(\Psi(\cdot)\) should have the following properties:

— **Reliable.** At any time, the same challenge should return the same response, i.e., \(\forall c_i \in C, \Psi_t(c_i) \cong \Psi_{t'}(c_i)\), where \(t' > t\) (at different instants of time, \(t\) and \(t'\), such that \(t'\) is greater than \(t\), the response is the same for a given challenge \(c_i \in C\)).

— **Unique.** The function \(\Psi(\cdot)\) cannot have an equivalent function, i.e., \(\forall c_i \in C, \Psi(c_i) = \Psi'(c_i)\), then \(\Psi(\cdot) = \Psi'(\cdot)\).

— **Onewayness.** The function \(\Psi(\cdot)\) is a one-way function and it is not invertible, i.e., \(\Psi \circ \Psi^{-1} \neq id_C\), where \(id_C\) is the identity function on the set of challenges \(C\).

\(^1\)When a PUF is constructed from a semiconductor, which is generally made of Silicon material, the PUF is categorized as a Silicon-PUF (SPUF).
— **Easy to evaluate.** For any challenge, the function $\Psi(\cdot)$ is easy to evaluate, i.e., $\forall c_i \in C, O(\Psi(c_i)) \leq O(1)$ (the temporal complexity for evaluating a PUF is constant).

— **Difficult to replicate.** The function $\Psi(\cdot)$ cannot be copied or cloned, i.e., if $\forall c_i \in C, \Psi(\cdot) = \Psi'(\cdot)$, then $O(\Psi'(\cdot)) \approx \infty^2$ (the temporal complexity of building a clone tends to be infinite).

— **Difficult to predict.** The responses of $\Psi(\cdot)$ are very difficult to guess or predict, i.e., $\forall \Gamma_{k\geq0} = \{(c_0, \Psi(c_0)), \ldots, (c_k, \Psi(c_k))\}$, $O(\Phi_{\Gamma_{k\geq0}}(c_{n>k}, \Psi(c_{n>k}))) \approx \infty$, where $\Phi(\cdot)$ is a prediction function based on the set $\Gamma_{k\geq0}$ (the temporal complexity of predicting a response, knowing a set of CRPs, tends to be infinite).

When the cardinality of the set of challenge and response pairs (CRPs) is large, the function $\Psi(\cdot)$ is called a strong PUF. Otherwise, the PUF is called a weak PUF\(^3\) [RSS\(^+\)10]. Also, by increasing the size of the CRPs set, the size of the PUF circuit increases as well. Hence, there exists a proportional relationship between the size of the circuit implementing the PUF and the cardinality of its CRPs set (strength of the PUF). In general, if the number of CRPs of a given PUF increases exponentially with the size of the PUF circuit, the function is considered strong, whereas if it scales in a linear or polynomial manner, the function is then considered weak [MBW\(^+\)19]. In general, the output of a PUF is subject to noise, which may slightly change the output value for a given challenge. Thus, to generate the correct output for a given input in the presence of noise, a fuzzy extractor is used [DOR\(^+\)04; JW99]. When a PUF is supported by a fuzzy extractor to eliminate the noise and correct the error, the PUF is called controlled-PUF [RSS\(^+\)10].

For the past twenty years, PUFs have attracted the attention of a large number of researchers. They are recognized as reliable and promising security tools for implementing future security solutions. The attraction is due to the following assets: (1) PUFs have a relatively lower hardware overhead compared to other hardware solutions. (2) PUFs

\(^2\)Here $\infty$ means a very large value that can be considered as impractical to realize.

\(^3\)Strong PUFs, such as arbiter-PUF, are generally used for device authentication, whereas weak PUFs are mainly used in key generation.
5.2. PUFS (PHYSICAL UNCLONABLE FUNCTIONS)

provide higher physical security as they can be used to extract secret keys (volatile-secret keys) from a physical system instead of generating them using software and storing the keys in non-volatile memories (e.g., ROM, fuses, or EEPROM) that need to be secured as well (which is difficult and expensive). This makes them resilient to known physical invasive attacks. (3) They can be used as a hardware random number generators to generate purely random numbers. (4) PUFS do not require special manufacturing processes and treatments (programming and testing steps), which makes them low-cost security solutions. (5) PUFS are practically unclonable. They can be used to design and manufacture low-cost tamper-resistant circuits and devices. (6) PUFS can be used for key agreement between two resource-constrained devices that have never met with each other (no prior key sharing). (7) PUFS are multidisciplinary technologies, where researchers from different fields are contributing to their development.

5.2.2 Types of PUFS

Since the emergence of PUFS, researchers have worked on designing PUFS by exploiting the physical characteristics of different materials, including Silicon, crystals, magnets, lens, solar cells, and papers. This has resulted in a large variety of PUFS, including but not limited to, delay-based PUFS (e.g., Arbiter-PUFS [GCvD+02], ring oscillator-PUFS [SD07], and glitch-PUFS [And10a]), memory-based PUFS (e.g., SRAM-PUFS [GKS+07a], DRAM-PUFS [TKX+15], SR Latch PUF [LHK+10], and Rowhammer-PUFs [SXA+17])\(^4\), acoustical-PUFs [Vri04], coating-PUFs [TSS+06], optical-PUFs (e.g., paper-PUFs [Nat93], Compact Disc-PUF [HDS09], and liquid crystal-PUF [LOR+17])\(^5\), and magnetic-PUFs [IM94]. These PUFS differ from each other in their environmental application, source of randomness (e.g.,

\(^4\)In some literature, delay-based and memory-based PUFS are often classified as electronic CMOS (Complementary Metal Oxide Semiconductor)-PUFs.

\(^5\)In some literature, coating-based and optical-based PUFS are often classified as MEMS (Micro-Electro-Mechanical Systems)-based PUFS.
semiconductors, lens, crystals, magnets, or solar cells), excitation mechanisms (e.g., electronic signals, beam of light or laser, or electromagnetic waves), or other parameters. Interested readers are referred to the work of McGrath et al., [MBW+19], where a comprehensive taxonomy of existing PUFs was extensively elaborated.

Despite of the existence of various types of PUFs, the Silicon-PUFs (a.k.a., delay-based PUFs) are the most commonly researched and adopted in the electronics and nanotechnology disciplines. This type of PUFs rely on the timing and delay information retrieved from semiconductors (i.e., electronic components made of Silicon material). Moreover, similar to delay-based PUFs, memory-based PUFs are becoming more and more popular in the research field as well as in the industrial field. Nowadays, most of the latest secure ICs adopt SRAM-based PUFs for secure storage. For example, the NXP company (the semiconductors division of Philips) has embedded an SRAM-based PUF on its SmartMX2 P60 microcontroller and LPC54S0xx family of microcontrollers [NXP18]. Also, Microsemi company has adopted SRAM-PUFs on their SmartFusion2 System-on-Chip FPGA devices [Mic16]. Other hardware from various companies have adopted SRAM-PUFs to implement physical security, such as Coherent Logix HyperX, Intel Altera Stratix 10 FPGA [Int15], Redpine Signals WyzBee, and the Xilinx Zynq Ultrascale+ [Cis16]. This attraction to SRAM-PUFs is due to the fact that this type of PUF is considered more resilient to temperature variations and more compact than many other memory-based PUFs.

It is important to note that despite the type of the PUF that is being adopted, the reliability\(^6\) of PUFs, in general, is affected by environmental fluctuations, which include temperature variations, voltage variation, and ambient noise [PDS+16; JBK+15; CBD+19; LHS18; WYD+17]. In the case of delay-based PUFs (Silicon-PUFs), the delay of wires and transistors strongly dependent on these environmental fluctuations. This, for instance, could result in generating two different responses for a given challenge at two different

---

\(^6\)The reliability of a PUF is a performance metric that measures its reproducibility. A reliable PUF produces the same response for a given challenge at different times and under different environmental conditions [MS09].
5.2. PUFS (PHYSICAL UNCLONABLE FUNCTIONS)

instant of times or under two different environmental conditions. Therefore, to improve the reliability of PUFs (as well as other properties, e.g., uniqueness), many researchers have proposed different solutions to guarantee a certain level of reliability. For example, fuzzy-extractors, error correcting codes, and assisting computed helper data are usually applied to generate correct PUF responses.

5.2.3 PUFs for Authentication

One of the recently explored security applications of PUFs is the development of on-the-fly and low-cost authentication protocols. These protocols would provide security for resource-constrained devices without having the devices to store any credentials on their limited non-volatile memories. This makes them resilient to invasive attacks [AK96; AK98].

![Figure 5.1: A typical PUF-based authentication protocol using challenge-response mechanism, where Phase 1 is the registration phase and Phase 2 is the authentication phase.](image-url)
To exploit the advantages of PUFs for integrated circuit authentication in general, and device authentication in particular, a typical challenge response-based protocol is adopted [GCvD+02]. The protocol consists of two main phases, the registration phase (a.k.a., enrolment phase) and the verification phase (a.k.a., authentication phase). During the registration phase, a device is enrolled in a database (a trust center) by registering pairs of challenges and responses (CRPs), generated from the PUF that is embedded in the device’s circuit. Therefore, each registered device $d_{id}$ will have its proper set of challenge and response pairs in the database $\Gamma_{id} = \{(c_0, r_0), \ldots, (c_n, r_n)\}$.

The authentication of a registered device $d_{id}$ consists of randomly selecting a pair of registered challenge and response $(c'_i, r'_i) \in \Gamma_{id}$ and interrogating the device $d_{id}$ by sending the challenge value $c'_i$ and obtain the corresponding response $r''_i$ from the device. Once the response $r'_i$ is received, it is compared with the registered response $r_i$ in the database. If both responses are identical (or closely identical), i.e., $r'_i \simeq r''_i$, the device $d_{id}$ is authenticated, otherwise, it is rejected. In general, the device is referred to as the prover $v$, whereas the authenticator party (which could be the trust center or another device), is referred to as the verifier $v$. This process is illustrated in the MSC\(^7\) of Fig. 5.1, where $\Psi_p(\cdot)$ denotes the prover’s PUF. We note that for higher security, the challenge-response pass during the authentication phase can be run multiple times (i.e., rounds). Also, after an authentication round, the used CRP $(c_i, r_i)$ is sometimes deleted from the verifier’s database to prevent replay attacks [RMA15].

Finally, besides low-cost authentication, PUFs are used in many other applications such as, for the protection of intellectual property (IP) of hardware components (e.g., smart cards [EFK+12] and FPGA circuits [LLZ+15; KGM+08; GKS+07b; SS06]), software components (e.g., software IP binding [GMS09] and images [ZCC20]), and embedded devices’ firmware integrity [MUS+18]. They are also employed for generating cryptographic keys.

\(^{7}\)MSC (Message Sequence Chart) is a graphical language for the description of the interaction between different components of a system. This language is standardized by the ITU (International Telecommunication Union).
and random numbers [ZQ20; SFI+14; GIK15; MHV12; PD11], and securing memories and processors.

5.2.4 Attacks on PUFs

In this subsection, we present some attacks on PUFs.

**PUF Invalidation Attack.** This attack assumes that the attacker has physical access to a device’s PUF. The attacker randomly tampers with the PUF circuit in such a way so that the behavior (i.e., output) of the PUF circuit changes and behaves as another PUF. For example, the attacker can spray the PUF circuit with some chemical substances so that the conductivity of the circuit gets affected. Also, the attacker can place some electromagnetic devices near the PUF circuit so that the PUF’s output gets altered. Therefore, any future authentication attempt with the device’s PUF will fail as the PUF has changed. To mitigate this attack, we need to physically secure the PUF circuit so that it does not get easily affected by any tampering action or environmental variation.

**PUF Spoofing Attack.** In this attack, a malicious insider has access to all the CRPs of another device. This could happen when the authenticator, generally, the trust center server (a.k.a., verifier), is compromised. In this case, the attacker can spoof the legitimate device and perform successful authentication with other devices that authenticate the spoofed device using the same CRPs. This could also occur in the T2T architecture where devices store CRPs of each other for authentication.

**Eavesdropping Attack.** An attacker intercepts authentication transactions that contain the challenge values and their response values that are computed by the PUF of a target device. The attacker then spoofs the device (prover) and responds correctly to the verifier. The success of this attack relies on the following two assumptions: (1) The number of possible challenge-response pairs is too small. (2) The same challenge is used twice. Such assumptions exist on weak PUFs.
Table 5.1: This table summaries the differences between the reviewed PUF-based authentication protocols for T2M (cf., Section 5.3.1) and T2T (cf., Section 5.3.2) authentication schemes. We have entitled some columns with a numbered letter $P_i$, where P1: Mutual authentication using PUFs, P2: Lightweight (no asymmetric cryptography and no preshared keys), P3: Scalability, P4: Smart, P5: Key establishment (for message authenticity, confidentiality, and integrity), P6: Security evaluation, P7: Performance evaluation, P8: PUF-circuit size evaluation. The symbol $✓$, $✗$, and $●$, indicate Yes, No, and Possible, respectively.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Protocol Used PUF</th>
<th>Protocol properties</th>
<th>Evaluation</th>
<th>Implementation platform</th>
<th>Possible attacks on the protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>T2M</td>
<td></td>
<td></td>
<td>P1</td>
<td>P2</td>
<td>P3</td>
</tr>
<tr>
<td>Boyapally et al., [BMP*20]</td>
<td>5-4 DAPUF</td>
<td>$✓$ $✓$ $✓$ $✓$ $✓$ $✓$</td>
<td></td>
<td>Digilent Nexys-4</td>
<td>Spoofing attacks</td>
</tr>
<tr>
<td>Qureshi et al., [QM20]</td>
<td>Modified APUF</td>
<td>$✓$ $✓$ $✓$ $✓$ $✓$ $✓$</td>
<td></td>
<td>ZYNQ-7000 SoC</td>
<td>Spoofing attacks</td>
</tr>
<tr>
<td>Yanamala et al., [YMK*19]</td>
<td>Unspecified</td>
<td>$✗$ $✗$ $✗$ $✗$ $✗$ $✗$</td>
<td></td>
<td>Xilinx ZC-700</td>
<td>Spoofing attacks</td>
</tr>
<tr>
<td>Nozaki et al., [NY19b]</td>
<td>XOR-APUF</td>
<td>$✓$ $✗$ $●$ $✓$ $✓$ $✓$</td>
<td></td>
<td>Xilinx Virtex-5 FPGA</td>
<td>Spoofing attacks</td>
</tr>
<tr>
<td>Chatterjee et al., [CGS*19]</td>
<td>5-4 DAPUF</td>
<td>$✓$ $✓$ $✓$ $✓$ $✓$ $✓$</td>
<td></td>
<td>Digilent Nexys-4 FPGA</td>
<td>Spoofing attacks</td>
</tr>
<tr>
<td>Liang et al., [LSL*19]</td>
<td>TSMCA-PUF</td>
<td>$✓$ $✓$ $✓$ $✓$ $✓$ $✓$</td>
<td></td>
<td>Xilinx Artix-7 FPGA</td>
<td>Spoofing attacks</td>
</tr>
<tr>
<td>Kim et al., [KYK*18]</td>
<td>Unspecified</td>
<td>$✓$ $✓$ $✓$ $✓$ $✓$ $✓$</td>
<td></td>
<td>XUPV5-LX110T</td>
<td>Desynchronization</td>
</tr>
<tr>
<td>Mahgul et al., [MLM*18]</td>
<td>Unspecified</td>
<td>$✗$ $✗$ $✗$ $✗$ $✗$</td>
<td></td>
<td>Unspecified</td>
<td>CRPs disclosure</td>
</tr>
<tr>
<td>Yilmaz et al., [YGH18]</td>
<td>TSMCA-PUF</td>
<td>$✓$ $✓$ $✓$ $✓$ $✓$ $✓$</td>
<td></td>
<td>Zolertia RE-Mote</td>
<td>Spoofing attacks</td>
</tr>
<tr>
<td>Idriss et al., [IB17]</td>
<td>Unspecified</td>
<td>$✓$ $✓$ $✓$ $✓$ $✓$</td>
<td></td>
<td>Simulation</td>
<td>Spoofing attacks</td>
</tr>
<tr>
<td>Bansal et al., [BNS*20]</td>
<td>Unspecified</td>
<td>$✓$ $✓$ $✓$ $✓$ $✓$</td>
<td></td>
<td>Simulation</td>
<td>Message forging</td>
</tr>
<tr>
<td>Barbaroschi et al., [BBM18]</td>
<td>SRAM-PUF</td>
<td>$✓$ $✓$ $✓$ $✓$ $✓$ $✓$</td>
<td></td>
<td>ST31F3F7-3MCU</td>
<td>Desynchronization</td>
</tr>
<tr>
<td>Mahgul et al., [MSM*18]</td>
<td>Unspecified</td>
<td>$✓$ $✓$ $✓$ $✓$ $✓$</td>
<td></td>
<td>Cubietruck-board</td>
<td>—</td>
</tr>
<tr>
<td>Feng et al., [FYZ*18]</td>
<td>SRAM-PUF</td>
<td>$✓$ $✓$ $✓$ $✓$ $✓$</td>
<td></td>
<td>MSP430G2 &amp; MSP430G2553</td>
<td>Spoofing attacks</td>
</tr>
<tr>
<td>Clupek et al., [CZ16]</td>
<td>Unspecified</td>
<td>$✗$ $✗$ $✗$</td>
<td></td>
<td>—</td>
<td>Secret disclosure</td>
</tr>
<tr>
<td>Feng et al., [FYZ*18]</td>
<td>SRAM-PUF</td>
<td>$✓$ $✓$ $✓$ $✓$ $✓$</td>
<td></td>
<td>MSP430G2553</td>
<td>Message forging</td>
</tr>
<tr>
<td>Clupek et al., [CZ16]</td>
<td>Unspecified</td>
<td>$✓$ $✓$ $✓$</td>
<td></td>
<td>—</td>
<td>Modeling attacks</td>
</tr>
</tbody>
</table>
5.2. **PUFS (PHYSICAL UNCLONABLE FUNCTIONS)**

**Machine-Learning Attack.** This attack is also known as PUF reverse-engineering or PUF modeling attack. In such an attack, an attacker collects a large number of CRPs used during an authentication or by interrogating a device’s PUF with many challenges to obtain their responses. The collected CRPs are then used to apply machine learning algorithms in order to produce a software model of the PUF that is capable of correctly predicting the response of new challenges. The efficacy of this type of attack on a given PUF depends on the number of CRPs to be collected from the PUF and the time it requires to build the model given the collected CRPs. This type of attack has been demonstrated in a large number of subsequent publications [ZWW+18; Del19; RSS+13; TG15; GSS15; RS14; YN17; AZ17; MRM+13].

**Side-Channel Attack.** This type of attacks on PUFs have been demonstrated in [MRM+13; WWN+14; RXS+14; DV13; KB15; NY19a; MSS+11a; MSS+11b]. The goal of these attacks is mostly to gain additional information about the PUF and use that information to improve machine-learning computation time (i.e., training times) from exponential to polynomial. For example, by applying power-based side-channel techniques, an attacker would aim to determine the amount of current drawn from the supply voltage during an output computation (e.g., transition from zero to one of a latch [RXS+14]) and hence the power consumption of the function. Also, by applying timing-based side-channel techniques, an attacker would aim to learn additional information about the delay of individual response bits. Yet, this type of attacks require complete physical access to the PUF, which is not always possible.

**Replay Attack.** This attack is possible if the authentication protocol allows the use of the same challenge twice and the challenges, as well as their responses, are sent in plaintext. An attacker can then intercept some challenge values as well as their corresponding response values to replay them later on. This would allow the attacker to impersonate the device whose challenge and response values were intercepted.
5.3. RECENT PUF-BASED AUTHENTICATION PROTOCOLS

**CRP Disclosure Attack.** As many PUF-based authentication protocols are designed by extending the typical authentication protocol illustrated in Fig. 5.1, these protocols are subject to CRP disclosure. In fact, as the CRPs are stored in plaintext at the verifier, if the latter is compromised, the stored CRPs of the concerned (registered) devices will be revealed. This would allow attackers to impersonate the PUFs of those devices.

In addition to the above mentioned attacks, other attacks on PUFs have been reported in the literature on specific physical unclonable functions. For example, Helfmeier et al., [HNB+13] demonstrated a PUF-cloning attack. They showed how they had managed to successfully reproduce the responses of a 2-kB SRAM-based PUF by gaining access to the initial states of the SRAM cells after memory-startup. They applied the PEA (Photonic Emission Analysis) technique [SNK+12] to dynamically extract the full content of the SRAM and create a clone of the SRAM. However, this attack requires physical access to the PUF circuit, extremely expensive equipments, as well as very long time, which makes it not affordable for most attackers. Nedospasov et al., [NHS+13] demonstrated a semi-invasive attack on SRAM-PUFs. They applied a laser probing technique to read the logic states of an SRAM cells.

5.3 Recent PUF-based Authentication Protocols

In this subsection, we analyze the security of some recent PUF-based authentication protocols. We identify security flaws and report possible attack scenarios on the reviewed protocols. This would allow future authentication protocol designers to learn from the identified flaws; lessons to consider when designing new PUF-based authentication protocols. Furthermore, we have classified these protocols into two categories based on the communication scheme for which each protocol was designed for. The first category (discussed in Subsection 5.3.1) gathers protocols that were developed for the T2M communication scheme, where a resource-constrained device (e.g., a sensor) and a resourceful device
5.3. RECENT PUF-BASED AUTHENTICATION PROTOCOLS

(e.g., desktop) securely authenticate each other. The second category (discussed in Subsection 5.3.2), groups protocols that were designed for the T2T communication scheme, where two resource-constrained devices authenticate each other. This classification is based on the corresponding authors’ claim when implementing the protocols.

We note that, throughout this subsection, when analyzing a particular protocol, we use the notation adopted by the original author of the protocol. This would help the reader referring to the original work. Also, in Table 5.1, we summarize the reviewed PUF-based authentication protocols, the type of the used PUF, authentication scheme, authentication protocol properties, implementation platform, protocol evaluation, and possible attacks. The property “Smart” (P4) refers to the resilience of the protocol against race-condition attacks discussed in the previous chapter.

Figure 5.2: Load modification-resistant smart meter authentication protocol by Boyapally et al., [BMP+20]. This protocol allows a PUF-enabled smartmeter to authenticate to a server in smart grid applications.
5.3. RECENT PUF-BASED AUTHENTICATION PROTOCOLS

5.3.1 T2M PUF-based Authentication Protocols

Load Modification-Resistant Smart Meter Authentication

Boyapally et al., [BMP+20] proposed a PUF-based authentication protocol for smart meters in the context of smart grid applications. The protocol allows a smart meter to be authenticated to a server using PUFs. The protocol adopts the DAPUF (Double Arbiter-PUF) [MYI+14] augmented with a linear feedback shift register (LSFR) module to take a 64-bit challenge and outputs a 128-bit response (which makes it harder to guess). The authors showed that it is resilient to man-in-the-middle attacks as well as to PUF-modeling attacks. They implemented the protocol over the Digilent Nexys Artix-7 FPGA-board in a smart meter. The protocol operates symmetric encryption (using AES) along with some asymmetric computations, such as bilinear pairings, to encrypt the response of the smart meters and store it securely on the server. Also, the responses of the smart meter are never exchanged in plaintext, which would prevent modeling attacks. Although the protocol seems to perform authentication of a smart meter to a server and establish a shared key using Diffie-Hellman key establishment protocol, we have found that the authentication can be easily breached and the server can be spoofed. We present the attack in the next paragraph.

