IoT Security in the Era of Artificial Intelligence

by

Dianqi Han

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved July 2022 by the
Graduate Supervisory Committee:

Yanchao Zhang, Chair
Martin Reisslein
Guoliang Xue
Junshan Zhang

ARIZONA STATE UNIVERSITY

August 2022

ABSTRACT

The security of Internet-of-Things (IoT) is essential for its widespread adoption. The recent advancement in Artificial Intelligence (AI) brings both challenges and opportunities to IoT security. On the one hand, AI enables better security designs. On the other hand, AI-based advanced attacks are more threatening than traditional ones. This dissertation aims to study the dual effects of AI on IoT security, specifically IoT device security and IoT communication security.

Particularly, this dissertation investigates three important topics: 1) security of acoustic mobile authentication, 2) Deep Learning (DL)-guided jamming attacks on cross-technology IoT networks, and 3) DL-powered scalable group-key establishment for large IoT networks. Chapter 2 presents a thorough study on the security of acoustic mobile authentication. In particular, this chapter proposes two mobile authentication schemes identifying the user's mobile device with its linear and nonlinear acoustic fingerprints, respectively. Both schemes adopt the Data Mining (DM) techniques to improve their identification accuracy. This chapter identifies a novel fingerprint-emulation attack and proposes the dynamic challenge and response method as an effective defense. A comprehensive comparison between two schemes in terms of security, usability, and deployment is presented at the end of this chapter, which suggests their respective suitable application scenarios. Chapter 3 identifies a novel DL-guided predictive jamming attack named DeepJam. DeepJam targets at cross-technology IoT networks and explores Deep Reinforcement Learning (DRL) to predict the victim's transmissions that are not subject to the Cross-Technology Interference (CTI). This chapter also proposes two effective countermeasures against DeepJam for resource capable and resource constrained IoT networks, respectively. Chapter 4 proposes a drone-aided DL-powered scalable group-key generation scheme, named

DroneKey, for large-scale IoT networks. DroneKey is a physical-layer key generation scheme. In particular, DroneKey actively induces correlated changes to the wireless signals received by a group of devices and explores DL techniques to extract a common key from them. DroneKey significantly outperforms existing solutions in terms of the scalability and key-generation rate.

DEDICATION

To my parents, my beloved wife, and my soon-to-be-born daughter.

ACKNOWLEDGMENTS

I owe my gratitude to the people who have accompanied me all the way here and made this challenging journey so colorful and enjoyable.

First of all, I feel tremendously grateful to my advisor, Dr. Yanchao Zhang, for his consistent support, guidance, and encouragement. Dr. Zhang, thank you for opening the door of academic research for me and guiding me through this challenging journey. You have set a great example as a scholar and advisor. I will benefit from your supervision for the rest of my life.

I would also like to express my thanks to Dr. Martin Reisslein, Dr. Guoliang Xue, and Dr. Junshan Zhang for serving on my dissertation committee. I really appreciate their guidance and help.

I also give my thanks to Dr. Rui Zhang for his guidance in research and help in life. I especially appreciate his detailed coaching in research, writing, and presentation skills. It's my great honor to work with him.

I would like to acknowledge my caring and inspiring friends and colleagues, including Dr. Jinxue Zhang, Dr. Jingchao Sun, Dr. Xiaocong Jin, Dr. Yimin Chen, Dr. Tao Li, Dr. Junwei Zhang, Dr. Xin Yao, Ang Li, Yan Zhang, Jiawei Li, and Yan (Ian) Zhang, at Cyber and Network Security Group (CNSG).

My biggest thanks to my family! I thank my parents, Xianxia Du and Jingqiao Han, for their unconditional love and uninterrupted support. I thank my parents-in-law, Jie Sun and Hexian Xu, for their encouragement and understanding. A special thanks goes to my beloved wife and best friend, Jingyi Xu, for being by my side all the time.

I also gratefully acknowledge the financial support I received from the National Science Foundation through grant CNS-1619251, CNS-1824355, CNS-1933069, and CNS-2055751.

TABLE OF CONTENTS

Page

LIST OF TABLES x

LIST OF FIGURES xi

CHAPTER

1 INTRODUCTION 1

2 THE SECURITY AND USABILITY OF ACOUSTIC MOBILE AUTHENTICATION 3

2.1 Overview 3

2.2 System and Adversary Models 5

2.2.1 A Generic Acoustic Mobile Authentication Model 5

2.2.2 Adversary Model 7

2.3 FRC-based Mobile Authentication: Proximity-Proof 9

2.3.1 Attacks Against Proximity-Proof 12

2.3.2 Proximity-Proof Design 13

2.3.2.1 Acoustic Transmission of 2FA Response 13

2.3.2.2 FRC Fingerprinting 16

2.3.2.3 Cross-Device Ranging 22

2.3.2.4 Self-Proof Case 25

2.3.3 Evaluation of Proximity-Proof 25

2.3.3.1 Implementation 26

2.3.3.2 Efficacy of Acoustic Fingerprinting 26

2.3.3.3 Security Against MiM attacks 28

2.3.3.4 Security Against Co-Located Attacks 31

2.3.3.5 Security Against Fingerprint-Emulation Attack 33

2.3.3.6 Countermeasure Against Fingerprint-Emulation Attack 34
2.4 ANP-based Mobile Authentication 37
2.4.1 M-ANP Design 38
2.4.1.