Attack Scenario. During the authentication of the smart meter to the server, the smart meter initiates the authentication by sending an authentication request. Based on the source of the request, the server retrieves from the database a cryptographic information, denoted by $\sigma = (C, \sigma_1, \sigma_2)$. This information contains the challenge $C$ as well as two values, $\sigma_1$ and $\sigma_2$. The value of $\sigma_1$ was generated during the enrolment phase as $\sigma_1 = E_{sk_A}(\alpha_2)$, where $sk_A = H(\Psi(C))$ is the secret key of the smart meter, $\alpha_2 = H(sk_{id} | C | A)$, $A$ the identity of the smart meter, $sk_{id}$ is the secret key of the server, $\Psi(\cdot)$ is the smart meter PUF function, and $H(\cdot)$ is a hash function.

The authentication is illustrated in the MSC of Fig. 5.2. To generate the attack, upon
5.3. RECENT PUF-BASED AUTHENTICATION PROTOCOLS

\[ \sigma = (C, \sigma_1, \sigma_2) \]

\[ \Psi(\cdot) \]

\[ \text{Authentication Request} \]

Generate nonce \( n \) and prepare \( \sigma = (C, \sigma_1, \sigma_2) \).

\[ \beta_1 = H(\alpha_1 | \alpha'_2 | n - 1), \quad \beta_2 = H(\alpha_1 | \alpha'_2 | n) \]

\[ \beta_2 \]

Save \( \beta_2 \) in \( w \) then drop authentication.

Verify that \( \beta_1 = w \)

Server authenticated.

Figure 5.3: A spoofing attack scenario on Boyapally et al., authentication protocol [BMP+20]. The attack allows an attacker to impersonate a server and perform a successful authentication.

receiving an authentication request from the meter, the attacker impersonates the server and sends the value \( \sigma \) and \( n - 1 \), where \( \sigma \) is the fixed cryptographic information intercepted from a previous authentication and \( n - 1 \) a nonce. The smart meter performs the computations shown in the MSC of Figure 10 and sends the value \( \beta_2 = H(\alpha_1 | \alpha_2' | (n - 1) + 1) \) to the attacker. The attacker receives the value of \( \beta_2 \), saves it in a variable \( w \) (i.e., \( w = \beta_2 \)), then drops the authentication. Upon a second authentication attempt from the smart meter, the attacker sends to the smart meter the value \( \sigma \) and \( n \). The smart meter performs the authentication computations and replies by sending the value \( \beta_2 = H(\alpha_1 | \alpha'_2 | n + 1) \). The
attacker sends the value of \( w \) to the smart meter. The smart meter checks the value of \( w \) with the value of \( \beta_1 = H(\alpha_1 | a'_2 | n) \) and finds that they are equal, which authenticates the attacker as the legitimate server. This attack is illustrated in Fig. 5.3.

**Lessons**

1. The nonces that are used by both parties are dependent (e.g., Server uses \( n \) and Meter uses \( n + 1 \)). Indeed, there is no randomness introduced by the Meter (i.e., no actual nonce). This consequently results in the non-uniqueness of authentication sessions, and hence the possibility of generating replay attacks.

2. The authentication proofs (i.e., \( \beta_2 \) and \( \gamma_1 \)) used by both authenticating parties are strongly related. This would allow an attacker to easily find a way to construct forged authentication proofs with minimum effort. For instance, in this protocol, an attacker constructs authentication proofs by interrogating the verifier (i.e., the Meter).

**Identity Preserving Protocol for IoT Authentication**

Qureshi et al., [QM20] proposed a PUF-based identity-preserving authentication protocol, called PUF-IPA, for T2M authentication scheme. Similar to the concept used in [YMK+19], the protocol uses shuffling techniques to store obfuscated and uncorrelated information about the registered devices’ PUFs in the server instead of storing the challenge and response pairs (CRPs) of registered devices in plaintext. The server can authenticate the registered devices without having the original CRPs stored on its database. The authors showed that the authentication protocol is resilient to machine-learning attacks.

Unfortunately, the protocol is vulnerable to replay attacks. In fact, no verification at the server side is performed to check whether the random number (denoted as \( n_x \)), sent from the device to the server, was previously used or not. Thus, if an attacker intercepts the exchanged values (denoted as \( n_x \) and \( s'_{x+1} \)), it will be able to impersonate the device and respond using the intercepted values. Also, the devices are subject to locking attacks (denial of service attacks). In fact, the protocol implements a countermeasure to prevent attackers
from conducting a brute force attack. The countermeasure is based on FSM (Finite State Machine) and was first proposed in [GMA+17]. It works by locking the device’s PUF circuit after a certain number of authentication failures. Hence, an attacker can paralyze the availability of the system by brute-forcing the registered devices and locking them all.

By analyzing the protocol, we observe that the security of the server relies on the security of a pre-installed cryptographic key stored on the server. If that key is compromised, an attacker can decrypt the stored information and impersonate the server to send dangerous commands to the devices. Finally, the work did not specify what happens when the stored random numbers on the server are all consumed (since there is a finite number of values stored on the server). In the case where the protocol restarts from the beginning, it will be eventually subject to replay attacks.

Lessons

3. Authentication protocol designers should make sure that all involved authenticating parties use their own proper nonces during authentication sessions to make the latter (sessions) unique. Otherwise, replay attacks become feasible.

4. When adopting a countermeasure, protocol designers should verify whether the adopted countermeasures can be used against the system to cause a denial of service, e.g., not responding after a certain number of authentication failures. In fact, an attacker may have the intention to only make the system unavailable by abusing the countermeasure.

5. A secure system should not rely on a single secret information to protect a large number of confidential and critical assets (i.e., no single point of failure).

Energy-Efficient Authentication of Medical Devices

An FPGA PUF-based authentication protocol was proposed for medical IoT devices by Yanambaka et al., in [YMK+19]. The protocol allows a medical device to be authenticated to the server (one-way authentication). A prototype of the protocol was implemented using two FPGA-boards, a microcontroller (medical IoT device), and a Raspberry Pi (server). The protocol applies the classical challenge-response scheme but in an obfuscated way. In
fact, the server first generates a challenge $c$ and computes its response $r$ (i.e., $\Psi_s(c) = r$, where $\Psi_s(\cdot)$ is the server’s PUF) using its embedded PUF. The server sends the response to the device, which uses the response as a challenge to produce a response $r'$ using its PUF (i.e., $\Psi_d(r) = r'$, where $\Psi_d(\cdot)$ is the device’s PUF). The device sends back the response to the server, which uses $r'$ as a challenge for its PUF to produce a final response $r''$ (i.e., $\Psi_s(r') = r''$) and store the tuple or CRP $(c, r'')$. This protects the CRPs in the case where the server is compromised. In fact, if an attacker gains access to the stored CRPs, it will not obtain the device’s CRPs as the response $r''$ is not the response $r'$ of the device’s PUF when the latter (i.e., PUF) is interrogated using the challenge $c$.

However, the protocol contains security vulnerabilities that can be exploited to generate different attacks. The fact that the challenge and response values are exchanged in plaintext (although they are obfuscated), an attacker will be able to learn the obfuscated CRPs of the device and generate a PUF-modeling attack on the device. Also, if the attacker is an insider, it will be able to disclose the CRPs of the server. The attacker just has to keep track of the challenge values that it has sent to the server. Furthermore, the protocol is vulnerable to replay attacks. In fact, the exchanged messages do not contain any timestamp for data freshness and do not contain any information about the authenticity of the source, in particular, the server, as it is not authenticated to the device.

**Lessons**

6. When adopting a challenge-response scheme, the protocol should not exchange the PUF-responses in plaintext. In fact, this would allow an eavesdropper to intercept the challenge values and their corresponding response and disclose the CRPs. Also, by intercepting a large number of CRPs, PUF-modeling attacks become feasible. A straightforward approach would be to hash the PUF-responses along with nonces.

7. The CRPs should never be stored on any trusted location (e.g., trust center) in the form where they can be directly used for impersonation.

8. All authentication messages that are exchanged during an authentication session should be fresh through the use of nonces. Also, each party should use a nonce.
5.3. RECENT PUF-BASED AUTHENTICATION PROTOCOLS

Otherwise, previously exchanged messages (old messages) will be reused for impersonation.

Machine-Learning Resilient Authentication Protocol

Nozaki et al., [NY19b] proposed a XOR-arbiter PUF-based one-way authentication protocol that is resistant to machine-learning attacks. In this protocol, instead of having the verifier v send one challenge value \( c_p \) to the prover p (in plaintext) to receive the corresponding response value \( r_p \) (in plaintext), the verifier sends two challenge values \( c_p \) and \( c'_p \) to the prover p, which replies back with one value \( w \), called distributed value [Sha79]. This distributed value \( w \) is computed by combining portions of the two response values \( r_p \) and \( r'_p \) for the challenges \( c_p \) and \( c'_p \), respectively. The verifier stores one response and one distributed value \( w' \) that is used to verify the response of the prover. By adopting this obfuscation approach, the prediction rate for a successful machine-learning attack is considerably reduced. Thus, if an attacker intercepts the exchanged distributed value \( w \), it will not be able to learn the responses \( r_p \) and \( r'_p \) for the challenges \( c_p \) and \( c'_p \), respectively. The protocol was implemented on a Xilinx Virtex-5 XC5VLX30 FPGA-board. The protocol is vulnerable to spoofing and CRPs disclosure. In fact, the verifier is capable of constructing the CRPs of the prover (as it stores the first response \( r \) and can compute \( r' \) using the stored distributed value \( w' \)). Hence, it will be able to impersonate the prover. Also, if the verifier is compromised, the CRPs are disclosed.

Lessons

9. The concept of distributed value is a good obfuscation approach for hiding PUF responses. In fact, this cryptographic concept allows revealing part of the PUF responses and not the entire responses, which harden spoofing attacks.

10. The CRPs should never be stored on any trusted location in the form where they can be easily reconstructed and used for impersonation (i.e., by malicious insiders). We have observed that most protocol designers do not consider the case of malicious
5.3. RECENT PUF-BASED AUTHENTICATION PROTOCOLS

insider attacks, where a malicious party takes advantage of the trustworthiness of the system and its access rights to generate attacks.

CRPs Preserving Authentication Protocol

Chatterjee et al., [CGS+19] proposed a PUF-based authentication protocol for IoT. The protocol allows an IoT device (i.e., Thing) to get authenticated to a verifier (e.g., gateway) by using its proper PUF and applying elliptic curve operations (2 multiplications and 3 additions), pairing operations, and a hash function. It adopts a hierarchical architecture, where each verifier authenticates a set of IoT devices and all the verifiers communicate with a central root security association provider, which stores correlated information about the CRPs of IoT devices and their associated verifiers. The protocol was implemented using a 5-4 DAPUF (Double Arbiter-PUF with 5 inputs and a 4-bit output) on the Digilent Nexys-4 FPGA-board (which contains the Xilinx Artix-7-FPGA module) for a video surveillance system.

Notwithstanding, we have found that the protocol has some security issues, where attacks can be generated. For example, to prevent DoS (Denial of Service) attacks, the authors configured the devices (Things) to shutdown (or slowdown) the protocol execution (e.g., by adopting an exponential back-off algorithm) when a large number of requests are sent to the devices. This countermeasure can be exploited by an attacker whose intention is to shutdown the devices for some time. Also, the verifier is configured to reject any further request that regards the same pair for which it has previously received a request. This can be exploited by an attacker since the initial request is not authenticated. The attacker has to spoof a device $d_i$ and generate requests to devices $d_{j \neq i}$ before the legitimate device does (race-condition [LZ19a; LZ19c; LZ20a]). This would make the verifier reject any request coming from the legitimate device $d_i$ to communicate with another device $d_{j \neq i}$.
Moreover, the protocol is designed in such a way so that IoT devices (which are assumed to be resource-constrained) perform 2 elliptic multiplications, 3 elliptic additions, one pairing operation, and 7 hash operations, before being able to authenticate the verifier. An attacker can send regular challenge requests to force the device to perform unnecessary operations that drain their power-source (usually a battery).

Also, although the authors have suggested the use of a strong PUFs\(^3\) and to delete authentication information about provers from the verifier database, it is possible for a malicious insider verifier to spoof any prover that is associated to it. In fact, when a prover (denoted as \(A\)) starts an authentication, the verifier (denoted as \(S\)) retrieves authentication information, denoted as \((C_A, C_S, HLP_A, a, B, d)\), about the prover. This information can be used to impersonate the prover \(A\).

Finally, the authentication protocol assumes that the CRPs as well as security keys are stored in a secure “offline” database in a trusted environment. Also, it assumes that the secret keys of each prover (denoted as \(K_S\)) are stored in a secure non-volatile memory, i.e., tamper-proof NVM (which is an expensive approach). If these keys are compromised, the security of the system may be compromised as well.

### Lessons

11. Setting up a countermeasure that can be used against the system itself is a major vulnerability. Protocol designers should evaluate the inverse consequences of a countermeasure (Similar to Lesson 4).

12. Protocols should be designed in such a way so that devices can decide whether to consider/drop a particular message by just performing fast and costless operations. Otherwise, attackers would flood the system with encrypted bogus messages to be unnecessarily decrypted and processed before being dropped.

13. Authentication information, such as CRPs, should not be transmitted or stored in a form where they can be easily generated and reused for impersonation. This is actually a major security vulnerability that most PUF-based authentication protocols suffer from.

14. When designing a protocol, the cost of assumptions has to be estimated as certain assumptions are expensive and are not always feasible.
5.3. RECENT PUF-BASED AUTHENTICATION PROTOCOLS

Service-Centric RFID Authentication Protocol for IoT

A PUF-based authentication protocol was developed for RFID systems in the context of IoT by Liang et al., in [LSL+19]. The protocol allows an RFID-tag to be authenticated to a back-end server by proving the possession of the PUF and the authentication of the server to the tag by proving that it knows the PUF model of the RFID-tag. The protocol consists of two phases, the seed generation phase and the verification phase. During the seed generation phase, the server (verifier) and the tag (prover) synchronize on a seed to be used as input for a PRNG (Pseudo-Random Number Generator) to generate the challenges for the verification phase. At each verification session, the seed is used to generate a challenge. The challenge is fed to the PUF to compute the response. The prover divides the response into two parts and randomly pads each part with bits before sending the concatenation of the results. The verifier, which has computed the response using the PUF model and divided the response into two parts, searches for the presence of the two parts in the received sequence of bits. If both parts are found, the server (verifier) authenticates the RFID-tag (prover).

A similar process is conducted to perform the authentication of the server to the tag using another seed, which is the XOR of the two previous parts of the response. The authors proposed a two-stage multiple-choice arbiter PUF (TSMCA-PUF) to design the protocol. The proposed PUF is claimed to be resilient to modeling attacks. They used the BAN (Burrows-Abadi-Needham) logic to prove the correctness of the protocol and performed an informal security analysis of the protocol with respect to modeling attacks, man-in-the-middle attack, replay attack, CRPs disclosure, and spoofing attacks. A performance analysis was also performed. The protocol was implemented using the Virtex II-PRO FPGA-board and Xilinx XUPV5-LX110T development boards. Nevertheless, our analysis shows that it is possible to generate a de-synchronization attack through message modification.

Attack Scenario. During the seed generation phase, the server (verifier \( v \)) generates a nonce \( T_v \), constructs a message \( M_1 = \{ H(T_v) \oplus T_v \} \), and sends it to the RFID-tag (prover \( p \)).
tag generates a nonce $T_p$, constructs a message $M_2 = \{M_1 \oplus T_p\}$, and sends $\{M_2 \mid T_p\}$ to the server (where $\mid$ is the concatenation). The server computes $T'_v = M_2 \oplus T_p \oplus H(T_v)$. If $T_v = T'_v$, the server sets up the seed to be $s = M_2$. An attacker $e$ could have changed the tag message to $\{M_2 \oplus T_p \oplus T_e \mid T_p \oplus T_e\}$, which would have made the server set the seed to $M_2 \oplus T_p$ instead of $M_2$. In this situation, both the tag and the server will be synchronized over a different seed, which would result in an authentication failure. The system can only be recovered by resetting (re-synchronizing) the tag and the server to use the same seed.

Furthermore, the server keeps a PUF model of the registered RFID-tags on its database. Compromising the server (e.g., through a malicious insider) would result in the disclosure of the PUF models of the RFID-tags and hence their CRPs. Also, the server can spoof any registered RFID-tag as it knows how to generate all the CRPs of the RFID-tag.

**Lessons**

15. When an authentication protocol is based on synchronization, protocol designers have to make sure that information related to the synchronization has to have its integrity protected. Otherwise, any unauthorized modification would create desynchronization and hence a malfunction of the protocol.

16. Establishing authentication parameters using unauthenticated messages is a serious security issue when there is no way for the involved parties to verify and confirm that the parameters have not been modified.

17. The concept of distributed value, used in [NY19b], can be used as a good obfuscation approach to solve the issue of keeping the CRPs of registered devices secure (Similar to Lessons 7, 10, and 13).

**Single CRP-based Authentication Protocol for IoT**

Kim et al., [KYK+19] proposed a PUF-based authentication protocol to allow a smart device to be authenticated by a server. The protocol tries to reduce the load issue of storing all the CRPs of all registered devices on a particular server (i.e., trust center) by storing only one CRP, $(c_{d_i}, r_{d_i})$, per device $d_i$, at a time. Therefore, at the registration phase, only one CRP per device is saved into the database. The stored CRP for a given device is updated
5.3. RECENT PUF-BASED AUTHENTICATION PROTOCOLS

after each authentication. That is, when a device \( d_i \) is authenticated, it generates a new CRP, \((c_{d_i}, r_{d_i})\), and sends it to the server. A prototype of the protocol was implemented on an STM32F4 MCU-board. The solution does not actually eliminate the storage-issue of storing all CRPs on a server. It only reduces the amount of space occupied by the storage of the CRPs. In fact, after deleting a used CRP, the server keeps a list of challenge values, which grows proportionally with the number of authentication attempts. Furthermore, we have found that the protocol is vulnerable to certain attacks:

**Attack Scenario 1.** In this protocol, the device starts by requesting an authentication from the server by sending its identity. The server replies back by sending a challenge \( c_i \). The device responds to the server by computing the corresponding response \( r_i \) and generating a symmetric key \( K_0 = c_i + r_i \) to send an encrypted response message to the server. The response message contains a new CRP, \((c_j, r_j)\). The server computes the key, decrypts the message, and then verifies the value of the response. If the response is correct, the device is authenticated, the new CRP, \((c_j, r_j)\), is saved, and an acknowledgement is sent to the device. Thus, as the authentication of the device \( d_i \) is based on successfully responding to one challenge \( c_i \), an attacker can brute force the response \( r_i \) by generating all possible values for \( r_i \) until getting an acknowledgement from the server. This would eventually depend on the size of the response, which is in general smaller than the challenge. That is, if the responses are on \( k \)-bits, the attacker just has to try at most \( 2^k \) authentication attempts. Once the value of \( r_i \) is disclosed, the secret key \( K_0 \) is cracked. Interestingly, if the attacker forges its authentication response messages in such a way so that the new CRP value is nonsense, e.g., \((c_j, r_j) = (0, 0)\), the server would then save the incorrect CRP. This would result in all future authentication attempts from the legitimate smart device to fail until a CRP-updating procedure is initiated to update the stored CRP.

**Attack Scenario 2.** The server and the smart device are both subject to DoS attacks. With respect to the server, as authentication messages, sent from the device to the server, are
5.3. RECENT PUF-BASED AUTHENTICATION PROTOCOLS

completely encrypted, the server has to first decrypt the messages before authenticating the
source. Thus, if an attacker floods the server with bogus messages, the server will perform
decryption operations before dropping the messages. With respect to the device, as the
server is not authenticated to the device, an attacker can flood the device with spoofed new
CRP requests and force it to consume its battery to unnecessarily generate new CRPs (since
here the server is the initiator of the CRP-updating procedure).

Lessons

18. Protocols should use variables (e.g., PUF-responses) of larger sizes to harden
the feasibility of brute force attacks. Hash functions could be used to complement
and enlarge the size of the PUF-responses by keeping the size of the PUF-circuits
smaller.

19. Many IoT applications are built using resource-constrained devices that operate
on rechargeable batteries. If authentication protocols are designed in a way where de-
vices can be abused by performing unnecessary heavyweight computations, then the
power-source of these devices will be drained and their availability will be breached
(Similar to Lesson 12).

Smarthome Authentication Protocol for IoT

Mughal et al., [MLM+18] proposed PAS (PUF-based Authentication Scheme), an authenti-
cation protocol for smart devices in IoT smart home applications. It allows users, equipped
with smartphones, to be authenticated to a gateway so that they can send command-
messages to connected smart devices (e.g., a lightbulb or thermostat). However, the gateway
is not authenticated to the users (no mutual authentication). Also, the protocol assumes
that smart devices and the gateway share a pre-established symmetric key to encrypt the
communication (i.e., the command-messages sent by the user). If the key is compromised
(e.g., through device physical invasion), an attacker will be able to forge new command-
message (e.g., turn camera Off) as well as modify command-messages sent to smart devices
by the legitimate user (e.g., increase temperature instead of decreasing it). In fact, the
command-messages are not authenticated. The protocol is also vulnerable to DoS attacks. Just to mention two of them, in the first attack scenario, an attacker can conduct a battery-depletion attack [LZ20a]. As the authentication of a message can only be performed after decrypting the message, the attacker can flood smart devices with bogus messages and force them to perform unnecessary decryption operations, which may drain their batteries. The second scenario is performed by a malicious insider device. The insider requests the gateway to register a large number of smart devices which do not exist, filling up the entire registered device table. In such circumstance, the gateway will not accept any new device registration (until reset to factory).

Lessons

20. Relying on a security single point of security is a major flaw. We urge that protocol designers should perform a dedicated analysis where the dependence and relationship between systems’ assets is evaluated. For example, taking over an asset, e.g., a secret key, should not allow attackers to transitive take over the remaining assets (Similar to Lesson 5).

21. Message authentication code, in particular HMAC\(^a\), should be the last cryptographic operation to be performed after the encryption of a message and not the other way around. In fact, if devices have to decrypt a message to be able to verify the authenticity of the source of the message (i.e., verify the HMAC), then attackers may take advantage of this design flaw to overwhelm resource-constrained devices with nonsense messages forcing them to decrypt the messages and draining their batteries, which would make those devices become unavailable (Similar to Lessons 12 and 19).

\(^a\)HMAC (Keyed-Hash Message Authentication Code) provides a message integrity code that is computed over a message and using a shared authentication key. This allows the authentication of the source as well as the protection of the integrity of the message.

Neural Network-based Authentication Protocol for IoT

An arbiter PUF-based authentication protocol was proposed for the T2M authentication scheme by Yilmaz et al., in [YGH18]. Based on the use of a neural network model of the PUF, the protocol allows a verifier \(v\) to authenticate a prover \(p\) without having to store
the challenge and response pairs (CRPs) on the verifier’s memory (database). The protocol makes use of a device MAC address and a timestamp to obfuscate the PUF response by applying the RC5 cipher. They showed, using Zolertia RE-Motes, that the protocol performs better than DTLS (Datagram Transport-Layer Security) and UDP (User Datagram Protocol), in terms of memory and energy consumption. Nevertheless, the protocol relies on the assumption that creating a neural network model for a PUF is evident, which is not always the case. The authors have built a PUF-model for arbiter PUFs which are known to have the simplest structure. Also, the security of the protocol was not evaluated. We have found that the protocol has many security vulnerabilities that can be exploited to generate various attacks.

The authentication proof that is generated by the prover, denoted as $\alpha_d$, and the authentication proof that is generated by the verifier, denoted as $\alpha_v$, are exactly the same, i.e., $\alpha_d = \alpha_v = \text{RC5(MAC, } R_i \oplus T_d\text{)}$; where MAC is the 48-bit MAC-address of the device, $R_i$ is the device PUF’s response to a challenge $C_i$ and, $T_d$ the device’s timestamp. This technically means that there is no authentication. In fact, the verifier can just resend the authentication proof that it has received from the prover to prove its authenticity to the prover. An attacker would eventually do the same to bypass the authentication. In addition, these authentication proofs are computed using RC5 symmetric cipher along with a pre-shared symmetric key. The size of the key has not been reported, which raises the question of RC5 cracking possibility, and hence PUF-responses and CRP disclosure attacks.

The protocol is vulnerable to insider spoofing attack. A verifier knows the complete PUF model of the prover, it can generate all the CRPs of the prover. Hence, the verifier can easily impersonate the prover. Also, if the verifier is compromised, the PUF model of all registered provers will be disclosed and the security of the system is paralyzed.
5.3. RECENT PUF-BASED AUTHENTICATION PROTOCOLS

Lessons

22. The authentication proofs of the verifier and the prover to prove their identities to each other are identical $\alpha_d = \alpha_v$. This actually allows any device to impersonate the verifier and bypass the mutual authentication of the protocol. Authenticating parties should not be able to take advantage over one another to produce correct proofs. A straightforward approach would be to require each authenticating party to use a proper nonce, so that the proofs become unique per authentication session and per party (Similar to Lesson 2).

23. Since the verifier (server) stores neural network models of PUFs of all registered provers (devices), the protocol is vulnerable to insider attacks, as well as, to CRP disclosure through physical invasion. Whatever the level of trust that is assigned to the verifier, the latter should not be able to reconstruct the CRPs of any other device (Similar to Lesson 10).

Secret-Message Exchange Authentication Protocol

Idriss et al., [IB17] proposed a PUF-based authentication protocol with key exchange for resource-constrained systems, such as RFID systems. The protocol relies on a challenge-challenge scheme, where a verifier $v$ sends $m$ challenges (of size $n$), $\{c_1, \ldots, c_m\}$, to the prover which replies back with $2 \times m$ challenges, $\{(c_{1,1}, c_{1,2}), \ldots, (c_{m,1}, c_{m,2})\}$, such that for every challenge $c_i$ sent by the verifier the following equation should hold:

$$\Psi_p(c_i) = \Psi_p(c_{i,1}) \oplus \Psi_p(c_{i,2})$$

In the same way, the prover authenticates the verifier by sending $m$ challenges and obtaining $2 \times m$ challenges for verification. Basically, the authentication of the prover to the verifier relies on the prover proving that it has the authentic PUF, whereas the authentication of the verifier to the prover relies on the verifier proving that it has the model of the prover’s PUF to compute its CRPs. The authors analyzed the security of the authentication protocol against guessing and challenge collection attacks.

Nevertheless, based on the authors’ assumption of using the protocol for RFID systems
(resource-constrained systems), this protocol has the following issues: (1) It requires from the device (which models the RFID-tag in [IB17]) to send $3 \times m \times n$ bits (transmission) and receive $2 \times m \times n$ bits (reception) to complete an authentication, which seems to be large amount of bits to be transferred on an RFID system. (2) Given a challenge value $c_i$, the prover as well as the verifier, have to go through a loop to generate the values $c_{i,1}$ and $c_{i,2}$ such that the condition $\Psi_p(c_i) = \Psi_p(c_{i,1}) \oplus \Psi_p(c_{i,2})$ holds. The probability to generate the values of $c_{i,1}$ and $c_{i,2}$ so that the previous equation is satisfied is $(1/2)$ for one challenge $c_i$ and $(1/2)^{-m}$ for all the challenges $c_i, i \in \{1, \ldots, m\}$. This means that there is 50% chance to fall into an infinite loop when looking for $c_{i,1}$ and $c_{i,2}$ for a given challenge $c_i$, which would make the system nonresponsive. (3) The PUF model of the prover is stored at the verifier. This would allow the verifier to be able to spoof the prover. For example, if the verifier is a malicious insider, it will be able to impersonate the prover (tag). Also, since the PUF model is stored on the verifier as is, compromising the latter would disclose the PUF model and hence the CRPs of the prover, i.e., the tag.

**Lessons**

24. A performance analysis has to be performed to demonstrate that the protocol conforms the performance requirements of the application for which it was developed for. In fact, many applications use low-cost and resource-constrained devices that cannot implement and operate certain heavyweight computations (e.g., using ECC on passive RFID-tags).

25. A formal verification of the protocol, using well-established tools should be performed to determine whether the protocol terminates, e.g., Issue 2 might end up in an infinite execution.

26. This authentication protocol allows a verifier to store PUF-models about other devices’ PUFs. This assumes that all verifiers are trustworthy, which is not always true. Authentication protocols should be designed in a way where insiders are considered part of the attacker’s model (Similar to **Lesson 10**).
5.3. RECENT PUF-BASED AUTHENTICATION PROTOCOLS

5.3.2 T2T PUF-based Authentication Protocols

Vehicle-to-Grid Authentication Protocol

Bansal et al., [BNS+20] proposed a PUF-based authentication protocol with key establishment, called SUKA, for vehicles smartgrid infrastructures. The protocol uses PUFs to allow the authentication of electric vehicles (EVs) to aggregators (a.k.a., mediator) as well as to the grid-server (GS). This protocol can be perceived as a T2T authentication protocol as the electric vehicles as well as the aggregator are considered resource-constrained devices that embed a proper PUF circuit. The authors performed a formal security analysis using Mao-Boyd Logic to prove the security of their protocol. Nevertheless, we have found that the security of the protocol can be breached and attacks such as spoofing and message forging can be generated. In fact, we discovered that it is possible to crack the derived session key between an aggregator and server (denoted $S_k$ as in [BNS+20]) and use the key to disclose all the CRPs (denoted as $(C'', K'')$ in [BNS+20]) of an electric vehicle. The attack can be repeated on each vehicle to disclose its CRPs.

**Attack Scenario.** To understand the attack, Fig. 5.4 illustrates the authentication part between the grid-server and the aggregator. The attacker starts by intercepting all the exchanged messages during the authentication. It uses the second message sent from the server to the aggregator to extract the values of $C$ and $V$. From the value $C$ the attacker can learn how many challenges are being used, i.e., the value of $m$. Then, using the value of $m$, the attacker computes $K_0$ (the aggregator’s PUF response for the first challenge $C_0$) using the value $N$ (i.e., $K_0 = m \oplus K_0 \oplus m$). Next, the attacker intercepts the third message and extracts the value $N'$ to compute the nonce $N_C$ using the previously learned value of $K_0$ (i.e., $N_C = N' \oplus K_0$). At this stage, to compute the secret key, $S_K = F(K_0, N_B) \oplus F(K_0, N_C)$, the attacker just needs to find the value of the nonce $N_B$. To that end, the attacker can run an offline brute force attack on the used message authentication code (MAC) function on the second message. The attacker holds the code value, i.e., $MAC(ID_m \mid M \mid m \mid N_B)$,
as well as three out of four input parameters for the function. If the attacker cracks the nonce $N_B$, it will compute the secret key $S_K$ and decrypt all the vehicle’s CRPs that will be used next time (denoted as $(C^g, K^g)$) in [BNS+20]$^8$. In this case, the attacker will have to spoof the electric vehicle before the vehicle performs a new authentication with the system.

$^8$The reader might think that cracking a hash function could take a long time. However, these will depend on the size of the nonce $N_B$ and the type of the MAC-function. The nonce could be cracked in the same way as a Wi-Fi WPA-key is cracked after capturing the handshake packets.
5.3. RECENT PUF-BASED AUTHENTICATION PROTOCOLS

otherwise, the CRPs will be updated at the server-side in the next authentication session.

Furthermore, the electric vehicle authenticates an aggregator by having the vehicle perform computations and infer that the aggregator knows the vehicle CRPs, which assumes that the aggregator already performed an authentication to the server and obtained the vehicle CRPs. In this case, there are two issues. The first issue is that if an attacker performs the attack described above, the attacker will be able to impersonate an aggregator and scam the vehicle. The second issue is that the grid server is considered a trusted party that stores “in plaintext” the CRPs of all registered electric vehicles as well as their identities. Therefore, if the server is compromised (e.g., through a malicious insider), all the CRPs are disclosed and the security of the system is completely breached. Finally, it was claimed that the protocol preserves the identity of the vehicle owners to protect them from being tracked. However, the identity of the vehicle owners is sent in plaintext when initiating an authentication with an aggregator. This would allow an attacker to keep track of the owners, which invades the privacy of certain vehicle owners.

Lessons

27. The XOR operator should not be applied to obfuscate secret information using easily derivable parameters. The XOR is a commutative and absorbent operator. In fact, $\forall x, y, z \mid z = x \oplus y$, then $z \oplus y = y \oplus z = x$. Thus, if we assume that $x$ is a secret, there should be no way for the attacker to learn the value of $y$ in the case where the attacker intercepts $z$.

28. Use automatic security protocol provers, e.g., TAMARIN, to check whether there is any vulnerability related to how the protocol applies the XOR operator.

29. The CRPs should never be stored in plaintext in any location, whatever is the trust of that location. This flaw is usual as many security protocol designers do not consider the case of insider attacks.

Synchronous Authentication Protocol

Barbareschi et al., [BBM18] proposed PHEMAP (PUF-based Mutual Authentication Protocol), an SRAM PUF-based protocol designed for terminal to terminal and terminal to
5.3. RECENT PUF-BASED AUTHENTICATION PROTOCOLS

It is based on the notion of chains, which are sequence of values generated by recursively invoking a PUF on a device. The chain is used to authenticate a device by other authenticating parties, which have to store parts of that chain, called links. The protocol strongly relies on synchronization. For example, at a given time $t$, if a device $v$ (verifier) wants to authenticate another device $p$ (prover), then the verifier $v$ has to be synchronized with the prover $p$. This synchronization consists of having the verifier $v$ store a specific link $\ell^v_t$ about the prover $p$ and having the prover $p$ be aware that the verifier $v$ is currently storing the link $\ell^v_t$. In 2019, the protocol was extended and adopted for cloud-edges IoT systems [BBM+19]. The protocol was implemented using the STMicroelectronics STM73F7-board (for terminals) and the Cubieboard Cubietruck-board (for gateways). We have found that the protocol is vulnerable to desynchronization attacks. In the following, we discuss two attack scenarios:

**Figure 5.5:** Dysynchronization attack on PHEMAP protocol [BBM18], where $M(v)_{x \rightarrow y}$ designates a message $M$ containing the value $v$, sent from authenticating party $x$ to $y$. The value $\ell^p_t$ represents the link of the prover $p$ at time $t$.

**Attack Scenario 1.** During the verification phase of the authentication, an attacker may spoof a prover $p$ (e.g., terminal) and send an incorrect response to the verifier $v$ (e.g.,
5.3. RECENT PUF-BASED AUTHENTICATION PROTOCOLS

gateway) before the legitimate prover does. Upon sending the correct response by the prover
p, the latter updates its link (e.g., \( \ell_p^t \)) to the next value on the chain (e.g., \( \ell_{t+1}^p \)). Then, when
the verifier \( v \) receives the incorrect response, it drops the message and unauthenticates the
prover \( p \) even if it receives the correct response later on. At that time, the prover \( p \) and the
verifier \( v \) are synchronized over a different value of the link (\( p \) stores the link \( \ell_{t+1}^p \), whereas
\( v \) stores the link \( \ell_t^v \)). Thus, any future authentication attempt would fail. This attack is
illustrated in the MSC of Fig. 5.5. Also, we note that this attack scenario can be generated
during the initialization phase of the protocol by spoofing the second message.

**Attack Scenario 2.** If we assume that the links are encoded on \( n \)-bits, then at a given time \( t \),
the attacker may spoof the verifier \( v \) and try all possible values for the current link \( \ell_t^v \) (i.e., \( 2^n \)
combinations). The attacker stops the attack when hitting the correct value of \( \ell_t^p \) (i.e., \( \ell_{t+1}^p \)).
When the correct value is hit (viz., the first message in the MSC of Fig. 5.5), the prover
\( p \) authenticates the verifier \( v \) and updates the current link to \( \ell_{t+2}^p \), which desynchronizes
the prover and the verifier. Although this second scenario seems to be time-consuming, its
impact is worse than the one of the first scenario. In fact, after this attack scenario, the
prover \( p \) will be storing the link \( \ell_{t+2}^p \), whereas the verifier \( v \) will be storing the link \( \ell_t^v \). The
attacker can repeat this attack scenario multiple times in such a way so that the current
chain gets invalidated.

**Lessons**

30. Synchronization information should not be updated before making sure that
authentication is established. For example, in this protocol, a device updates the
synchronization information “link” before confirming that it is communicating with
the right counterpart, which caused the desynchronization of both devices, and hence
a denial of service attack.

31. The PUF challenges and responses should be large enough to harden the feasibil-
ity of brute force attacks (Similar to Lesson 18). Also, as it is more common to
adopt a countermeasure that increases the complexity of a brute force attack, e.g.,
by limiting the number of requests, protocol designers should be careful when adapting
a countermeasure that can be used to abuse the system’s availability.
Secure Wi-Fi Authentication Protocol

A PUF-based authentication protocol was proposed for secure Wi-Fi authentication of IoT devices by Mahalat et al., in [MSM+18]. The protocol is adapted for a T2M authentication scheme, where a Wi-Fi IoT device gets authenticated to a Wi-Fi router. This protocol aims to add to the existing security mechanisms, e.g., WPA2 (Wireless Protected Access 2), a physical security to mitigate existing MAC-layer attacks. For each registered Wi-Fi device, the protocol stores three CRPs on the Wi-Fi router. The router uses all three CRPs to authenticate the device (the device infers that it is communicating with the legitimate router as the router knows the device’s CRPs). Also, the CRPs are updated after each authentication to prevent replay attacks (in case the same nonce is used twice or the nonce is brute-forced). Similar to other protocols, this protocol stores the CRPs of registered Wi-Fi devices into the Wi-Fi router database in a plaintext format. Thus, if the access point is compromised, the CRPs of the registered Wi-Fi devices will be disclosed and the affected devices can be spoofed. Furthermore, the security of the protocol can be entirely compromised by observing two authentication sessions from the same Wi-Fi device as explained in the next paragraph.

Attack Scenario. The protocol is illustrated by the MSC of Fig. 5.6. We can see that the Wi-Fi router (verifier v) stores three pairs of challenge and response (\{(C_1, R_1), (C_2, R_2), (C_3, R_3)\}) for a given Wi-Fi device (prover p). Then, upon receiving a request from a device, the router generates a random value nonce and uses the CRPs to send the values
\[
C_1' = C_1 \oplus \text{nonce}, \quad C_2' = C_2 \oplus \text{nonce}, \quad C_3, \quad R_2' = R_2 \oplus \text{nonce}, \quad \text{and} \quad R_3' = R_3 \oplus \text{nonce}.
\]
The Wi-Fi device receives the values, performs computation to extract the value nonce, and verifies the authenticity of the router by checking that \(\Psi_p(R_2) = R_2' \oplus \text{nonce}\), where \(\Psi_p(\cdot)\) is the device’s PUF. The device performs other computations and sends the values
\[
R_{1\text{new}}' = \Psi_p(C_1 \oplus C_3) \oplus \text{nonce}, \quad R_{2\text{new}}' = \Psi_p(C_2 \oplus C_3) \oplus \text{nonce}, \quad R_{3\text{new}}' = \Psi_p(C_3 \oplus \text{nonce}) \oplus \text{nonce}, \quad R_1' = \Psi_p(C_1) \oplus \text{nonce},
\]
and a hash (\(H_{\text{client}}\)), as shown in the MSC of Fig. 5.6. When the router receives these values,
it authenticates the Wi-Fi device by checking that $R'_1 \oplus \text{nonce} = R_1$. Then, it updates the CRPs as shown in Fig. 5.6. We can observe that the strength of the protocol resides in keeping the value $\text{nonce}$ secret at each authentication session. However, since the new CRPs are computed based on the previous CRPs, finding the value of $\text{nonce}$ will allow the attacker to infer all the original values of the CRPs (i.e., $C_1$, $C_2$, and $C_3$) and hence infer the current value of the $\text{nonce}$. To that end, an attacker intercepts and records the
Figure 5.7: Attack on protocol by Mahalat et al., [MSM+18]. For clarification, the blue values are captured by the attacker during a previous authentication session (viz., Fig. 5.6), whereas the purple values represent the values that the prover computes in the current session. The black values are the stored and transmitted ones. As can be seen at the end of the attack, the attacker has the CRPs of the prover that will be used in the next authentication session. The attacker can spoof both the router and the device, that can cause denial of service and generate more sophisticated attacks.
exchanged messages during an authentication between the device and the router. Since the parameter \( C_3 \) is sent in plaintext during the authentication and the next new value for \( C_3 \) is \( C_3 \oplus \text{nonce} \), the attacker will be able to compute the value \( \text{nonce} = C_3 \oplus \text{nonce} \oplus C_3 \). Using the value \( \text{nonce} \) and the intercepted values \( C_1', C_2', R_2' \) and \( R_3' \), the attacker infers the value of \( C_1, C_2, R_2, \) and \( R_3 \). This would allow the attacker to infer the new value of the nonce and compute the next CRPs of the device to mount other attacks as illustrated in the last part of the MSC of Fig. 5.7.

32. The verifier (access point) stores the CRPs of the registered Wi-Fi stations in plaintext. This is fundamentally insecure as it allows attacker to disclose the CRPs using different types of attacks, as well as, performing spoofing attacks in the case where the access point is hijacked. A secure PUF-based authentication protocol should store the CRPs in a way so that it cannot use them directly. For example, the concept of distributed value, as adopted in [NY19b], can be used as a good obfuscation approach to solve this issue.

33. In this protocol, the authentication responses used in a former authentication session are dependent on the authentication responses that are used in a later authentication session. This creates the possibility of generating replay attacks. Therefore, authentication protocol designers should ensure that authentication responses from one session to another differ significantly and cannot be reused (Similar to Lessons 1 and 2).

34. Incorrect usage of the XOR operator can affect the security of the entire protocol. Protocol verifiers such as AVISPA, TAMARIN, and ProVerif, can verify the correctness of the usage of the XOR operator (Similar to Lessons 27 and 28).

Lightweight Attestation and Authentication Protocol

Feng et al., [FYZ+18] proposed a PUF-based lightweight attestation and authentication protocol, called AAoT, for IoT (Internet of Things) and CPS (Cyber-Physical Systems). The protocol uses PUFs along with fuzzy extractors to derive a secret key that is used to establish mutual authentication between devices. SRAM-PUFs were adopted to implement the protocol on MSP-EXP430G2 development boards. The protocol is vulnerable to
5.3. RECENT PUF-BASED AUTHENTICATION PROTOCOLS

spoofing attacks, CRP disclosure, and replay attacks. In fact, the protocol stores the CRPs of the prover on the verifier database. This would allow the verifier (in case the verifier is a malicious insider) to impersonate the provers whose CRPs are stored on the verifier’s database. If the verifier is compromised (e.g., through invasion attack), the provers’ CRPs are disclosed.

The prover relies on a 16-bit random number generator (RNG) along with a response of a challenge $c$ (sent by the verifier) to operate a fuzzy extractor module and produce a helper data $h$ as well as a secret key $K$. The helper value $h$ is sent to the verifier which uses it along with the response $r$ (as it has the prover’s CRPs) to compute the key $K$. An attacker can intercept an authentication session $S_j$ and use the intercepted parameter (e.g., $c_j, h_j,$ and $M_j$) to spoof the verifier in another session $S_k$. To that end, the attacker employs brute-forcing and challenges the prover with the same challenge $c_j$ (intercepted during session $j$) until the prover responds with $h_j$. In that case, the attacker will be able to correctly respond using $M_j$ (intercepted during session $j$) and get authenticated. The attacker will just have to challenge the prover at most $2^{16} = 65536$ times with the same challenge $c_j$. This attack is possible as long as the prover does not implement any mechanism to limit the number of challenges with the same challenge in short time duration.

---

**Lessons**

35. CRPs stored in plaintext make the protocol vulnerable to CRP disclosure and PUF impersonation. Obfuscation techniques, such as distributed value (adopted in [NY19b]), can be used to secure the CRPs of other devices when they are stored or transmitted (Similar to Lessons 7, 10, and 29).

36. PUF responses are expressed in 16 bits, which is relatively small to thwart brute force attacks. In fact, the attacker will have to try at most 65536 values before revealing the correct PUF response. This would take few seconds on a modern computer. Therefore, protocol designers should consider at least 64 bits, to increase the temporal complexity for performing brute force attacks (Similar to Lesson 18).
Third Trusty Party-based Authentication Protocol

Clupek et al., [CZ16] proposed a PUF-based mutual lightweight authentication protocol for T2T authentication schemes. It uses a third trusty party (TTP) as a trusted authority to establish a robust authentication between Things. It relies on the use of low-cost PUFs, simple hash functions, and the binary eXclusive-OR operator. We have found that this protocol is vulnerable to certain attacks as follows:

\[
\begin{align*}
M(0x01 | \text{Id}_v | c_p | h_1) & \rightarrow p \\
M(0x02 | \text{Id}_p | r_c | c_v | h_2) & \rightarrow v \\
\text{Ignore} \\
\text{Extract } s' \text{ from } p' \\
M(0x04 | \text{Id}_p | c_v | p | h_4) & \rightarrow p \\
\text{Ignore} \\
\text{Secret } s \neq s \\
\text{Secret } s
\end{align*}
\]

Figure 5.8: Compromising the secret information \( s \) in Clupek et al., protocol [CZ16]. The communication channel is assumed to be insecure. The notation \( M(v)_{x \rightarrow y} \) designates a message \( M \) containing the value \( v \), sent from authenticating party \( x \) to \( y \). The symbol “|” represents the concatenation of bits. We denote by \( a \approx p \), an attacker \( a \) that is spoofing a prover \( p \).

**Attack Scenario 1.** During the authentication, the authenticating parties exchange their messages (5 messages) over an insecure channel in plaintext. An attacker can disclose the secret information \( s \) and breach the confidentiality of the system. In fact, if the attacker intercepts the second messages of the protocol, it captures the response \( r_p \) that was sent by the prover \( p \) to the verifier \( v \). Then, by intercepting the fourth message, which contains the value \( s \oplus H(r_p) \), the attacker would be able to compute the secret information \( s \) by
5.3. RECENT PUF-BASED AUTHENTICATION PROTOCOLS

Evaluating $s \oplus H(r_p) \oplus H(r_p)$, where $H(\cdot)$ is a public hash function. In this case, the secret information $s$ has no longer become secret to the attacker (i.e., no confidentiality).

**Attack Scenario 2.** During the authentication, the challenge values and their corresponding response values are exchanged in plaintext. This would allow an attacker to intercept these values and perform a machine-learning attack [RSS+13; TG15; GSS15]. In this attack, the attacker learns different challenge and response values (i.e., CRPs) and build a perception model of a given PUF. Using the constructed perception model, the attacker can impersonate the concerned PUF of a target device. This would compromise the protocol.

**Attack Scenario 3.** As an attacker can read the content of all messages exchanged during an authentication, the attacker can eventually tamper-with the secret information $s$. In fact, by intercepting the challenge $c_p$ (from message 1) and its corresponding response $r_p$ (from message 2), the attacker would be able to forge the fourth message that contains the secret information $s$ and include another information $s'$. The attacker just has to make sure that its message is received the first before the legitimate verifier sends it secret information. This attack is possible as there is no way to authenticate the source of a given message. Also, the hash function used for message integrity does not use any secret. It can be computed by any party, including the attacker. This attack scenario is illustrated in the MSC of Fig. 5.8.

**Lessons**

37. The XOR operator can be used to brake the security of the protocol and reveal the secret parameter $s$ due to its incorrect usage. This bitwise operator provides properties that allow the implementation of secure and lightweight cryptographic operation. However, it can destroy the security of the protocol if it is used incorrectly (Similar to Lessons 27, 28, and 34).

38. Transmitting the CRPs in plaintext allows eavesdroppers to construct a model of the concerned PUF function by just intercepting a certain number of CRPs, depending on the type of PUF that is being adopted. The concept of distributed value, adopted in protocol Nozaki et al., [NY19b], is an interesting concept for obfuscating the PUF responses when being transmitted. Also, sending the hash of the PUF response along with nonce would be a good approach as well (Similar to Lesson 13).
5.4 Discussion

In this section, we further discuss the lessons that we have drawn from the previous protocols’ analysis. We summarize the major lessons learned from the most common security design flaws as follows:

**CRP Disclosure Flaw.** We have observed that many PUF-based authentication protocols suffer from CRPs disclosure attacks and PUF impersonation (e.g., attacks on protocols proposed in [BNS+20; YMK+19; NY19b; CGS+19; LSL+19; FYZ+18; YGH18; IB17]). This issue is due to the incorrect design of the part of the protocol that is responsible for keeping the CRPs of registered devices secure when these CRPs are transmitted or stored. This actually could be an interesting research direction to design PUF-based authentication protocols with a focus on securing the CRPs of registered devices from attackers. This would make the protocols resilient to CRP disclosure and PUF’s impersonation through malicious insider attacks. Also, we urge that authentication protocol designers should use security protocol verifiers, such as Isabelle/HOL, Tamarin, and ProVerif, to prove security properties in their protocols. For example, one can verify the secrecy of the CRPs to check whether it is possible for an attacker to reveal the CRPs of other devices.

**XOR Operator Misuse Flaw.** We have found that various protocols adopt the logical bitwise exclusive OR operator (i.e., XOR, denoted by \( \oplus \)) as a secure operator to perform lightweight computations. Notwithstanding, if this operator is used in an incorrect way, e.g., with easily deducible parameters to protect a credential, then this operator becomes the key for revealing other related credentials in a transitive way. In fact, due to the absorption property of XOR, publicly-known parameters can easily be eliminated from a logical expression that was computed using XOR, disclosing the values of other parameters, which could be secret parameters (e.g., attacks on protocols proposed by Bansal et al., [BNS+20], Mahalat et al., [MSM+18], and Clupek et al., [CZ16]). Therefore, authentication protocol designers should be really careful when using this powerful logical operator. Automatic
security protocol verifiers can be used to identify security flaws resulting from the incorrect usage of the operator.

**Authentication Session Uniqueness Flaw.** We have identified a common protocol-design flaw when it comes to the use of nonces. We have found that in some authentication protocols, the nonces are not used per authenticating party to make an authentication session unique w.r.t. the involved variables. This opens the possibility of generating replay attacks. Also, in some protocols, the nonces from different authenticating parties are related (dependent), which makes it possible for an attacker (spoofing a prover) to generate authentication proofs by exploiting an authenticating party, usually the verifier, and using the obtained proofs to bypass authentication (e.g., attack on Boyapally et al., [BMP+20]).

**Countermeasure Inverse Consequences Flaw.** Some authentication protocols (e.g., Chatterjee et al., [CGS+19] and Qureshi et al., [QM20]) implement a security measure (countermeasure) to mitigate or slowdown certain unusual and “suspicious” activities when the latter (activities) are detected. For example, when a large amount of authentication requests (or messages, in general) coming from the same source is detected, the system starts ignoring any new message coming from that suspicious source. Notwithstanding, an attacker could take advantage of this countermeasure to attack the availability of the system by spoofing legitimate devices and flooding the system with bogus messages. This would make the system blacklist those legitimate devices and ignore any message coming from them. Therefore, protocol designers should evaluate the impact of any countermeasure to check whether the countermeasure can be used against the system.

**Critical Resource Abusing Flaw.** Certain authentication protocols (e.g., Chatterjee et al., [CGS+19], Kim et al., [KYK+19], and Mughal et al., [MLM+18]) are designed in such a way so that devices operate heavyweight computations (e.g., message decryption) before being able to determine whether a given message is to be considered or ignored (dropped). In this case, resource-constrained devices that operate on batteries of limited lifetime can
be abused for the purpose of draining their batteries (battery depletion attack \[LZ20a\])
by flooding those devices with bogus encrypted messages. Therefore, authentication pro-
tocols should be designed in such a way so that devices can easily and with the lowest
cost determine whether to process or ignore a received message. For example, a message
authentication code can be verified by computed using an HMAC function.

Last but not least, we point out that all reviewed protocols are vulnerable to connection
deprievation through race-condition attacks (cf. Subsection 4.2.1 in the previous chapter).
In fact, existing authentication protocols, in general, and the reviewed authentication pro-
tocols, in particular, follow a state machine that transits from one state to another based
on the first unauthenticated message that is received. That is, if the protocol is in a state
of expecting the reception of a message, then it will transit to another state upon the re-
ception of that message so that it processes the message and takes further decisions (e.g.,
reply or abort). This sounds totally consistent with respect to standard protocol behaviors.
Notwithstanding, if we consider an attacker model where it is possible for an attacker to
interfere during the execution of the protocol, then the message that is received could be
a modified copy (e.g., containing incorrect values) of the expected message. Consequently,
if the modified message is received before the genuine one, then the receiving device will
be misled to fail the execution of the protocol (e.g., incorrect password derivation). This
generally ends up on the occurrence of a denial of service attack, where devices are deprived
from being able to successfully establish authentication and get connected to a network as
it was demonstrated in \[LZ19c\]. Thus, future authentication protocols should implement a
countermeasure to thwart these attacks.

5.5 Conclusion

In this chapter, we have mainly focused on reviewing the security of recent PUF-based
authentication protocols. For each protocol, we have demonstrated the feasibility of some
attacks. We have drawn lessons and proposed recommendations to be considered while developing future PUF-based authentication protocols so that the future protocols can be free from the identified security design flaws. Then, we have summarized the major lessons by presenting five common design flaws: (1) CRP disclosure flaw, (2) Xor operator misuse flaw, (3) authentication session uniqueness flaw, (4) countermeasure inverse consequences flaw, and (5) critical resource abusing flaw.

Finally, based on the reviewed PUF-based authentication protocols and based on the security lessons that we have drawn from analyzing these protocols, in the next chapter, we design and implement a secure PUF-based mutual authentication protocol for Thing-to-Thing communication. The protocol will be free from the security issues that we have discussed in this chapter.
Chapter 6

A Thing-to-Thing Mutual Authentication Protocol

In this chapter, we design and implement a generic PUF-based mutual authentication protocol for Thing-to-Thing (T2T) communications. We analyze the security of the authentication protocol and show that it is resilient to various known attacks. Also, we conduct a performance evaluation of the authentication protocol and show its efficiency w.r.t. execution time, communication overhead, and energy consumption.

6.1 Introduction

Although many authentication protocols have been proposed in the literature during the past decade, most of them do not fulfill the IoT security and performance requirements. Furthermore, only a very small number of these protocols can be used in Thing-to-Thing (T2T) architectures, where Things autonomously authenticate each other without involving any human intervention. Therefore, the need to design secure and lightweight T2T authentication protocols that conform to IoT security and performance requirements has become one of the top priorities for IoT security.

In the previous chapter, we analyzed recent PUF-based authentication protocols and drawn security lessons to consider in future protocol designs. Based on the learned security lessons and the reviewed authentication protocol, in this chapter, we develop a novel and generic lightweight T2T mutual authentication protocol (T2T-MAP). The protocol employs...
6.2. DESIGN OF THE AUTHENTICATION PROTOCOL

PUFs technology to allow each Thing to uniquely identify and authenticate itself in an IoT infrastructure by using the physical randomness of its circuitry. We design the protocol and perform a security analysis to show that it is secure against known attacks. Also, we prove the security of the protocol using a security protocol verification tool. We implement a prototype of the protocol on resource-constrained devices and then conduct performance analysis to show that the protocol allows fast authentication, reasonable communication overhead, and low energy consumption.

Finally, to broaden the usability and application of the protocol that we develop in this chapter, we keep the protocol as generic as possible and do not make it dependent on any wireless communication technology. In this way, the protocol can be used on any wireless communication technology, including those considered in Chapter 2.

6.2 Design of the Authentication Protocol

We design T2T-MAP following two configurations. The first configuration allows two Things to mutually authenticate each other via a central gateway. The second configuration however, allows two Things to directly authenticate each other in the absence of a gateway. We start by introducing the concept of eCRPs (extended CRPs) that T2T-MAP uses for authentication. Then, we present the phases of T2T-MAP for its establishment. Next, we discuss the security and performance properties and analyze the security of T2T-MAP against known attacks. Table 6.1 reports a summary of the adopted notation.

6.2.1 The Concept of Extended CRPs (eCRPs)

If the CRPs (Challenge-Response Pairs) of registered Things are stored on a verifier in plaintext, they are subject to physical invasion attacks and insider spoofing, where they can be disclosed. Also, if for each registered Thing a verifier stores a large set of CRPs, the whole system becomes less efficient and not scalable as Things have limited storage
6.2. DESIGN OF THE AUTHENTICATION PROTOCOL

Table 6.1: Summary of the notation adopted in the proposed authentication protocol.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_i$</td>
<td>Thing $T_i$.</td>
</tr>
<tr>
<td>$T_0$</td>
<td>The Gateway.</td>
</tr>
<tr>
<td>$T_{ij}$</td>
<td>Thing $T_i$ and $T_j$, respectively.</td>
</tr>
<tr>
<td>$x_i^v$</td>
<td>The $v^{th}$ challenge $x$ generated by Thing $T_i$.</td>
</tr>
<tr>
<td>$\alpha_{ij}^v$</td>
<td>The $v^{th}$ obfuscated response $\alpha$ used by a Thing $T_i$ to verify the authenticity of Thing $T_j$.</td>
</tr>
<tr>
<td>$\Psi_i(\cdot)$</td>
<td>The PUF function of Thing $T_i$.</td>
</tr>
<tr>
<td>$\Psi_i(x_i^v)$</td>
<td>The response of the PUF $\Psi_i(\cdot)$ on the $v^{th}$ challenge $x_i^v$.</td>
</tr>
<tr>
<td>$\beta_{ij}^v$</td>
<td>The $v^{th}$ obfuscated response $\beta$ used by a Thing $T_j$ to prove its authenticity to Thing $T_j$.</td>
</tr>
<tr>
<td>$H(\cdot)$</td>
<td>A public hash function.</td>
</tr>
<tr>
<td>$\oplus$</td>
<td>The exclusive OR operator ($x \oplus y = x \cdot y + \overline{x} \cdot y$).</td>
</tr>
<tr>
<td>$\Gamma_i(T_j)$</td>
<td>The extended CRP (eCRP) used by Thing $T_i$ to challenge and verify the authenticity of Thing $T_j$.</td>
</tr>
<tr>
<td>$N_i$</td>
<td>A nonce generated by Thing $T_i$.</td>
</tr>
<tr>
<td>$N_{ij}$</td>
<td>The XOR of Nonces $N_i$ and $N_j$, i.e., $N_i \oplus N_j$.</td>
</tr>
<tr>
<td>$H(*)$</td>
<td>The hash of a message $m$ (payload) for integrity protection.</td>
</tr>
<tr>
<td>$K$</td>
<td>A cryptographic key.</td>
</tr>
</tbody>
</table>

capacity. Therefore, to thwart these security and performance issues, we have adopted the following two approaches:

**First Approach.** We have transformed the classical CRP structure from a 2-tuple $(c, r)$ to a 5-tuple $(x_1, x_2, x_3, \alpha_1, \alpha_2)$, which we have called the extended CRP, or eCRP for short. We denote the eCRP by the letter $\Gamma$. For example, if a Thing $T_i$ stores an eCRP about Thing $T_j$, we write $\Gamma_i(T_j)$. The content of an eCRP $\Gamma_i(T_j)$ consists of three challenge values, $x_1^i, x_2^i, x_3^i$, and two obfuscated response values $\alpha_{ij}^1$ and $\alpha_{ij}^2$. These obfuscated response values are computed using the challenges and the PUF functions of both involved Things as expressed in the following two equations, where $\Psi_i(\cdot)$ and $\Psi_j(\cdot)$ are the PUF functions of Thing $T_i$ and $T_j$, respectively.

$$\alpha_{ij}^1 = \Psi_j(\Psi_i(x_1^i)) \oplus \Psi_j(\Psi_i(x_2^i)) \quad (6.1)$$

and

$$\alpha_{ij}^2 = \Psi_j(\Psi_i(x_1^i)) \oplus \Psi_j(\Psi_i(x_3^i)) \quad (6.2)$$
Under this form, the obfuscated response values $\alpha_{ij}^1$ and $\alpha_{ij}^2$ can only and exclusively be used by Thing $T_i$ and $T_j$, and cannot be used by either of them to impersonate the other (e.g., in case of a malicious insider). Moreover, if a Thing $T_i$ is compromised, the values of $\alpha_{ij}^1$ and $\alpha_{ij}^2$ cannot be used by another Thing $T_k$ to impersonate $T_i$ or $T_j$.

We note that this method of obfuscating the response by combining two responses together into one single response is inspired by the concept of distributed value used in cryptography [Sha79]. Also, the idea of using the PUF function of both authenticating parties to construct responses is inspired by the work in [YMK+19].

Second Approach. Considering the limited storage-capacity of resource-constrained devices, we only store one eCRP per Thing on a Thing’s memory. That is, if a Thing $T_0$ is configured to authenticate $n$ other Things $T_i \neq 0$, it only stores $n$ extended CRP per Thing $T_i \neq 0$. However, we note that the eCRP is constructed based on three classical CRPs as we have used three challenges, $x_{i1}^j, x_{i2}^j$ and $x_{i3}^j$, and computed the obfuscated response values of $\alpha_{ij}^1$ and $\alpha_{ij}^2$ by combining their responses. This way of storing the eCRPs allows using a given challenge-response pair more than once. In fact, it is not the case in classical PUF-based authentication protocols, where a strong PUF is required to generate a large number of CRPs, and each CRP has to be used only once.

6.2.2 T2T-MAP Phases

Similar to all other PUF-based authentication protocols, T2T-MAP consists of two phases: the enrolment and the authentication phase. In this subsection, we present how the enrolment phase occurs in T2T-MAP, then show the verification and authentication. It is important to note that in addition to performing authentication, T2T-MAP allows the establishment of a symmetric cryptographic key between two Things once mutually authenticated.
6.2. DESIGN OF THE AUTHENTICATION PROTOCOL

Figure 6.1: The enrolment phase of the developed PUF-based T2T authentication protocol. During this phase Things $T_1$ and $T_2$ exchange challenges and responses to construct each an extended CRP (a 5-tuple), denoted by $\Gamma_1(T_2)$ ($\Gamma_2(T_1)$, respectively). The 5-tuple $\Gamma_1(T_2)$ ($\Gamma_2(T_1)$, respectively) stores 3 challenges, $x_1^1$, $x_2^1$, and $x_3^3$ ($x_1^2$, $x_2^2$, and $x_3^2$, respectively), and 2 authentication values, $\alpha_1^{12}$ and $\alpha_2^{12}$ ($\alpha_1^{21}$ and $\alpha_2^{21}$, respectively). These two values are computed using the locally generated challenges and the PUF functions of both Things, i.e., $\Psi_1(\cdot)$ and $\Psi_2(\cdot)$.

### Enrolment Phase

This setup phase is conducted in a secure environment, where two Things (here the gateway is represented by a Thing, e.g., $T_j$), $T_i$ and $T_j$, are brought next to each other to exchange 3 challenges and reveal 2 obfuscated responses each. Both Things, $T_i$ and $T_j$, start by generating random challenges, $x_1^{ij}$, $x_2^{ij}$, and $x_3^{ij}$, then use their local PUF functions, $\Psi_i(\cdot)$ and $\Psi_j(\cdot)$, to compute the responses $\Psi_i(x_1^{ij})$, $\Psi_i(x_2^{ij})$, and $\Psi_j(x_3^{ij})$. Once computed,
6.2. DESIGN OF THE AUTHENTICATION PROTOCOL

these responses constitute the new challenges to be sent to the other Thing. Each Thing $T_{ij}$ sends the new challenges to the other Thing $T_{ji}$. Upon receiving the challenges, each Thing uses its proper PUF to compute 3 response values over the received challenges, i.e., $$\Psi_{ji}(x_{ij}^1), \Psi_{ji}(x_{ij}^2), \text{ and } \Psi_{ji}(x_{ij}^3).$$ These responses are then used to compute two values $\alpha_{ij}^{i}$ and $\alpha_{ij}^{j}$. By xoring the first response with the second one, the value of $\alpha_{ij}^{i}$ is computed. Also, the value of $\alpha_{ij}^{j}$ is computed by xoring the first response with the third one, as expressed in Equation 6.1 and 6.2. Finally, both Things send to each other the values $\alpha_{ij}^{i}$ and $\alpha_{ij}^{j}$ so that each one of them constructs the extended CRP as follows:

$$\Gamma_{ij}(T_{ji}) = (x_{ij}^1, x_{ij}^2, x_{ij}^3, \alpha_{ij}^{i}, \alpha_{ij}^{j}) \quad (6.3)$$

In the following paragraphs, we first start by presenting the authentication phase of the protocol in the first configuration (i.e., T2T with gateway) then we present the second configuration (i.e., T2T without gateway).

**Authentication Phase (Configuration 1)**

In this first configuration, resource-constrained Things authenticate each other through a central gateway. For example, this configuration can be used when Things cannot store eCRPs about other Things due to their limitation in storage capacity. They only store the eCRP of the gateway for authentication purposes. This phase is illustrated by the MSC\(^1\) of Fig. 6.2, where we have set $i = 1$ and $j = 2$ for simplicity. The gateway is generally assumed to have more resources than Things. Although the gateway is also a Thing in the context of IoT, we rather prefer to explicitly call it a gateway instead of a Thing to make a difference in the resource capabilities. The steps of the authentication are as follows:

**Step 1.** T2T-MAP assumes that a Thing $T_{ij}$ starts the protocol by sending a direct request

---

\(^1\)MSC (Message Sequence Chart) is a graphical language for the description of the interaction between different components of a system. This language is standardized by the ITU (International Telecommunication Union).
6.2. DESIGN OF THE AUTHENTICATION PROTOCOL

Figure 6.2: The developed PUF-based T2T authentication protocol. This first part of the protocol allows only a mutual authentication of two Things to a gateway. The notation $M_{x \rightarrow y}$ denotes a message $M$ sent from $x$ to $y$ (where $x, y \in \{0, 1, 2\}$, 0 refers to the gateway $G$, 1 refers to Thing $T_1$, and 2 refers to Thing $T_2$). The PUF function of $G$, $T_1$, and $T_2$, are $\Psi_0(\cdot)$, $\Psi_1(\cdot)$, and $\Psi_2(\cdot)$, respectively. Also, $N_0$, $N_1$, and $N_2$ are three nonces generated by $G$, $T_1$, and $T_2$, receptively, $\oplus$ is the bitwise eXclusive OR operator, $N_{ij}$ is the XOR of the two nonces $N_i$ and $N_j$ ($N_{ij} = N_i \oplus N_j$), and $\mathcal{H}(\cdot)$ is the hash function. The variables $x^j_1$, $x^j_2$, and $x^j_3$, where $i \in \{0, 1, 2\}$, are challenges generated by a Thing $T_i$, whereas $\alpha^j_i$ and $\beta^j_i$, where $i \in \{1, 2, 3\}$, $j \in \{0, 1, 2\}$, and $k \in \{0, 1, 2\}$, are values that are computed using the PUF of two parties $j$ and $k$ over specific challenges. Finally, the notation $\mathcal{H}(\cdot)$ refers to the hash of an entire message (i.e., its message integrity code). This values is verified upon the reception of each message $m_i$. 

\begin{align*}
\mathcal{T}_1 & = (x^j_1, x^j_2, x^j_3, \alpha^j_i) \quad (j \in \{1, 2, 3\}) \\
\mathcal{T}_2 & = (x^j_1, x^j_2, x^j_3, \alpha^j_i) \quad (j \in \{1, 2, 3\})
\end{align*}

Verify: $\mathcal{H}(\mathcal{T}_1^j, N_{ij}, N_{ij}) = \mathcal{H}(\mathcal{T}_2^j, N_{ij}, N_{ij})$.
for authentication to another Thing $T_{ji}$. This step is not illustrated in Fig. 6.2. Then, Things $T_i$ and $T_j$ use the eCRP that is related to the gateway to retrieve the challenges $x_1^{ij}, x_2^{ij},$ and $x_3^{ij}$ and apply their local PUF to produce the responses $\Psi_{ij}(x_1^{ij}), \Psi_{ij}(x_2^{ij}),$ and $\Psi_{ij}(x_3^{ij})$, which are then sent to the gateway as challenges along with a nonce $N_{ij}$.

**Step 2.** The gateway $G$ receives the challenges from both Things and applies its local PUF to compute the values of $\beta_1^{0|j0}$ and $\beta_2^{0|j0}$ for each Thing, as follows:

$$\beta_1^{0|j0} = \Psi_0(\Psi_{ij}(x_1^{ij})) \oplus \Psi_0(\Psi_{ij}(x_2^{ij}))$$  \hspace{1cm} (6.4)$$

and

$$\beta_2^{0|j0} = \Psi_0(\Psi_{ij}(x_1^{ij})) \oplus \Psi_0(\Psi_{ij}(x_3^{ij}))$$  \hspace{1cm} (6.5)$$

The gateway generates a nonce $N_0$ and then takes the first value $\beta_1^{0|j0}$ and hashes it along with the XOR of its nonce $N_0$ and the corresponding Thing’s nonce $N_{ij}$, i.e., computes $H(\beta_1^{0|j0}, N_{ij} \oplus N_0)$. This hash is again hashed along with the nonces to compute the obfuscated response value $H(H(\beta_1^{0|j0}, N_{ij} \oplus N_0), N_{ij}, N_0)$. This value is the response of the gateway to the challenges.

**Step 3.** The gateway uses the extended CRP that is related to each Thing to retrieve the challenges $x_1^0, x_2^0,$ and $x_3^0$ and applies the local PUF function to produce the responses $\Psi_0(x_1^0), \Psi_0(x_2^0),$ and $\Psi_0(x_3^0)$, which are the challenges to be sent to Things. The gateway then constructs and sends two messages $m_1$ and $m_1'$ that contain the identities of Things along with its identity, the nonces, the three challenges, the computed obfuscated response, and a hash value, denoted by $H(\ast)$, that is computed over the entire message for message integrity protection.

**Step 4.** Things receive the gateway’s messages and start by verifying its integrity, i.e.,
6.2. DESIGN OF THE AUTHENTICATION PROTOCOL

Then, the extended CRP is used to retrieve the value of $\alpha_{1|i,j}^{0|j_0}$ which is hashed along with the XOR of the nonces and then hashed again with the nonce of the concerned Thing $T_{i,j}$ and the nonce of the gateway $N_0$. For authentication, the result of hashing $\alpha_{1|i,j}^{0|j_0}$, i.e., $\mathcal{H}(\alpha_{1|i,j}^{0|j_0}, N_{ij} \oplus N_0, N_{ij}, N_0)$, is compared to the received value $\mathcal{H}(\mathcal{H}(\beta_{1|i,j}^{0|j_0}, N_{0|i,j} \oplus N_0, N_{ij}, N_0))$. If both values are equal, the gateway is authenticated by both Things.

**Step 5.** Both Things then use the received challenges to apply their local PUF function and compute the corresponding responses, $\Psi_{i,j}(\Psi_0(x_1^0))$, $\Psi_{i,j}(\Psi_0(x_0^0))$, and $\Psi_{i,j}(\Psi_0(x_0^0))$. Using the computed values, both Things compute the values of $\beta_{1|i,j}^{0|j_0}$ and $\beta_{2|i,j}^{0|j_0}$, as follows:

$$\beta_{1|i,j}^{0|j_0} = \Psi_{i,j}(\Psi_0(x_1^0)) \oplus \Psi_{i,j}(\Psi_0(x_0^0)) \quad (6.6)$$

and

$$\beta_{2|i,j}^{0|j_0} = \Psi_{i,j}(\Psi_0(x_1^0)) \oplus \Psi_{i,j}(\Psi_0(x_0^0)) \quad (6.7)$$

Once computed, the value of $\beta_{1|i,j}^{0|j_0}$ is hashed along with the XOR of the nonces and then hashed again with the nonces, i.e., $\mathcal{H}(\mathcal{H}(\beta_{1|i,j}^{0|j_0}, N_{ij} \oplus N_0, N_{ij}, N_0))$. A message $m_2$ (and $m_2'$) is constructed using the identities of the authenticating parties, the nonces, the hashed response, and the hash of the entire message $\mathcal{H}(\ast)$ for integrity protection. The message is then sent to the gateway.

**Step 6.** The gateway receives the messages $m_2$ and $m_2'$ from Thing $T_i$ and $T_j$, respectively. It uses the stored value $\alpha_{1|i,j}^{0|j_0}$ to compute the value of $\mathcal{H}(\mathcal{H}(\alpha_{1|i,j}^{0|j_0}, N_{ij} \oplus N_0, N_{ij}, N_0))$ and compares it to the received response $\mathcal{H}(\mathcal{H}(\beta_{1|i,j}^{0|j_0}, N_{0|i,j} \oplus N_0, N_{ij}, N_0))$. If they are equal for both Things, then both Things are authenticated.

**Step 7.** The gateway generates a secret random nonce $L_0$ and computes two additional

\footnote{For the sake of efficiency, we have decided to start the verification of the integrity of the messages before their authenticity. Our decision is based on the principle of fast decision with less computation. In fact, integrity verification consists of 1 hash operation and 1 comparison, while the authenticity consists of 2 hash operations and 1 comparison. In the worst case, the order does not matter as both orderings end up performing 3 hash operations and 2 comparisons.}
values $\alpha^0_3$ and $\alpha^0_j$ using the following equation:

$$\alpha^0_3 = \alpha^0_1 \oplus \alpha^0_2$$ (6.8)

Also, the gateway computes the values $L_0 \oplus H(\alpha^0_3)$ and $H(\alpha^0_2) \oplus \alpha^0_i$, and then sends two messages $m_3$ and $m'_3$ to Thing $T_i$ and $T_j$, respectively. These messages contain the identity of the authenticating parties, the nonces, the value $L_0 \oplus H(\alpha^0_3)$, the value $H(\alpha^0_2) \oplus \alpha^0_i$, and the hash of the entire message.

**Step 8.** Things receive the gateway’s messages and start by verifying its integrity, i.e., $H(*)$. The extended CRP is used to retrieve the value of $\alpha^0_3$, which is then hashed along with the XOR of the nonces ($N_{0ij0} = N_{ij} \oplus N_0$) and then hashed again with the nonce so that it can be compared with the received value $H(H(\beta^0_3, N_{ij} \oplus N_0, N_{ij}, N_0))$ to authenticate the source of the message. Once authenticated, Things compute the value of $\beta^0_3$ as follows:

$$\beta^0_3 = \beta^0_1 \oplus \beta^0_2$$ (6.9)

Using the previously computed value $\beta^0_2$ in **Step 5** (Equation 6.7) and the value of $\beta^0_3$ (Equation 6.9), Things compute the value of $\alpha^0_2$ = $H(\alpha^0_3 \oplus \alpha^0_2) \oplus H(\beta^0_2)$ and the value of $L_0$ = $(L_0 \oplus H(\alpha^0_3)) \oplus H(\beta^0_3)$. At this stage, both Things have a shared secret nonce $L_0$ as well as a part of the extended CRP of the other Thing. These two information will be used to establish a T2T authentication between $T_i$ and $T_j$.

**Step 9.** In this step, both Things start authenticating each other. To that end, Thing $T_{ij}$ that first initiated the protocol generates a key $K$ then xores the key with the hash of the xor of the obfuscated response $\alpha^0_{0ij}$ of Thing $T_{ji}$ and the secret nonce $L_0$, to compute the value $K \oplus H(\alpha^0_{0ij} \oplus L_0)$. This value allows to securely send and share the key with the other Thing. Then, it uses the nonces and the secret nonce $L_0$ to compute the value of $H(\alpha^0_{0ij}, N_{ij}, N_i, N_j, L_0)$. This value allows authenticating the source of the message to
be sent as well as to link this second part of the authentication to the first part that was performed with the gateway (through the use of the secret nonce $L_0$). These values are sent to the other party along with a message integrity code, i.e., $H(*)$.

**Step 10.** Upon the reception of the message from $T_{ij}$, Thing $T_{ji}$ checks the integrity of the message, i.e., $H(*)$, and then uses the previously computed value $\beta_2^{0j|0i}$ in Step 5 (Equation 6.7) to check the authenticity of the source of the message as well as its linkability to the previous authentication part with the gateway. This consists of comparing the received value $H(H(\alpha_2^{0j|0i}, N_{ij}), N_i, N_j, L_0)$ with $H(H(\beta_2^{0j|0i}, N_{ij}), N_i, N_j, L_0)$. If the values are equal, then $T_{ji}$ authenticates $T_{ij}$. In this case, $T_{ji}$ extracts the key $K$ by computing $K = (K \oplus H(\alpha_2^{0j|0i} \oplus L_0)) \oplus H(\beta_2^{0j|0i} \oplus L_0)$. Once the key is computed, Thing $T_{ji}$ hashes the obfuscated response $\alpha_2^{0i|0j}$ of Thing $T_{ij}$ along with the key $K$ to prove the correct key computation to Thing $T_{ij}$. Also, it computes the value $H(H(\alpha_2^{0i|0j}, N_{ij}), N_i, N_j, L_0)$ for message source authenticity and sends these values along with the message integrity code.

**Step 11.** Thing $T_{ij}$ receives the message of Thing $T_{ji}$ (i.e., $m_5$) and starts by checking its integrity code, i.e., $H(*)$. Then it uses the previously computed value $\beta_2^{0i|0j}$ in Step 5 (Equation 6.7) to check the authenticity of the source of the message as well as its linkability to the previous authentication part with the gateway by comparing the received value $H(H(\alpha_2^{0i|0j}, N_{ij}), N_i, N_j, L_0)$ with $H(H(\beta_2^{0i|0j}, N_{ij}), N_i, N_j, L_0)$. If the values are equal, then $T_{ij}$ authenticates $T_{ji}$. Also, Thing $T_{ij}$ checks that the received value $H(K \oplus \alpha_2^{0i|0j} \oplus L_0)$ is equal to $H(K \oplus \beta_2^{0i|0j} \oplus L_0)$ to confirm that Thing $T_{ji}$ has correctly computed the key.

This second part of the protocol is illustrated by the MSC$^1$ of Fig. 6.3, where we have set $i=1$ and $j=2$ for the sake of simplicity.

**Authentication Phase (Configuration 2)**

In this configuration, resource-constrained Things authenticate each other without the help of any central gateway. For example, this configuration can be used when Things can store
6.2. DESIGN OF THE AUTHENTICATION PROTOCOL

Generate a key: $K$.

Verify:
$$H(H(α_{02}^2, N_{12}), N_1, N_2, L_0) = H(H(β_{02}^2, N_{12}), N_1, N_2, L_0).$$

T1 Authenticated

Compute:
$$K = (K ⊕ H(α_{02}^2 ⊕ L_0)) ⊕ H(β_{02}^2 ⊕ L_0).$$

m4

T1

m5

T2

Generate a key: $K$.

Verify:
$$H(H(α_{01}^2, N_{12}), N_1, N_2, L_0) = H(H(β_{01}^2, N_{12}), N_1, N_2, L_0).$$

T2 Authenticated

Verify: $H(K ⊕ α_{01}^2 ⊕ L_0) = H(K ⊕ β_{01}^2 ⊕ L_0)$.

Key correctly derived by T2

$K$

$K$

$m_4 = \{id_1, id_2, N_1, N_2, K ⊕ H(α_{02}^2 ⊕ L_0), H(H(α_{02}^2, N_{12}), N_1, N_2, L_0), H(+)\}_{1→2}$,

$m_5 = \{id_2, id_1, N_1, N_2, H(K ⊕ α_{01}^2 ⊕ L_0), H(H(α_{01}^2, N_{12}), N_1, N_2, L_0), H(+)\}_{2→1}$.

Figure 6.3: Second stage of T2T authentication through gateway. In this second part of the authentication, both Things, $T_1$ and $T_2$, authenticate each other and agree on a symmetric cryptographic key $K$ to be used to secure future communications.

eCRPs about other Things or when Things are deployed in hostile environment where it is difficult to deploy a gateway. The steps of the authentication are pretty similar to the ones of Configuration 1. In fact, similar to Configuration 1, each Things $T_{ij}$ uses the first obfuscated response $α_{ij}^{ij'}$ of Thing $T_{ji}$ to verify its authenticity. This would consist of having each Thing $T_{ij}$ send three challenge values, $Ψ_{ij}(x_{i1}^{ij})$, $Ψ_{ij}(x_{i2}^{ij})$ and $Ψ_{ij}(x_{i3}^{ij})$, to
6.3. T2T-MAP Properties and Security Analysis

In this section, we start by discussing some security as well as performance properties of T2T-MAP. Then, we perform a security analysis on the protocol by informally discussing the resilience of T2T-MAP against some known attacks. We formally prove its security using Tamarin security protocol prover [MSC+13].

6.3.1 Protocol Properties

T2T-MAP provides a set of security and performance properties that conform to IoT security and performance requirements. In the following paragraphs, we present these properties:
Figure 6.4: The developed PUF-based T2T authentication protocol without central gateway. This protocol allows a mutual authentication of two Things $T_1$ and $T_2$. The notation $M_{x\rightarrow y}$ denotes a message $M$ sent from $x$ to $y$ (where $x, y \in \{0, 1, 2\}$, 0 refers to the gateway $G$, 1 refers to Thing $T_1$, and 2 refers to Thing $T_2$). The PUF function of $G$, $T_1$, and $T_2$, are $\Psi_0(\cdot)$, $\Psi_1(\cdot)$, and $\Psi_2(\cdot)$, respectively. Also, $N_1$ and $N_2$, are two nonces generated by $T_1$ and $T_2$, receptively, $\oplus$ is the bitwise eXclusive OR operator, $N_{ij}$ is the XOR of the two nonces $N_i$ and $N_j$ ($N_{ij} = N_i \oplus N_j$), and $H(\cdot)$ is the hash function. The variables $x_1^i$, $x_2^i$, and $x_3^j$, where $i \in \{1, 2\}$, are challenges generated by a Thing $T_i$, whereas $\alpha_{1k}^i$ and $\beta_1^{jk}$, where $i, j, k \in \{1, 2\}$, are values that are computed using the PUF of two parties $j$ and $k$ over specific challenges. Finally, the notation $H(\ast)$ refers to the hash of an entire message (i.e., its message integrity code, or MAC). This values is verified upon the reception of each message $m_i$. 

\[
\begin{align*}
\Gamma_1(T_2) &= (x_1^1, x_2^1, x_3^1, \alpha_1^2), \\
\alpha_1^2 &= \Psi_2(\Psi_1(x_1^1)) \oplus \Psi_2(\Psi_1(x_2^1)), \\
\alpha_1^2 &= \Psi_1(x_1^1) \oplus \Psi_2(\Psi_1(x_2^1)) \\
\end{align*}
\]
Lightweight. T2T-MAP is considered as a lightweight, if not an ultra-lightweight, authentication protocol. This is due to the following reasons: The protocol is based on the use of PUFs (Physical Unclonable Functions) to prove device identities. Also, it uses simple hash functions as well as the bitwise logical exclusive OR operator (XOR) to establish authentication and secret key establishment. In case encryption is used, the authentication protocol can adopt a lightweight encryption algorithm such as ChaCha [IET16].

Mutual Authentication. Our protocol provides mutual authentication. In fact, to establish authentication between three (Configuration 1) or two (Configuration 2) entities, each entity is required to use its own PUF function to generate authentication responses that will prove its identity.

Scalability. The protocol can be qualified as scalable for the following reasons:

- In terms of storage overhead, each individual Thing $T_i$ has to store $2n + 3$ values, where $n > 0$ is the number of Things $T_j \neq i$ registered on Thing $T_i$’s memory. For example, in Configuration 1, each Thing $T_i$ stores only 5 values ($n = 1$). In fact, each Thing will only store the eCRP of the gateway (the challenges $x_1$, $x_2$, and $x_3$, are the same for all Things). Hence, if the variables are expressed on 32 bytes (in the most secure case), each Thing will permanently have to store only 160 bytes, which is a reasonable storage overhead. This overhead grows linearly following the function $f(n) = 2n + 3$, where $n > 0$ is the number of registered eCRPs on a Thing.

- The response time of the authentication protocol will remain constant while adding new Things to the infrastructure only if the gateway is decentralized. In this thesis, we have adopted a “centralized” infrastructure that has only one gateway. This infrastructure is open for decentralization. Thus, a load-balancing could be established to maintain a constant response time. Nevertheless, in both configurations (Configuration 1 and 2), the response time of a given Thing $T_i$ to a group of Things $T_j$ would depend on many factors: (1) Thing’s
6.3. T2T-MAP PROPERTIES AND SECURITY ANALYSIS

Protocol code must be a multi-threaded process. (2) Thing’s CPU must be a multi-core processor. (3) The speed to access to wireless medium is managed and limited by the MAC-layer media access control protocols, e.g., CSMA/CA (Carrier Sense Multiple Access - Collision Avoidance).

**Smart.** In terms of specification, the protocol specifies an optional security functionally to allow authenticating parties to activate a race-condition attack prevention option. This function allows each Thing $T_i$, on reception mode, to collect response messages for a short and arbitrary time frame before processing the messages. Therefore, in the ordinary case where there is no attacker, if a Thing $T_i$ is expecting a message $m$ from Thing $T_j$, then Thing $T_i$ will only receive that message $m$. However, in the case where an attacker plans a race-condition attack [LZ19a; LZ19c; LZ20a], Thing $T_i$ may receive multiple “different” instances of the message $m$. By treating these messages, Thing $T_i$ will be able to take a smart decision to whether it should move forward on the authentication protocol or abort the communication. A Thing $T_i$ moves forward as soon as it treats a valid and legitimate message $m$. In this thesis (version of the protocol), we have not implemented this function.

**Key Establishment.** T2T-MAP allows the establishment of a secret key at the end of the authentication. This key can be used for encryption (e.g., using ChaCha [IET16]) and data integrity. The key is randomly generated during the authentication and immediately destroyed at the end of the session.

**Autonomous.** The developed protocol is completely autonomous and does not require the involvement of any human-being during an authentication session. Nevertheless, the enrolment phase would eventually require the involvement of a user for an initial set-up. A typical enrolment scenario could be a user (e.g., citizen or resident) may register its vehicle’s eCRPs at a provincial service (e.g., Service Ontario) to download the eCRPs of the road signs, toll-collection barriers, and law-enforcement/emergency vehicles that are operational within the province so that its vehicle authenticates these facilities.
6.3. T2T-MAP PROPERTIES AND SECURITY ANALYSIS

**Freshness and Liveness.** The protocol uses nonces for each authentication session, which makes, to some extent, each authentication session fresh and unique. This would prevent any replay attack using messages from a previous authentication session. Also, in Configuration 1, the protocol uses the nonce $L_0$ to allow Things $T_1$ and $T_2$ verify that the current communication is part of the previous communication with the gateway. This would prevent any session hijacking attempts.

**Availability.** For the authentication protocol to be available at all times, each authenticating party has to have its authentication information, i.e., eCRPs, up-to-date. In the current version of the protocol, the stored eCRPs are static and not updated, which would make the protocol always available (no desynchronization). Nevertheless, following the security principles, these eCRPs need to be updated at some point. Therefore, we assume the existence of a secure and frequent eCRP-updating procedure to allow Things to store and use new eCRPs. This would consequently prevent the possibility of forward security-related attacks as well as brute force attacks as discussed in the next paragraph.

**Forward Security.** This property is strongly related to the lifetime of the used authentication information (e.g., the lifetime of an eCRP). If the lifetime is too long, an attacker would have enough time to brute force and disclose authentication information to perform impersonation. For example, through a session hijacking attack (cf., next subsection), an attacker would be able to learn 50% of the eCRP of a given Thing $T_i$ (i.e., $\alpha_{0i}^1$). Then, through brute forcing and one single XOR-operation, the attacker will be able to learn the remaining part of the eCRP (i.e., $\alpha_{1i}^0$ and $\alpha_{3i}^0$). Using this information, the attacker can completely impersonate Thing $T_i$ during a future authentication session. Therefore, to mitigate this attack and provide forward security, we assume the existence of a secure eCRP-updating procedure that will allow Things to frequently and securely update the stored eCRPs.
6.3. T2T-MAP Properties and Security Analysis

Non-repudiation. As PUFs are widely assumed to be unclonable functions and the authentication in T2T-MAP relies on the use of PUFs, we can claim that the security service of non-repudiation is guaranteed and enforced in T2T-MAP. Thus, no authenticating party can deny having participated in an authentication session.

6.3.2 Security Analysis

In what follows, the security strength of T2T-MAP against known attacks is discussed:

Machine Learning Attacks. These attacks are prevented by not exchanging the responses in plaintext during an authentication. T2T-MAP adopts a combination of two challenges, e.g., \( \beta_{1ji} = \Psi_{ij}(x_{1ji}) \oplus \Psi_{ij}(x_{2ji}) \), and sends the response double hashed along with nonces, e.g., \( \mathcal{H}(\mathcal{H}(\beta_{1ji}, N_{ij}), N_i, N_j) \). This obfuscation technique prevents eavesdroppers from collecting challenges and their corresponding responses during authentication to build a prediction model. Also, the base value of the challenges is not sent during authentication. In fact, what is being sent is the response from the local PUF for a challenge that is never revealed. For example, when Thing \( T_1 \) tries to authenticate the gateway, it sends the value \( \Psi_1(x_{1}) \) and keeps \( x_{1} \) confidential.

Replay Attack. A replay attack is prevented through the use of nonces. For example, throughout the authentication, three nonces are used: \( N_0 \) (generated by the gateway), \( N_1 \) (generated by Thing \( T_1 \)), and \( N_2 \) (generated by Thing \( T_2 \)). The obfuscated responses (i.e., \( \beta_{1ji}, \beta_{2ji}, \beta_{3ji} \)) are never sent in plaintext. They are first hashed along with the XOR of the nonces \( (N_{ij} = N_{ji} = N_i \oplus N_j) \) and then hashed again with the nonce, e.g., \( \mathcal{H}(\mathcal{H}(\beta_{1ji}, N_{ij}), N_i, N_j) \). This makes the responses linked to the current session and cannot be used in another session. An attacker may think of intercepting some messages and using the same nonce \( N_{ij} \) to replay the messages and spoof Thing \( T_{ij} \). This would be difficult, or impossible, for the attacker as the latter cannot force the other party, \( T_{ji} \), to use the same nonce \( N_{ji} \), used in a previous authentication session.
Compromising Gateway/Thing. If an attacker manages to gain access to the gateway or a Thing, it will not be able to infer and learn the eCRP of any other Thing. In fact, the eCRP, $\Gamma_{ij}(T_{ji})$, which is stored on Thing $T_{ij}$ about Thing $T_{ji}$, is an information that is used by Thing $T_{ij}$ to only verify the authenticity of Thing $T_{ji}$. It cannot be used in anyways by Thing $T_{ij}$ to prove that it is Thing $T_{ji}$. Also, an eCRP, $\Gamma_{ij}(T_{ij})$, is tightly-coupled to the two Things $T_{ij}$ and $T_{ji}$, and cannot be used by another Thing $T_k$ ($k \neq i \neq j$).

Session Hijacking. In this attack scenario, a malicious insider Thing, let us say $T_k$, performs a legitimate authentication with Thing $T_i$ through the gateway and obtains the value $\alpha_2^{0i}$. As previously discussed, this value is used by Thing $T_i$ to prove its identity to another Thing $T_j$. Then, after some time, the malicious Thing $T_k$ eavesdrops another authentication between the gateway and Things $T_i$ and $T_j$. When the authentication with the gateway is achieved and both Things $T_i$ and $T_j$ have received (from the gateway) the information $\alpha_2^{0i0j}$ about the counterpart $T_{ji}$, the malicious Thing $T_k$ spoofs Thing $T_i$ and blocks its messages so that they do not reach Thing $T_j$. At this point, the malicious Thing $T_k$, can prove to $T_j$ that it is $T_i$ as it has the value $\alpha_2^{0i}$ that it has learned from a previous legitimate authentication session. Therefore, to prevent such an attack, we have used the nonce $L_0$ of the gateway as an indicator of the current session. The value of $L_0$ is secretly shared among the legitimate parties. It will be used by the communicating Things, in this case $T_i$ and $T_j$, to verify whether the received messages are somehow related to the authentication session that occurred with the gateway (liveness). Therefore, without the knowledge of $L_0$, the malicious Thing $T_k$ will not be able to hijack the session.

Insider Spoofing. A malicious Thing, say $T_k$, could operate a legitimate authentication with another Thing, say $T_i$, and obtain the response information $\alpha_2^{0i}$ to spoof $T_i$ later on. In this situation, the malicious Thing will not be able to spoof $T_i$ as it has only one obfuscated response out of two obfuscated responses to prove that it is $T_i$. In addition, the initial part of the authentication with the gateway requires the attacker the knowledge of the
6.3. T2T-MAP PROPERTIES AND SECURITY ANALYSIS

information $\alpha_{0i}$, which $T_k$ cannot obtain. Also, the value $\alpha_{1i}$ is never sent in plaintext but always double hashed along with nonces.

**Brute Forcing PUF-Responses.** As the nonces, $N_i$ and $N_j$, are sent in plaintext during the authentication, an attacker may try to brute force the hash function $\mathcal{H}(\cdot)$ to find the obfuscated response value that was generated by an authenticating party. This would depend on the size of the response. If the response is on $n$ bits, the attacker has to try at most $2^n$ values before finding the response. To make this attack difficult and infeasible, instead of using the obfuscated response as the first parameter for the hash function, i.e., $\mathcal{H}(\beta_{vij_i}|N_i, N_j)$, where $v \in \{1, 2\}$, we send the hash along with nonces of the hash of the obfuscated response along with the XOR of nonces, i.e., $\mathcal{H}(\mathcal{H}(\beta_{vij_i}, N_{ij}), N_i, N_j)$, where $N_{ij} = N_i \oplus N_j$. If the attacker brute forces the transferred hash, it will find the value of $\mathcal{H}(\beta_{vij_i}, N_{ij})$, which needs to be brute-forced again to find the value of $\beta_{vij_i}$. In this case, the attacker will face a complexity of $O(2^{n+2})$ to be able to disclose the eCRP of a particular Thing $T_i$. Also, if the attacker (compromised Thing $T_j$) applies the session hijacking approach to learn the value of $\alpha_{0i}^2$ of Thing $T_i \neq j$, the attacker will have to brute force a complexity of $O(2^{n+1})$. Therefore, we recommend at least 64-bit for the response values so that the brute force attack on the hash function would be very difficult. This would eventually prevent the replay attack.

**Secret Keys Disclosure Attack.** During the authentication, the protocol allows the communicating parties to establish a secret key to be used for encryption and/or data authenticity. This key is immediately destroyed at the end of the communication. Therefore, the key is generated only when needed and destroyed once the session is terminated, which means that no secret keys are present permanently on the devices. This allows the devices to be resistant to any physical key disclosure attacks.
6.3. T2T-MAP PROPERTIES AND SECURITY ANALYSIS

Security Verification

In addition to the above analysis, we have used TAMARIN [MSC+13], a well-established software tool for security protocol verification and theorem proving, to formally prove the security of T2T-MAP. To that end, we have implemented T2T-MAP in TAMARIN’s protocol specification language. This basically consists of writing rules of facts and lemmas. In fact, TAMARIN models protocol’s set of executions as a labeled transition system (LTS). The states of the LTS are multisets of facts formalizing the local states of the authenticating party running the protocol, the attacker’s knowledge, and the exchanged messages. The transitions of the LTS are formalized using rules. Lemmas however, are used to express security properties, e.g., secrecy, to be verified by the tool in order to claim the security property for the protocol. The specification code of the protocol (a.k.a., the security protocol theory in TAMARIN) consists of 216 lines of code for Configuration 1 and 135 lines of code for Configuration 2. They can be accessed online at [LZ20f].

Next, based on the design of the protocol as well as on the above security analysis, we can observe that the security of T2T-MAP strongly relies on the secrecy of the following variables: (1) The challenges $x_j^i$, where $i, j \in \mathbb{N}$. (2) The obfuscated responses, $\alpha_i^{jk}$ and $\beta_i^{jk}$, where $i, j, k \in \mathbb{N}$. (3) The shared nonce $L_0$ (generated by the gateway). (4) The shared key $K$. Therefore, we can claim that if these variables are kept secrete and not disclosed to the attacker in anyways, then the protocol cannot be compromised. To verify this claim using TAMARIN prover, we have written lemmas to express the secrecy of the aforementioned variables and prove whether the lemmas are always true. After executing the verification of the lemmas on TAMARIN (which took around +/-3 seconds), the tool proved the lemmas to be always true (viz., Figure 6.5). This means that there exist no possible execution for the protocol that leads to the disclosure of any of those variables.
6.4 Protocol Implementation and Evaluation

In this section, we present the implementation detail of T2T-MAP for both configurations and then evaluate the performance for different protocol configurations.

6.4.1 Protocol Implementation

To implement T2T-MAP, we have used ARDUINO development boards as low-cost and resource-constrained devices. We have specifically used two types of ARDUINO boards, namely, the ARDUINO Mega 2560 (R3) and the ARDUINO DUE. These two boards embed
different microcontrollers of different processing and storage capacity as illustrated in Table 6.2. Also, we have used different configurations for the protocol with respect to the size of the challenges, the responses, the nonces, and the hash outputs. In this subsection, we present how we have implemented the PUF, the extended CRPs, the communication, and the protocol code.

**PUF Implementation.** As there exist many types of PUFs, in this thesis, we do not apply a particular PUF function neither we implement a new one. We rather consider the use of an abstract PUF and keep the protocol as generic. We represent this abstract PUF, that could be implemented using any type of PUF, by a simple hash function. Concretely, for the sake of this implementation, we have adopted the BLAKE\(^3\) hash function that is available in the Arduino Cryptographic Library\(^4\) to implement the abstract PUF as well as the hash function \(H(\cdot)\) that is used in the protocol. We emphasize that the abstraction is only to implement a generic prototype of the protocol so that a performance analysis can be performed. Researchers can use any type of PUF to build their own version of the protocol.

Nevertheless, to use a hash function instead of a PUF to implement the protocol and perform a performance evaluation (in particular, in terms of execution time and energy

---

3BLAKE is a cryptographic hash function based on ChaCha stream cipher.
consumption), we need to demonstrate that the execution time as well as the energy consumption of a real PUF function are bound by the execution time and energy consumption of the adopted hash function (i.e., BLAKE), respectively. To that end, we analyze the execution time and energy consumption of PUF functions that are developed and evaluated in the literature. However, it is important to note that most, if not all, PUF’s evaluation times that are reported in the literature are not raw measures that express the PUF’s output generation time. In fact, the PUF’s evaluation time usually includes the time of additional operations, such as the use of a fuzzy extractor, an encoder, a decoder, an error-corrector, and the application of a hash function to output a desired number of bits. These additional operations are required to guarantee a high PUF reliability, uniqueness, and uniformity.

Khoshroo et al., [Kho13] reported that a ring oscillator-based PUF, implemented on a Virtex II Pro FPGA board, would take around 1200µs to generate a one-bit output. Hence, if we assume a linear relationship between the PUF’s output size and its evaluation time, a ring oscillator-based PUF would take around 76.80ms, 153.60ms, and 307.20ms, to generate an 8-byte, 16-byte, and 32-byte output, respectively. Maes et al., [Mae12] showed that a ring oscillator-based PUF, implemented on a Xilinx Spartan-6 FPGA, requires 5.62ms to produce a 16-byte output. Nevertheless, the ring oscillator-PUF is known to be a slow PUF due to its runtime, which is the delay that is related to the counters that are used in this type of PUF to produce the output. Wang et al., [WXO20] implemented a Lattice-based PUF on a Xilinx Spartan-6 FPGA and showed that a 100-bit output can be generated in 4.40ms. Also, Jia et al., [JXW+15] demonstrated that a NAND-flash-memory PUF implemented on an 8-bit micro-controller requires 93.83ms to produce a 16-byte output.

These results show that PUF’s evaluation time depends on the type of the PUF, the size of the output, and the implementation platform. Although these values seem to be

---

5Ring oscillator-PUFs use multiple ring oscillators that produce a clock signal. Depending on the challenge value, a couple of ring oscillators are selected and the frequency of their produced clock signal are evaluated by components called counters. These counters induce a delay while evaluating the frequencies. By comparing the frequencies, an output bit is calculated.
high compared to the execution time of the adopted hash function on the Arduino boards, other researchers have reported much lower-values. For example, Feng et al., [FYZ+18] reported that their SRAM-based PUF generates a 1314-bit output in 1.471ms on a MSP340-microcontroller (at 1MHz). Also, Khoshroo et al., [Kho13] demonstrated that the Anderson PUF (a Glitch-PUF) [And10b] and the SR-Latch PUF [LHK+10], implemented on the Virtex II Pro FPGA board, takes around 0.08µs and 0.04µs to generate a one-bit output, respectively. These results can be extrapolated to claim that it is possible to generate a 32-byte output in less than 21µs using an Anderson-based PUF and less than 11µs using an SR Latch-based PUF. Thus, we can assume the existence and the use of a PUF function that can be evaluated in few microseconds. We can generalize the final execution time $t$ of the protocol for a given node, e.g., Thing $T_i$, as:

$$t = t' + n \cdot \tau$$

Where, $t'$ is the execution time of the protocol without the use of a real PUF, $n$ is the number of PUF operations in the protocol (in Thing $T_i$), and $\tau$ the average PUF’s evaluation time. In our case, we consider that $t \simeq t'$, i.e., $t' \ll n \cdot \tau$.

Furthermore, some researchers have demonstrated that PUFs consume a very negligible amount of energy during their evaluation. For example, Lin et al., [LHK+10] reported that their proposed PUF (SR Latch PUF) consumes 0.047pJ per cycle. Also, Lim et al., [LLG+05] reported that their PUF (Ring oscillator-PUF) has an energy consumption of 0.136pJ per cycle.

Based on the above reported values, it is possible to assume that the PUF evaluation time as well as its energy consumption can be neglected compared to the hash function evaluation time and energy consumption. In any case, a hash function will be used to provide a desired output size (e.g., 8, 16, and 32 bytes) and allow the use of a PUF function that requires a small area size for its implementation and produce outputs of smaller sizes.
Finally, to advocate why we have not chosen to adopt a particular PUF function over others for our protocol, we provide the following justifications: (1) Some PUF functions are sometimes unreliable, which means, the output for a given challenge is more often not the same (e.g., metastable states in Arbiter-PUFs) under different temperatures. (2) Certain PUF functions have a very low level of uniqueness (e.g., 1.05% in [GLC+04]). (3) Many of the existing PUF functions have been shown to be vulnerable to modelling attacks [ZWW+18; Del19; RSS+13; TG15; GSS15; RS14; YN17; AZ17; MRM+13]. (4) Some PUF functions require a large circuit area to be implemented. They cannot be used on tiny resource-constrained devices. (5) A large number of the proposed PUFs that aim to increase reliability, uniqueness, randomness, and resistance to modeling attacks of previous implementations, tend to have a more complex design and a longer evaluation time. (6) Most of the PUF’s implementations and evaluations are performed on FPGA development boards. This allows to compare the performance of different PUFs when implemented on the same development board. Nevertheless, the performance results cannot be guaranteed on a different board and cannot be used as part of the performance of any other protocol.

**eCRPs Implementation.** We have used random nonces as challenges and the hash of the nonces as their corresponding responses. Also, we have used different sizes, namely, 8 bytes, 16 bytes, and 32 bytes, for the challenges and responses so that we can study the performance of the protocol when the sizes are changed (e.g., when upgrading the security level). The Arduino Cryptographic Library provides a random number generator to generate nonces of arbitrary sizes. Thus, for an $n$-bit configuration (e.g., 16-byte), we have used the random number generator to generate nonces on $n$ bits and feed the BLAKE hash function to produce their corresponding responses, which are also on $n$ bits.

**Communication Implementation.** To allow different boards to communicate with each other, we have used the embedded serial communication medium, which is also known as the URAT (Universal asynchronous receiver-transmitter) system. In fact, on each programming
6.4. PROTOCOL IMPLEMENTATION AND EVALUATION

Figure 6.6: T2T with gateway network configuration using ARDUINO MEGA 2650 boards (Left) and the T2T without gateway network configuration using ARDUINO DUE boards (Right). In the first configuration (Left), the transmission (TXD) GPIO pin of a board (Thing T₁) is connected to the receiving (RXD) GPIO pin of another board (Thing T₂) and vice-versa. Also, their ground pins (GND) have to be connected to each other. In the second configuration (Right), The transmission (TXD) GPIO pin of the left-hand side board (Thing T₁) is connected to the receiving (RXD) GPIO pin of the right-hand side board (Thing T₂) and vice-versa. Also, their ground pins (GND) are connected to each other.

board, there exist some GPIO (General Purpose Input/Output) pins that are dedicated for communication, namely, TXD for transmission and RXD for reception⁶. Thus, to connect two development boards through the serial medium, the TXD pin of one board has to be connected to the RXD pin of the other board and vice-versa. Also, their ground pins (GND) have to be connected to each other. Such wiring is illustrated in Fig. 6.6, where Fig. 6.6 (Left) illustrates the wiring in Configuration 1 and Fig. 6.6 (Right) illustrates the wiring in Configuration 2 of the protocol. For the communication speed, we have set the serial communication rate to 100000 baud and 200000 baud, which is 100kbps and 200kbps, respectively.

Also, we have considered that the communication environment is characterized by the following features: (1) There is no adversary (attacker) present during the execution of the protocol and during its evaluation. (2) It is assumed that there is no communication errors and that none of the messages is lost. (3) The authenticated parties (Thing T₁, Thing T₂, and gateway) are considered not compromised. (4) The protocol is executed and

⁶Note that other GPIO pins (e.g., from 2 to 13) can also be configured to be used for communication through a software library, e.g., SoftwareSerial.
evaluated under an ambient temperature so that the circuits behave normally. (5) Although the gateway is supposed to be a node that has more resources than Things, in this work we consider all the nodes (Things and gateway) to have the same resources and capabilities.

**Protocol Code.** For the protocol to run over an Arduino board, the protocol has to be first written using C/C++ programming language and then compiled for a specific Arduino board (e.g., Arduino Mega 2560 R3) using the Arduino IDE v1.8.13. The protocol as well as the firmware are then uploaded and written into the flash memory of the board (i.e., flashing the board). Tables 6.3 and 6.4 report the size, in Kb, of the sketch (code) of the protocol for each configuration, Arduino development board, node (Thing T₁, Thing T₂, and Gateway), and for different size of the variables, i.e., challenges, responses, nonces, and message integrity codes.

Table 6.3: The size, in Kb, of the protocol’s code (Configuration 1) compiled for both Arduino development boards and for different sizes of the challenges, responses, nonces, and message integrity codes (hash).

<table>
<thead>
<tr>
<th>Board’s name</th>
<th>Node</th>
<th>Variables’ sizes</th>
<th>8 bytes</th>
<th>16 bytes</th>
<th>32 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino Mega 2560</td>
<td>Thing T₁</td>
<td>17470</td>
<td>1740</td>
<td>17782</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Thing T₂</td>
<td>17292</td>
<td>17532</td>
<td>17580</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Gateway</td>
<td>19956</td>
<td>18972</td>
<td>20310</td>
<td></td>
</tr>
<tr>
<td>Arduino DUE</td>
<td>Thing T₁</td>
<td>19620</td>
<td>19740</td>
<td>19980</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Thing T₂</td>
<td>19564</td>
<td>19788</td>
<td>19956</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Gateway</td>
<td>20308</td>
<td>19780</td>
<td>20636</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.4: The size, in Kb, of the protocol’s code (Configuration 2) compiled for both Arduino development boards and for different sizes of the challenges, responses, nonces, and message integrity codes (hash).

<table>
<thead>
<tr>
<th>Board’s name</th>
<th>Node</th>
<th>Variables’ sizes</th>
<th>8 bytes</th>
<th>16 bytes</th>
<th>32 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino Mega 2560</td>
<td>Thing T₁</td>
<td>16894</td>
<td>16908</td>
<td>16554</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Thing T₂</td>
<td>16972</td>
<td>17024</td>
<td>16820</td>
<td></td>
</tr>
<tr>
<td>Arduino DUE</td>
<td>Thing T₁</td>
<td>19516</td>
<td>19564</td>
<td>19388</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Thing T₂</td>
<td>19460</td>
<td>19524</td>
<td>19388</td>
<td></td>
</tr>
</tbody>
</table>

We can notice that the code of the gateway is larger than the code of Things (T₁ and T₂). This is because the gateway has to communicate with two different Things during
6.4. PROTOCOL IMPLEMENTATION AND EVALUATION

a given authentication session. The size of the programs should not necessarily increase by increasing the size of the variables. In fact, the impact of increasing the size of the variables is related to the amount of SRAM (central memory) that will be used to execute the program, which will increase with the size of the allocated variables. Furthermore, we cannot make a relationship between the size of the variables and the size of the codes as well as between the size of the codes on both boards. In fact, the Arduino may use different optimization techniques on different variable sizes and boards. We note that the Arduino compiler was left with default settings with respect to the possible optimization options\(^7\).

6.4.2 Protocol Evaluation

In this subsection, we evaluate the performance of the protocol in terms of execution time, communication overhead, and power consumption. We run the experiments for both protocol configurations, namely, Things with gateway (Configuration 1) and Things without gateway (Configuration 2), in the ideal case (i.e., in the absence of attackers). This would provide us with results that can be compared to the literature.

Execution Time Evaluation

To evaluate the execution time, we have used the embedded timer of the board to retrieve the current time in microseconds. This is possible through the invocation of the \texttt{micros()} function, which returns how many microseconds have passed since the board was booted. Therefore, by invoking the \texttt{micros()} at the beginning of the protocol and then later on after some execution, we can compute how much time the board has taken to execute a specific part of the authentication protocol\(^8\). To that end, we have started by identifying the claim points of the protocol, i.e., the moment where a device makes the claim of successfully

\(^7\)In the \texttt{Arduino platform.txt} file, the compiler can be optimized (by specifying an optimization option: \texttt{-Os, -O0, -O1, -O2, or -O3}) to reduce the code size which would generally result in a faster execution. We left the compiler on its default settings (i.e., option \texttt{-Os}).

\(^8\)We note that the authentication protocol’s code starts executing upon powering up the Arduino-board on which it was uploaded to.
authenticating another device or successfully deriving a cryptographic key (see the **bold** and **green** claims in the MSCs of Fig. 6.2, Fig. 6.3, and Fig. 6.4). For example, during the authentication of two Things through a gateway (Configuration 1), the protocol has the following claim points: (i) Thing $T_i$ authenticates the gateway $G$. (ii) The Gateway $G$ authenticates Thing $T_i$. (iii) Thing $T_i$ authenticates Thing $T_j$. (iv) Thing $T_i$ verifies that Thing $T_j$ has successfully derived a shared key $K$. In the second configuration (Configuration 2), the same claim points exist, but without the claims of authenticating the gateway by both Things. Also, we have wired the boards in such a way so that they start at the same time to reduce the execution time difference. In fact, as each board has its own clock and all boards constitute together a tiny distributed system, the clocks are subject to desynchronization, which may result in inconsistent results. An example of such inconsistency is an event $e_1$ that occurs on board $B_1$ before another event $e_2$ (i.e., $e_1 < e_2$) that occurs on board $B_2$, but the occurrence time of $e_1$ is larger than the occurrence time of $e_2$.

We have measured how much time each board spends to reach to a specific claim point for both protocols configurations (Configuration 1 and 2) and for different sizes of the variables (i.e., challenges, responses, nonces, and message integrity codes). We performed at least 50 consecutive executions to compute the average time for each specific claim. Fig. 6.7 and Fig. 6.8 illustrate the average execution time of the authentication protocol in Configuration 1 for different variables sizes (8, 16, and 32 bytes) and serial communication rates (100kbps and 200kbps), for the Arduino Mega 2650 and Arduino DUE, respectively.

In the case of the Arduino Mega 2650 (R3), the entire protocol (authentication and key derivation confirmation) requires an average execution time of 225.020ms, 288.988ms, and 398.062ms, when the variable size is 8, 16, and 32 bytes, respectively, and the serial communication rate is 100kbps. When the serial communication rate is increased to 200kbps, the protocol requires an average execution time of 214.515ms, 275.849ms, and 351.282ms, when the variable size is 8, 16, and 32 bytes, respectively. Also, the mutual authentication of Thing $T_1$ and $T_2$ requires 221.475ms, 285.425ms, and 394.453ms, when the variable...
### 6.4. Protocol Implementation and Evaluation

The average execution time of the T2T authentication protocol in the presence of the gateway (Configuration 1) on Arduino Mega 2560 R3 boards (8-bit CPU at 16 MHz with 8 KB of SRAM) for different variables’ sizes and serial communication rates (i.e., 100kbps and 200kbps).

![Execution Time Chart](image)

<table>
<thead>
<tr>
<th>Size</th>
<th>100kbps Execution Time (ms)</th>
<th>200kbps Execution Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 bytes</td>
<td>100.970ms, 16 bytes</td>
<td>272.286ms, 32 bytes</td>
</tr>
</tbody>
</table>

It is important to bear in mind that our protocol involves the mutual authentication of two/three communicating parties (Thing T1, Thing T2, and gateway in Configuration 1, and Things T1 and T2 in Configuration 2) as well as the establishment of a symmetric key (with key derivation). Also, the authentication of Thing T1 and T2 with respect to the gateway G is processed sequentially (T1 then T2). Although the authentication execution would run faster if the authentication requests are processed in parallel, we decided to keep the processing sequential for simplicity. More importantly, the hardware platform on which
the protocol is running is an 8-bit microcontroller that runs at a speed of 16MHz with 8Kb of SRAM. Thus, for a resource-constrained platform like this one, the obtained execution time is acceptable if we compare it to the reviewed protocols and ECC-based solutions.

In the case of the Arduino DUE, however, the protocol runs faster. It requires an average execution time of 28.403ms, 52.368ms, and 114.909ms, when the variable size is 8, 16, and 32 bytes, respectively, and the serial rate is 100kbps. When the serial communication rate is 200kbps, the protocol requires 15.739ms, 27.850ms, and 89.150ms, when the variable size is 8, 16, and 32 bytes, respectively. The mutual authentication of Thing T₁ and T₂ requires 28.349ms, 52.327ms, and 114.839ms, for the considered three sizes respectively, and
6.4. PROTOCOL IMPLEMENTATION AND EVALUATION

15.685ms, 27.789ms, and 89.081ms, when the rate is 200kbps. In this second experiment, the protocol runs faster as the hardware platform is a 32-bit microcontroller that operates at a speed of 84MHz with 96KB of SRAM. Although, this hardware configuration is still not that powerful, the average execution time of the protocol is excellent.

In both cases (i.e., Arduino Mega and DUE), the execution time has increased when we have increased the size of the variables from 8 bytes to 16 bytes, and then to 32 bytes. Also, the execution time has improved when we have increased the serial communication rate from 100kbps to 200kbps. We believe that applying this protocol on a communication technology that has a higher data rate (e.g., 1MB in Bluetooth) would considerably improve the execution time. Nevertheless, we have observed that the impact of increasing the serial communication rate as well as the size of the variables is much noticeable on the Arduino DUE board than on the Arduino Mega board. This could be due to the speed difference that distinguish both boards. In fact, on a slower board it is generally difficult to notice the differences between various configurations than on a faster board.

If we consider the serial communication rate of 100kbps, then on the Arduino Mega 2650 board, we can observe that the 16-byte configuration is around 28% slower than the 8-byte configuration, but 100% more secure. Hence, it is worth to sacrifice around 63ms more time for the execution of the protocol to guarantee a double security. Moreover, the 32-byte configuration is around 28% slower than the 16-byte configuration, but 100% more secure. For a double security, the choice of a 32-byte configuration would be straightforward. Also, even though the 32-byte configuration is around 43% slower than the 8-byte configuration, it offers a security strength that is four times stronger than the security of the 8-byte configuration (200% more secure), which confirm our previous choice. The same conclusion can be drawn if we used a higher communication rate, e.g., 200kbps. Nevertheless, on the Arduino DUE board, we can see that by doubling the size of the variables (doubling the security), the execution time doubles. Hence, the choice of a configuration will depend on the application constraints. Although, the 8-byte and the 16-byte configuration are 200%
and 100%, respectively, slower than the 32-byte configuration, the latter offers a stronger security with an acceptable execution time (\(\sim 115\text{ms}\) and \(\sim 89\text{ms}\) for a communication rate of 100kbps and 200kbps, respectively).

<table>
<thead>
<tr>
<th>Variable Size (Bytes)</th>
<th>Execution Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>115</td>
</tr>
<tr>
<td>16</td>
<td>89</td>
</tr>
<tr>
<td>32</td>
<td>100</td>
</tr>
</tbody>
</table>

Figure 6.9: The average execution time of the T2T authentication protocol without gateway (Configuration 2) on Arduino Mega 2560 boards (8-bit CPU at 16 MHz with 8 KB of SRAM) for different variable sizes (i.e., 8, 16, and 32 bytes) and serial communication rates (i.e., 100kbps and 200kbps).

<table>
<thead>
<tr>
<th>Variable Size (Bytes)</th>
<th>Execution Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>20</td>
</tr>
<tr>
<td>16</td>
<td>40</td>
</tr>
<tr>
<td>32</td>
<td>60</td>
</tr>
</tbody>
</table>

Figure 6.10: The average execution time of the T2T authentication protocol without gateway (Configuration 2) on Arduino DUE boards (32-bit CPU at 84 MHz with 96 KB of SRAM) for different variable sizes (i.e., 8, 16, and 32 bytes) and serial communication rates (i.e., 100kbps and 200kbps).
Fig. 6.9 and Fig. 6.10 illustrate the average execution time of the authentication protocol in Configuration 2 for different variables sizes (8, 16, and 32 bytes) for the Arduino Mega 2650 and Arduino DUE development board, respectively. In the case of the Arduino Mega 2650 (R3), the entire protocol (authentication and key derivation) requires an average execution time of 174.857ms, 199.619ms, and 274.488ms, when the variable size is 8, 16, and 32 bytes, respectively, and the serial communication rate is 100kbps. When the communication rate is 200kbps, the execution time is 164.507ms, 182.128ms, and 232.934ms, respectively. Also, the mutual authentication of Thing T1 and T2 requires 96.886ms, 110.160ms 150.450ms, when the variable size is 8, 16, and 32 bytes, respectively, and the communication rate is 100kbps. This requires 90.535ms, 97.418ms, and 124.865ms, when the communication rate is 200kbps, for the respective variable sizes.

Similar to Configuration 1, in the case of the Arduino DUE, the protocol runs faster with an average execution time of 23.025ms, 64.991ms, and 85.500ms, when the variable size is 64, 128, and 256 bits, respectively, and the communication rate is 100kbps. It takes 12.778ms, 29.542ms, and 44.346ms, for the protocol to execute when the serial communication rate is 200kbps. Also, the mutual authentication of Thing T1 and T2 requires 13.958ms, 26.741ms, and 52.286ms, for the considered three sized, respectively, when the serial communication rate is 100kbps. These values get slightly reduced to 7.672ms, 14.111ms, and 26.980ms, respectively, when the serial communication rate is 200kbps.

We can observe the same behavior as the one in Configuration 1. If we consider the serial communication rate of 100kbps, then on the Arduino Mega 2650 board (viz., Fig. 6.9), we can determine that the 16-byte configuration is around 12% slower than the 8-byte configuration, but 100% more secure. Hence, it is worth to sacrifice around 24ms more time for the execution of the protocol to guarantee a double security. Moreover, the 32-byte configuration is around 27% slower than the 16-byte configuration, but 100% more secure. We can also observe that the 32-byte configuration is 36% slower than the 8-byte configuration, but 200% more secure. Considering the average execution time of 274.488ms (when is
100kbps used) and 232.934 (when is 200kbps used), we would definitely choose to apply the 32-byte configuration, which guarantees a higher security with a reasonable execution time. Nevertheless, on the Arduino DUE board (viz., Fig. 6.10), the average execution time is almost doubled when the size of the variables is doubled. That is, for a 100% increase on the security of the system, the average execution time will eventually increase by 100%. Hence, the choice of a configuration in this case depends on the application constraints.

With respect to the execution time, we believe that these results are excellent. For example, if we consider the T2T authentication scheme presented by Barbareschi et al., [BBM+19] and compare its execution time to our scheme of Configuration 2, our protocol is much faster than their protocol. They have used an STMicroelectronics STM32F7 board, which has a 32-bit ARM Cortex M7 CPU, 1MB of flash memory, and 340KB of RAM, to represent a Thing. Also, they have used the Cubieboard Cubietruck board to represent a more powerful device (e.g., a gateway). The Cubietruck board has a 64-bit octacore CPU (A15 × 4 at 2.0GHz and A7 × 4 at 1.3GHz), 2GB of RAM, and 64 GB of storage capacity. They have used Ethernet (IEEE 802.3) to connect both devices together. The authors reported that the mutual authentication takes in average 98.46ms. The authors have used a 16-byte (128-bit) variable size for the challenges and responses [BBM18]. For the same variable size, our protocol performs considerably better on the Arduino DUE board, with an average execution time of 46.991ms (100kbps) and 29.542ms (200kbps). Also, our protocol may execute faster if it uses a higher communication rate as the one provided by Ethernet 802.3.

**Communication Overhead Evaluation**

We analyze the communication overhead of the protocol for both configurations (Configuration 1 and 2). Table 6.5 reports the size, in bytes, of the messages that are exchanged during the protocol execution for both configurations and for different variables sizes. The size of a message doubles along with the size of the variable. For example, the message $m_0$
Table 6.5: The size of the exchanged messages during the authentication protocol execution for both configurations (Configuration 1 and Configuration 2) and for each variable size configuration (i.e., 8, 16, and 32 bytes).

<table>
<thead>
<tr>
<th>Node</th>
<th>Message in MSC</th>
<th>Message Sent to</th>
<th>8 bytes</th>
<th>16 bytes</th>
<th>32 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Config-1</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thing T1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$m_0$</td>
<td>Gateway G</td>
<td>40</td>
<td>80</td>
<td>160</td>
<td></td>
</tr>
<tr>
<td>$m_2$</td>
<td>Gateway G</td>
<td>32</td>
<td>64</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>$m_4$</td>
<td>Thing T2</td>
<td>40</td>
<td>80</td>
<td>160</td>
<td></td>
</tr>
<tr>
<td>Thing T2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$m'_0$</td>
<td>Gateway G</td>
<td>40</td>
<td>80</td>
<td>160</td>
<td></td>
</tr>
<tr>
<td>$m'_2$</td>
<td>Gateway G</td>
<td>32</td>
<td>64</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>$m_5$</td>
<td>Thing T1</td>
<td>40</td>
<td>80</td>
<td>160</td>
<td></td>
</tr>
<tr>
<td>Gateway G</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$m_1$</td>
<td>Thing T1</td>
<td>56</td>
<td>112</td>
<td>224</td>
<td></td>
</tr>
<tr>
<td>$m'_1$</td>
<td>Thing T2</td>
<td>56</td>
<td>112</td>
<td>224</td>
<td></td>
</tr>
<tr>
<td>$m_3$</td>
<td>Thing T1</td>
<td>48</td>
<td>96</td>
<td>192</td>
<td></td>
</tr>
<tr>
<td>$m'_3$</td>
<td>Thing T2</td>
<td>48</td>
<td>96</td>
<td>192</td>
<td></td>
</tr>
<tr>
<td><strong>Config-2</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thing T1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$m_0$</td>
<td>Thing T2</td>
<td>40</td>
<td>80</td>
<td>160</td>
<td></td>
</tr>
<tr>
<td>$m_2$</td>
<td>Thing T2</td>
<td>32</td>
<td>64</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>$m_4$</td>
<td>Thing T2</td>
<td>40</td>
<td>80</td>
<td>160</td>
<td></td>
</tr>
<tr>
<td>Thing T2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$m_1$</td>
<td>Thing T1</td>
<td>56</td>
<td>112</td>
<td>224</td>
<td></td>
</tr>
<tr>
<td>$m_3$</td>
<td>Thing T1</td>
<td>40</td>
<td>80</td>
<td>160</td>
<td></td>
</tr>
</tbody>
</table>

that is sent from Thing T1 to the gateway has a total size of 40 bytes when the variable is expressed in 8 bytes. This size doubles to 80 bytes when the variable is expressed in 16 bytes, and to 160 bytes when the variable is in 32 bytes.

Fig. 6.11 and Fig. 6.12 illustrate the amount of bytes each authenticating party sends during the authentication for Configuration 1 and Configuration 2, respectively. In Configuration 1, the amount of bytes sent by Thing T1 and Thing T2 is equal. The gateway however, needs to send almost the double of what both Things send during the execution of the protocol as the gateway needs to communicate with both Things. In Configuration 2, the amount of bytes sent by Thing T1 is a little bit higher than the amount of bytes sent by Thing T2. This is related to the implementation of the protocol. In fact, in this configuration, we have made Thing T2 generate the symmetric key. This would require Thing T1 to send an additional message to prove to Thing T2 that it has correctly derived the symmetric key (key derivation confirmation). However, in Configuration 1, we made Thing T1 generate the key and combined the authentication of Thing T2 with the verification of
the correct derivation of the key using the same message (viz., message $m_5$ in Fig. 6.3).

In Configuration 1, the protocol requires 432 bytes (3456 bits), 864 bytes (6912 bits), and 1728 bytes (13824 bits), when the size of the variables is 8 bytes, 16 bytes, and 32 bytes, respectively. In Configuration 2, the protocol requires 208 bytes (1664 bits), 416 bytes (3328 bits), and 832 bytes (6656 bits), when the size of the variables is 8 bytes, 16 bytes, and 32 bytes, respectively. The second configuration requires less amount of bytes due to the absence of the gateway. Overall, these amounts of bytes are totally acceptable for a resource-constrained environment where the network bandwidth is limited. In fact, if we consider Wi-Fi, Bluetooth, or ZigBee technology, where the payload size is 1500, 251, and 128 bytes, respectively, we can see that it is possible to send all the messages of the protocol using one single frame even though we adopt the 32 byte configuration.

Moreover, it is important to note that we have arbitrarily chosen that all fields (i.e., nonces, challenges, and hash outputs) in a given message have the same size, either 8, 16, or 32 bytes. It is possible to express these fields in different sizes (adopt a different size...
6.4. PROTOCOL IMPLEMENTATION AND EVALUATION

Figure 6.12: The amount of Bytes sent by both authenticating parties, i.e., Thing $T_1$ and Thing $T_2$, during the T2T authentication protocol in Configuration 2 for different variable sizes (i.e., 8, 16, and 32 bytes).

distribution over the fields of the messages), which would certainly reduce the size of certain messages and improve the communication overhead.

**Energy Consumption Evaluation**

Energy consumption is a fundamental performance metric to evaluate communication protocols, in general, and authentication protocols, in particular. We evaluate the energy consumption of the developed PUF-based T2T mutual authentication protocol, for both configurations, i.e., Configuration 1 and 2, for different variable sizes, i.e., 8 bytes, 16 bytes, and 32 bytes, and for both types of boards, i.e., Arduino Mega and Arduino DUE. However, we only evaluate the energy consumption of the authentication protocol with a serial communication rate of 200kbps.

To compute the electrical energy consumed by each Thing (or board) during $t$ seconds, we apply the Joule's law as follows, where $E$ is the energy in Joule, $V$ is the supplied voltage in Volt, $I$ the intensity of the current flowing to the board in Ampere, and $t$ is the time
6.4. PROTOCOL IMPLEMENTATION AND EVALUATION

duration in Second:

\[ E_{(Joule)} = V_{(Volt)} \times I_{(Ampere)} \times t_{(Second)} \]  \hspace{1cm} (6.10)

Therefore, to calculate the energy that is consumed during the execution of the authentication protocol, we need to know the voltage that is being supplied to the board, the intensity of the electric current flowing within the board during the execution of the protocol, and the execution time of the protocol. The supplied voltage is the operating voltage of the boards. According to the boards’ specifications, it is 5V for the Arduino Mega and 3.3V for the Arduino DUE. The execution times of the authentication protocol, for different configurations, are known from the previous experiments. The intensity of the electrical current, however, needs to be measured during the execution of the authentication protocol. To that end, we have used the Tektronix MSO3014 digital storage oscilloscope to measure the intensity of the current.

To measure the intensity of the electrical current flowing within the board during a period of time using an oscilloscope, we can apply one of the following two approaches: (1) We connect a resistor in series to the power cable of the board. Then, we connect the oscilloscope’s voltage probe cable to both resistor’s extremities to measure the voltage drop between the resistor. Having the measured voltage drop \( V \) and the known resistor’s capacity \( R \), we apply the Ohm’s law \( V = R \times I \) to calculate the intensity of the electrical current \( I \). (2) We connect the oscilloscope’s current probe cable directly to the power cable of the board and measure the intensity of the electrical current that is flowing through it. Although the second approach is straightforward, we have adopted the first approach due to the unavailability of the oscilloscope’s current probe cable in our laboratory. Thus, we have used the oscilloscope to measure the voltage drop over a 10Ω resistor and set up the oscilloscope to automatically apply the Ohm’s law. This will allow us to visualize the current flow instead of the voltage drop (viz. Fig. 6.13 and Fig. 6.14).
Figure 6.13: The experimental setting: An oscilloscope, model Tektronix MSO3014, three Arduino DUE boards (from left to right: Thing T₁, Gateway, and Thing T₂), and 10Ω resistor wired in series on the power cable of an arbitrary board (Thing T₁ in this figure). The oscilloscope’s voltage probe is connected to both resistor’s extremities to measure the voltage drop between the resistor.

Practically, to measure the intensity of the current during the execution of the protocol with a high precision, we need to identify the part of the signal, i.e., the signal of I(t), during which the authentication protocol is executing. To that end, we have reprogrammed each Thing (board) in such a way so that the board performs the following operations: (1) Booting. (2) Stay idle for \( \tau \) ms. (3) Turn an LED light ON for \( \tau \) ms and then turn it OFF. (4) Execute the protocol. (5) Turn the LED light ON for \( \tau \) ms then turn it OFF. For the Arduino Mega board, the duration of \( \tau \) is 1000ms, whereas for the Arduino DUE it is 200ms. This is due to considerable difference in the execution speed of the authentication protocol on both boards. In this way, we can identify the portion of the signal during which the protocol is executing.

Fig. 6.15, 6.16, and 6.17, illustrate the execution of the program (and the protocol) on the Arduino Mega boards for Configuration 1, and for variable sizes 8 bytes, 16 bytes, and 32 bytes, respectively. Similarly, Fig. 6.21, 6.22, and 6.23, illustrate the execution of the
Figure 6.14: The experimental setting: An oscilloscope, model Tektronix MSO3014, three Arduino DUE boards (from left to right: Thing T₁, and Thing T₂), and 10Ω resistor wired in series on the power cable of an arbitrary board (Thing T₁ in this figure). The oscilloscope’s voltage probe is connected to both resistor’s extremities to measure the voltage drop between the resistor.

protocol on the Arduino Mega boards for Configuration 2, and for variable sizes 8 bytes, 16 bytes, and 32 bytes, respectively. We note that these figures have been generated using a sampling rate of 100 samples per second. We can observe in Fig. 6.15, 6.16, and 6.17, that the Arduino Mega board draw an \( \triangledown \)-shaped current intensity during its boot phase (approximately within the first second). Then the current intensity drops down for another 1 second during the board idle state. The current intensity jumps up when the LED light is turned ON for a duration of 1 second. Then it drops down during the authentication protocol execution. Once the protocol terminates, the current intensity jumps up again due to the lighting of the LED for another 1 second before it turns OFF. At this point, we can easily identify that the part of the signal that represents the execution of the protocol is the part of the signal where the current intensity has dropped down between two consecutive lighting of the LED. The same can be observed in Fig. 6.18, 6.19, and 6.20, on Arduino DUE board, except that the DUE board draws an \( \triangledown \)-shaped current intensity during its
boot phase instead of \( \text{M} \) for Configuration 1. The same can be observed in Fig. 6.24, 6.25, and 6.26, for Configuration 2. Schematically, the part of the signal that represents the execution of the protocol is situated within the two dashed vertical lines on both figures.

Figure 6.15: The electrical current (in Ampere) drawn by Thing T\(_1\) (a), Gateway G (b), and Thing T\(_2\) (c), during the execution of the authentication protocol on ARDUINO MEGA boards when the 8-byte configuration is adopted (in Configuration 1).

Figure 6.16: The electrical current (in Ampere) drawn by Thing T\(_1\) (a), Gateway G (b), and Thing T\(_2\) (c), during the execution of the authentication protocol on ARDUINO MEGA boards when the 16-byte configuration is adopted (in Configuration 1).
6.4. PROTOCOL IMPLEMENTATION AND EVALUATION

Figure 6.17: The electrical current (in Ampere) drawn by Thing T₁ (a), Gateway G (b), and Thing T₂ (c), during the execution of the authentication protocol on Arduino MEGA boards when the 32-byte configuration is adopted (in Configuration 1).

Figure 6.18: The electrical current (in Ampere) drawn by Thing T₁ (a), Gateway G (b), and Thing T₂ (c), during the execution of the authentication protocol on Arduino DUE boards when the 8-byte configuration is adopted (in Configuration 1).

Figure 6.19: The electrical current (in Ampere) drawn by Thing T₁ (a), Gateway G (b), and Thing T₂ (c), during the execution of the authentication protocol on Arduino DUE boards when the 16-byte configuration is adopted (in Configuration 1).
6.4. PROTOCOL IMPLEMENTATION AND EVALUATION

Figure 6.20: The electrical current (in Ampere) drawn by Thing T₁ (a), Gateway G (b), and Thing T₂ (c), during the execution of the authentication protocol on ARDUINO DUE boards when the 32-byte configuration is adopted (in Configuration 1).

Figure 6.21: The electrical current (in Ampere) drawn by Thing T₁ (a) and Thing T₂ (b), during the execution of the authentication protocol on ARDUINO MEGA boards when the 8-byte configuration is adopted (in Configuration 2).

Figure 6.22: The electrical current (in Ampere) drawn by Thing T₁ (a) and Thing T₂ (b), during the execution of the authentication protocol on ARDUINO MEGA boards when the 16-byte configuration is adopted (in Configuration 2).
6.4. PROTOCOL IMPLEMENTATION AND EVALUATION

Figure 6.23: The electrical current (in Ampere) drawn by Thing T1 (a) and Thing T2 (b), during the execution of the authentication protocol on Arduino Mega boards when the 32-byte configuration is adopted (in Configuration 2).

Figure 6.24: The electrical current (in Ampere) drawn by Thing T1 (a) and Thing T2 (b), during the execution of the authentication protocol on Arduino DUE boards when the 8-byte configuration is adopted (in Configuration 2).

Figure 6.25: The electrical current (in Ampere) drawn by Thing T1 (a) and Thing T2 (b), during the execution of the authentication protocol on Arduino DUE boards when the 16-byte configuration is adopted (in Configuration 2).
6.4. PROTOCOL IMPLEMENTATION AND EVALUATION

Figure 6.26: The electrical current (in Ampere) drawn by Thing T₁ (a) and Thing T₂ (b), during the execution of the authentication protocol on ARDUINO DUE boards when the 32-byte configuration is adopted (in Configuration 2).

Table 6.6: The execution time (ms) of the protocol measured using the oscilloscope for Thing T₁, Gateway, and Thing T₂ (Configuration 1), on both Arduino boards (i.e., MEGA and DUE), and for different sizes of the variables.

<table>
<thead>
<tr>
<th>Board’s name</th>
<th>Node</th>
<th>Variables’ sizes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>8 bytes</td>
</tr>
<tr>
<td>Arduino Mega 2560</td>
<td>Thing T₁</td>
<td>214.20</td>
</tr>
<tr>
<td></td>
<td>Thing T₂</td>
<td>163.80</td>
</tr>
<tr>
<td></td>
<td>Gateway</td>
<td>135.00</td>
</tr>
<tr>
<td>Arduino DUE</td>
<td>Thing T₁</td>
<td>14.76</td>
</tr>
<tr>
<td></td>
<td>Thing T₂</td>
<td>10.52</td>
</tr>
<tr>
<td></td>
<td>Gateway</td>
<td>07.08</td>
</tr>
</tbody>
</table>

Table 6.7: The execution time (ms) of the authentication protocol measured using the oscilloscope for Thing T₁ and Thing T₂ (Configuration 2), on both Arduino boards (i.e., MEGA and DUE), and for different sizes of the variables.

<table>
<thead>
<tr>
<th>Board’s name</th>
<th>Node</th>
<th>Variables’ sizes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>8 bytes</td>
</tr>
<tr>
<td>Arduino Mega 2560</td>
<td>Thing T₁</td>
<td>159.60</td>
</tr>
<tr>
<td></td>
<td>Thing T₂</td>
<td>161.60</td>
</tr>
<tr>
<td>Arduino DUE</td>
<td>Thing T₁</td>
<td>04.68</td>
</tr>
<tr>
<td></td>
<td>Thing T₂</td>
<td>09.44</td>
</tr>
</tbody>
</table>

To confirm the results of the first experiments (execution time evaluation), we have measured the time length of these parts of the signal that represent the execution of the protocol with the help of the oscilloscope’s cursors. We have found that most of the obtained values are almost equal to the ones computed during the experimental evaluation of the protocol’s execution time, which would constitute a confirmation of the previously obtained
results (cf., Fig. 6.7 and Fig. 6.8). Table 6.6 and Table 6.7 report the execution times measured using the oscilloscope’s cursors for Configuration 1 and 2, respectively. The values shown in green are the results that are approximately equal (+/-2.5ms) to the ones obtained in the previous experiment (i.e., execution time experiments), whereas the values shown in red are the ones which are not exactly the same value that we found in the previous execution time experiments (the values obtained using the oscilloscope are +/-30ms less than the ones obtained from the micros() function on the boards). This is due to the following reason: When we have measured the execution time of the protocol using the oscilloscope, we have considered each Thing (board) individually, in the sense where the measured board executes the protocol irrespective of the time that the other Things started executing the protocol. However, when we evaluated the execution time during the first experiment, we paid attention to have all the boards synchronized so that they all start executing the protocol at the same time. Thus, the precision in the later experiments (i.e., using the micros() function), will depend on clock errors (clock drifting) of each board as well as the manual precision of starting the board together.

It is important to note that the execution time of the entire authentication protocol is expressed by the execution time of Thing T\textsubscript{1} (respectively T\textsubscript{2}) in Configuration 1 (respectively Configuration 2), as the latter Thing is the last authenticating party that finishes executing the authentication protocol.

Most importantly, we have measured the intensity of the electrical current during the execution of the protocol as follows: We have recorded the current intensity values during the protocol execution phase using a sampling rate of 500k samples per second. Then, we have computed the average value of the current intensities. Table 6.8 and Table 6.9 reports the average current intensity measured using the oscilloscope for Configuration 1 and 2, respectively.

In Configuration 1, taking the upper bound of the obtained values, we can state that the average electrical current intensity used by Thing T\textsubscript{1}, Gateway G, and Thing T\textsubscript{2}, is 87mA,
Table 6.8: The average current intensity (mA) drawn by each board during the authentication protocol execution for Configuration 1, on both Arduino boards (i.e., MEGA and DUE), and for different sizes of the variables.

<table>
<thead>
<tr>
<th>Board’s name</th>
<th>Node</th>
<th>Variables’ sizes</th>
<th>8 bytes</th>
<th>16 bytes</th>
<th>32 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino MEGA 2560</td>
<td>Thing T₁</td>
<td>87.91</td>
<td>86.77</td>
<td>86.01</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Thing T₂</td>
<td>77.97</td>
<td>79.21</td>
<td>78.32</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Gateway</td>
<td>75.57</td>
<td>72.78</td>
<td>73.30</td>
<td></td>
</tr>
<tr>
<td>Arduino DUE</td>
<td>Thing T₁</td>
<td>57.70</td>
<td>57.63</td>
<td>57.16</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Thing T₂</td>
<td>61.18</td>
<td>61.27</td>
<td>61.60</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Gateway</td>
<td>58.56</td>
<td>57.87</td>
<td>57.81</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.9: The average current intensity (mA) drawn by each board during the authentication protocol execution for Configuration 2, on both Arduino boards (i.e., MEGA and DUE), and for different sizes of the variables.

<table>
<thead>
<tr>
<th>Board’s name</th>
<th>Node</th>
<th>Variables’ sizes</th>
<th>8 bytes</th>
<th>16 bytes</th>
<th>32 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino MEGA 2560</td>
<td>Thing T₁</td>
<td>80.63</td>
<td>80.60</td>
<td>80.41</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Thing T₂</td>
<td>72.62</td>
<td>73.04</td>
<td>73.23</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Gateway</td>
<td>58.03</td>
<td>57.64</td>
<td>57.21</td>
<td></td>
</tr>
<tr>
<td>Arduino DUE</td>
<td>Thing T₁</td>
<td>60.78</td>
<td>60.92</td>
<td>60.81</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Thing T₂</td>
<td>02.06</td>
<td>04.31</td>
<td>11.82</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Gateway</td>
<td>01.35</td>
<td>01.89</td>
<td>09.07</td>
<td></td>
</tr>
</tbody>
</table>

75mA, and 79mA, respectively, when the Arduino MEGA board is being used. However, it is 57mA, 61mA, and 58mA, when the Arduino DUE board is being used. Also, in Configuration 2, we can state that the average electrical current intensity used by Thing T₁ and Thing T₂ is is 80mA and 73mA, respectively, when the Arduino MEGA board is being used, and it is 58mA and 60mA, when the Arduino DUE is being used.

Table 6.10: The amount of energy (mJ) consumed during the execution of the authentication protocol on Thing T₁, Gateway G, and Thing T₂ (Configuration 1), on both Arduino boards (i.e., MEGA and DUE), and for different sizes of the variables.

<table>
<thead>
<tr>
<th>Board’s name</th>
<th>Node</th>
<th>Variables’ sizes</th>
<th>8 bytes</th>
<th>16 bytes</th>
<th>32 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino MEGA 2560</td>
<td>Thing T₁</td>
<td>93.18</td>
<td>117.54</td>
<td>151.554</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Thing T₂</td>
<td>61.42</td>
<td>75.52</td>
<td>98.40</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Gateway</td>
<td>53.32</td>
<td>71.61</td>
<td>91.32</td>
<td></td>
</tr>
<tr>
<td>Arduino DUE</td>
<td>Thing T₁</td>
<td>02.77</td>
<td>05.19</td>
<td>12.22</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Thing T₂</td>
<td>02.06</td>
<td>04.31</td>
<td>11.82</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Gateway</td>
<td>01.35</td>
<td>01.89</td>
<td>09.07</td>
<td></td>
</tr>
</tbody>
</table>

At this stage, knowing the average current intensity used by each Thing (board) during
Table 6.11: The amount of energy (mJ) consumed during the execution of the authentication protocol on Thing T1 and Thing T2 (Configuration 2), on both Arduino boards (i.e., Mega and DUE), and for different sizes of the variables.

<table>
<thead>
<tr>
<th>Board’s name</th>
<th>Node</th>
<th>Variables’ sizes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8 bytes</td>
<td>16 bytes</td>
</tr>
<tr>
<td>Arduino Mega 2560</td>
<td>Thing T1</td>
<td>63.84</td>
</tr>
<tr>
<td></td>
<td>Thing T2</td>
<td>58.98</td>
</tr>
<tr>
<td>Arduino DUE</td>
<td>Thing T1</td>
<td>00.89</td>
</tr>
<tr>
<td></td>
<td>Thing T2</td>
<td>01.87</td>
</tr>
</tbody>
</table>

Table 6.12: The amount of energy (mJ) consumed during the execution of the authentication protocol on Thing T1 and Thing T2 (Configuration 2), on both Arduino boards (i.e., Mega and DUE), and for different sizes of the variables.

the execution of the protocol, and knowing the amount of time needed to execute the protocol, we have computed the amount of energy that is consumed by each Thing during the execution of the protocol by applying the Joule’s law (which is expressed in Equation 10). Table 6.10 and Table 6.12 report the energy consumed by each Thing during the execution of the authentication protocol, on both boards, and for different variable sizes.

6.5 Discussion

In this section, we discuss the security and performance features of T2T-MAP and compare them with the protocols described in the related work (Chapter 5). Then, we discuss the application of T2T-MAP in IoT, in general, and in short-range IoT wireless communication technologies, in particular.

6.5.1 T2T-MAP Performance and Security

T2T-MAP provides security as well as performance properties. For security, the protocol allows the mutual authentication of devices using their embedded PUF circuits. This important feature makes T2T-MAP more secure than the reviewed related work protocols, where the PUF is used only by one authenticating party. For example, by imposing the use of PUFs on each authenticating party, T2T-MAP enforces the service of non-repudiation.
Also, the protocol allows the establishment of a secret key that will be used to provide message confidentiality and integrity. In addition, the protocol provides forward security and prevents replay attacks through the use of an eCRP-update procedure and fresh nonces, respectively. Furthermore, we have shown, in Chapter 5, that most of the reviewed protocols (9 out of 15) are vulnerable to CRPs disclosure (e.g., during their storage or transmission) and PUF impersonation by malicious insiders. T2T-MAP thwarts these attacks by adopting the concept of extended CRPs (eCRPs) along with the cryptographic concept of distributed value (cf., Subsection 6.2.1). Moreover, in Subsection 6.3.2, we have discussed the resilience of T2T-MAP against machine learning attacks, replay attacks, node compromising attacks, session hijacking, malicious insider spoofing, brute force attacks, and secret key disclosure. Also, we have used Tamarin security protocol verifier to automatically prove the security of T2T-MAP by proving the property of secrecy.

In terms of performance characteristics, T2T-MAP is a lightweight protocol that uses PUFs, simple hash functions, and the bitwise exclusive logical OR operator (XOR). Also, in the case where an encryption is used to secure the communications, lightweight ciphers, such as ChaCha [IET16], can be used to maintain the property of lightweightness. This important feature allows the protocol to be adopted by resource-constrained devices. In addition, the protocol is considered to be scalable as it theoretically allows a large number of Things to be mutually and securely authenticated without any constraints.

We have implemented the protocol on resource-constrained devices and evaluated the performance of the protocol w.r.t. execution time, energy consumption, and communication overhead, for two different architectures (Configuration 1 and 2), three different configurations (w.r.t. variable sizes), and under two different hardware platforms. In Configuration 1, T2T-MAP allows a mutual authentication of three parties (e.g., Thing T₁, Thing T₂, and gateway) to be performed in 214.20ms, 270.20ms, and 348.40ms, when variables (i.e., nonces, PUF’s challenges, PUF’s responses, and hash outputs) are expressed in 8 bytes, 16 bytes, and 32 bytes, respectively, on Arduino Mega boards. This authentication is even
faster on Arduino DUE boards, where it takes around 14.76ms, 27.58ms, and 65.00ms, for the authentication to be done for the respective variable sizes. In Configuration 2, T2T-MAP allows a mutual authentication of two Things (e.g., Thing T1 and T2) to be performed in 161.60ms, 182.80ms, and 231.60ms, when variables (i.e., nonces, PUF’s challenges, PUF’s responses, and hash outputs) are expressed in 8, 16, and 32 bytes, respectively, on Arduino Mega boards. However, similar to Configuration 1, on Arduino DUE boards, the execution is faster as it takes 9.44ms, 23.82ms, and 45.20ms, for the respective variables’ sizes. This execution times sound perfect with respect to the execution times reported by the related work on protocols that perform authentication between two entities, where at least one of them is resourceful.

With respect to energy consumption, T2T-MAP shows reasonable results. Depending on the device (i.e., being Thing T1, T2, or gateway), in Configuration 1, the energy consumption varies between 53.32mJ and 93.18mJ, 71.61mJ and 117.54mJ, and 91.32mJ and 151.55mJ, for the respective variable sizes on Arduino Mega. However, energy consumption is considerably lower on Arduino DUE as it varies between 1.35mJ and 2.77mJ, 1.89mJ and 5.19mJ, and 9.07mJ and 12.22mJ, for the respective variable sizes. Also, in Configuration 2, the energy consumption varies between 58.98mJ and 63.84mJ, 66.72mJ and 68.00mJ, and 84.52mJ and 88.12mJ, for the respective variable sizes. On Arduino DUE boards, the consumption is lower and varies between 0.89mJ and 1.87mJ, 3.39mJ and 4.72mJ, and 6.35mJ and 8.95mJ, for the respective variable sizes. These values are reasonably low for resource-constrained devices, such as the ones used in IoT infrastructures.

During the protocol communication, the bandwidth consumption reveals an acceptable overhead. In fact, in Configuration 1, where three authenticating parties are involved, the protocol requires the transmission of 432 bytes, 864 bytes, 1728 bytes, for the respective variable sizes. Also, in Configuration 2, the protocol requires the transmission of 208 bytes, 416 bytes, 832 bytes, for the respective variable sizes. This communication overhead is perfectly suitable for networks with limited bandwidths. Also, it is possible to distribute the
size of the variables in a non-uniform way where the different fields are expressed in unequal sizes. This would certainly improve the communication overhead as certain messages will have a smaller size.

To appraise the efficiency of our authentication protocol, we wanted to compare our performance results with the results of the related work. Nevertheless, we found that it was a very challenging task that may end up drawing unfair conclusions. In fact, we were not able to perform any consistent comparison due to the following reasons: (1) Many implementations from the related work use powerful devices that are not resource-constrained devices, which makes it unfair to compare with an implementation that is fully resource-constrained. (2) Most of the related work do not provide enough information about the configurations and the results. This leads to an incomplete comparison. (3) Most, if not all, related work, do not provide the complete structure of the messages that are exchanged during the execution of the protocol. Making random assumptions on the structure of the messages results in an unfair comparison.

6.5.2 Experimental Environment

During the development of T2T-MAP authentication protocol, we have used serial wired communication between authenticating parties instead of a short-range wireless technology. This actually was an arbitrary choice which we justify in the following paragraphs:

• To broaden the usability and application of the T2T-MAP protocol, we have kept the protocol as generic as possible and did not make it dependent on any wireless communication technology. In this way, the protocol can be used on any wireless communication technology, including those considered in Chapter 2.

• To preserve the security of T2T-MAP and solve the security issues (most attacks) discussed in Chapter 3, we have intended to integrate the protocol, or at least, the concepts that the protocol introduces in future MAC-layer security protocols that wireless communication
6.6. CONCLUSION

technologies will adopt. In fact, if the T2T-MAP is used as a high-level security protocol that runs on top of the MAC-layer, then the attacks that we have presented in Chapter 3 will still be possible, as the latter (attacks) target MAC-layer security and communication protocols.

• Running T2T-MAP protocol at higher-layers will add burden to the system which is supposed to be a resource-constrained environment. We initially planned that T2T-MAP should be used at the MAC-layer. However, we do not claim that the protocol should be used in its entirety as a MAC-layer security protocol, but urge to use the concepts that the protocol introduces, e.g., PUFs, extended CRPs, and distributed values, to implement future MAC-layer security protocols for IoT wireless technologies. In fact, wireless security protocols are not limited to only performing authentication, but also enforcing security services over different types of frames. For example, in Wi-Fi technology, a security protocol should provide security for data, management, and control frames.

• The performance of the T2T-MAP will certainly vary from one wireless technology to another. In fact, each technology has a data rate that is different from another technology. For example, Wi-Fi’s data rate can reach 6.75Gbps (802.11ac), whereas Bluetooth is limited to 1Mbps and ZigBee to 250kbps. Also, the radio module for each communication technology has its own specifications w.r.t. power consumption.

6.6 Conclusion

In this chapter, we have developed T2T-MAP, a lightweight mutual authentication protocol for T2T architectures in the context of IoT. The protocol applies PUFs (Physical Unclonable Functions), as a physical security-by-design technology, to allow Things to efficiently authenticate each other with the lowest cost. The protocol can be used in two different authentication architectures. The first configuration allows two Things to authenticate each other and derive a secret key through a central gateway, whereas in the second
configuration Things can authenticate each other and derive a secret key without involving any gateway. Compared to the existing PUF-based authentication protocols, we claim that T2T-MAP is more resilient to various attacks, such as CRPs (Challenge-Response Pairs) disclosure, malicious insider, replay, session hijacking, brute force, secret disclosure, machine-learning, and node compromising attacks. Furthermore, it provides mutual authentication, non-repudiation, forward-security, and liveness. Besides being resilient to various attacks, T2T-MAP is scalable, lightweight, and energy-efficient.
The IoT (Internet of Things) connects billions of heterogeneous devices, called Things, using different communication technologies and protocols to provide end-users, all over the world, with access to a variety of smart applications. It also invites cybercriminals who exploit the IoT infrastructures to conduct large scale, distributed, and devastating cyberattacks. The security of IoT infrastructures strongly depends on the security of its wired and wireless infrastructures. While the wireless infrastructure is thought to be the most outspread part in IoT, it is at the same time the most vulnerable and accessible for attackers. Hence, more focus should be placed on the security of wireless infrastructures of IoT, in particular, on the infrastructures that adopt short-range wireless communication technologies. In fact, the number of the short-range wireless infrastructures is unbelievably growing every year, creating more and more security holes in IoT and making the attack surface wider for cybercriminals.

Furthermore, if we carefully analyze the cause behind the most recent IoT cyberattacks, we can easily draw that these cyberattacks are due to vulnerabilities in the adopted authentication protocols. Indeed, as the service of authentication constitutes the spine of all security properties, any vulnerability residing on the applied authentication protocol will be exploited to generate security attacks that may breach the authenticity, confidentiality, integrity, and availability of the system. Thus, authentication is considered the first step
toward the prevention of any cyberattack.

Due to the heterogeneity and the particular security requirements of IoT, developing secure, low-cost, and lightweight authentication protocols has become a serious challenge. This has excited the research community to design and develop new authentication protocols that meet IoT requirements. In fact, ten years after the emergence of IoT, a large number of authentication protocols were designed and proposed for various IoT wired or wireless infrastructures. These protocols relied on the application of classical cryptographic mechanisms. However, these protocols suffered from known security vulnerabilities that are related to classical cryptographic mechanisms, which resulted in the re-occurrence of known cyberattacks but with an IoT flavor. Hence, developing new authentication protocols that adopt new security concepts and that better meet IoT security and performance requirements, has become an obligation as the size of IoT is growing at a very high speed.

Finally, exploiting vulnerabilities in authentication protocols to generate attacks on availability has become a common attack methodology that cybercriminals adopt. Such attacks require a cheaper cost, less effort, but could create lot of damage and expensive consequences. Exploiting authentication protocols for the sake of generating denial of service attacks should be seriously investigated.

7.1 Summary

To address the security concerns that we have highlighted above, we have organized this thesis into two main parts:

- In the first part, we have focused on the security of IoT wireless infrastructures. We have reviewed the most used IoT short-range wireless communication technologies, namely, WiFi, Bluetooth, ZigBee, and RFID. We started by presenting the four wireless communication technologies and discussed their respective security mechanisms. This analysis served as an important background for the reader to understand the terminology and notions presented
later in the thesis. We have developed a service-based attack classification to categorize attacks in wireless IoT. The taxonomy allows a simple but concise categorization of attacks, which would certainly help security-engineers to better visualize and filter attacks of ones concern. We have adopted the classification to review the attacks that happened during the past two decades on the considered technologies. We have also discussed the countermeasures that are possible to apply in order to mitigate, or at least delay the attacks. We have analyzed the security of authentication mechanisms of Wi-Fi and Bluetooth technologies. We have presented new vulnerabilities and demonstrated the feasibility of various denial of service attacks. Evidently, we have proposed possible countermeasures to fix the vulnerabilities. Furthermore, we have studied the application of FHSS (Frequency Hopping Spread Spectrum) as a physical-layer security mechanism for PKESs (Passive Keyless Entry and Start Systems) to mitigate relay attacks and prevent auto-thefts.

- In the second part, we have focused on PUF-based authentication for IoT. We have reviewed and analyzed the security of the most recent PUF-based authentication protocols for IoT. We have demonstrated possible attacks on the reviewed protocols and have drawn security lessons to consider in future protocol design. Then, we have designed and implemented T2T-MAP, a lightweight PUF-based authentication protocol for Thing-to-Thing communications. We have evaluated the security as well as the performance of the protocol.

7.2 Contributions

In this thesis, we have specifically made the following contributions:

1. We have developed a security service-based attack taxonomy to classify attacks on Wi-Fi, Bluetooth, ZigBee, and RFID. We have adopted the classification to review attacks on these technologies and discuss possible countermeasures to mitigate the attacks. The taxonomy would help security-engineers to better focus on specific type or category of attacks rather than having a haphazard long list of attacks that would
7.2. CONTRIBUTIONS

require additional efforts to filter and understand

2. We have identified and exploited security vulnerabilities in the authentication protocols of Wi-Fi and Bluetooth technologies and demonstrated the feasibility of various DoS (Denial of Service) attacks. Also, we have proposed countermeasures to fix the discovered vulnerabilities. The vulnerabilities that we have identified, as well as, the attacks that we have generated would allow protocol designers and security-engineers to perceive vulnerabilities from a new angle that has not been “seriously” discussed in the literature (i.e., attacks through race-condition).

3. We have studied the application of FHSS (Frequency Hopping Spread Spectrum) in RFID technology, in general, and in PKESs (Passive Keyless Entry and Start Systems), in particular, for the sake of proposing a fundamental countermeasure to mitigate relay attacks for auto-theft. We confidently consider our studies as a first step toward a complete and efficient countermeasure for relay attacks that would wake up car manufacturers and car insurance companies from an endless nightmare.

4. We have analyzed the security of recent PUF (Physical Unclonable Function)-based authentication protocols for IoT and demonstrated how their claimed security can be breached through various types of attacks. We have drawn security lessons to be considered by future authentication protocol designers so that future authentication protocol implementations are free from the identified security flaws.

5. We have designed a lightweight PUF-based authentication protocol for Thing-to-Thing (T2T) communications. We have analyzed the security of the protocol and have shown its resilience against various attacks. Also, we have proven the security of the protocol using TAMARIN, a well-established protocol verifier. Furthermore, we have implemented the protocol on resource-constrained devices and conducted performance analysis. We have shown that the protocol is efficient in terms of execution
time, communication overhead, and energy consumption. The protocol can be used in IoT resource-constrained applications to establish secure and efficient authentication.

7.3 Limitations and Future Work

In the following paragraphs, we present the limitations that we have identified throughout this thesis and left as potential research directions for future work:

We have analyzed the security of the most recent PUF (Physical Unclonable Functions)-based authentication protocols for IoT. We have pointed out security weaknesses on each protocol and have demonstrated the feasibility of various attacks. However, we have manually analyzed those protocols without the use of any software tool, such as protocol verifiers. In fact, we have noticed that current security protocol verifiers, such as Tamarin, AVISPA, Isabelle/HOL and ProVerif, do not integrate the concept and properties that are related to PUFs. Protocol designers have to manually express the properties, which makes it time-consuming. This seems to be a research direction that others would benefit from.

We have developed T2T-MAP, a Thing-to-Thing mutual authentication protocol that uses PUFs for IoT. We have shown that the protocol allows a secure authentication with reasonable performance. Nevertheless, despite all the advantageous features that T2T-MAP provides, the protocol still has some limitations as nothing is perfect. First of all, the protocol is designed in such a way so that each device, i.e., Thing, stores only one eCRP about any other device. We have assumed the existence of an eCRP update procedure so that devices can renew the stored eCRPs and maintain the property of forward-security as well as the resilience against brute force attacks. This eCRP update procedure may consist of storing one additional eCRP per device during the enrolment phase and particularly use these additional eCRPs for the update procedure. We plan to investigate and integrate this procedure on a future version of the protocol. Furthermore, along with all related work protocols, T2T-MAP is vulnerable to race condition-based attacks. We plan to improve
the security of T2T-MAP with respect to these attacks by implementing a proper countermeasure and use a real PUF, e.g., an SRAM-PUF, to deploy the protocol on a real case IoT application. Finally, we will investigate the possibility of integrating the protocol, or at least its concepts, to design a MAC-layer authentication protocol for short-range wireless communication technologies, e.g., Wi-Fi, Bluetooth, and ZigBee.
Bibliography


[Exp18] Express. Two Mercedes’ worth over £111,000 stolen from driveway in seconds - is yours at risk too? https://www.express.co.uk/life-style/cars/, 2018. article 1050901.


[RMA15] A. C. D. Resende, K. Mochetti, and D. F. Aranha. PUF-based Mutual Multi-
factor Entity and Transaction Authentication for Secure Banking. In Light.

Mile Away. https://av.tib.eu/media/36217 (Last access date: 12 Nov,

Devices Security Testing Framework. In IEEE 11th International Conference

[RS14] U. Rührmair and J. Sölter. PUF Modeling Attacks: An Introduction and
Overview. In Design, Automation Test in Europe Conference Exhibition, pages 1–
6, 2014.

Modeling Attacks on Physical Unclonable Functions. In 17th ACM conference

Dror, J. Schmidhuber, W. Burleson, and S. Devadas. PUF Modeling Attacks
on Simulated and Silicon Data. In IEEE Trans. Information Forensics and

[RSW+18] E. Ronen, A. Shamir, A. Weingarten, and C. O’Flynn. IoT Goes Nuclear:

[RXS+14] U. Rührmair, X. Xu, J. Sölter, A. Mahmoud, M. Majzoobi, F. Koushanfar,
and W. Burleson. Efficient power and timing side channels for physical unclon-
able functions. In Cryptographic Hardware and Embedded Systems, pages 476–
492, 2014.

hacking_btle_i_am_jacks_heart_monitor_mikeryan_toorcon (Last
access date: 12 Nov, 2020), 2012.

[SB07] D. Spill and A. Bittau. BlueSniff: Eve Meets Alice and Bluetooth. In First

[SC10] F.-L. W. Frank Stajano and B. Christianson. Multichannel Protocols to Pre-

[Sch18] B. Schneier. Click Here to Kill Everybody, Security and Survival in Hyper-

com/academic/archives/1999/12/attack_trees.html (Last access date:


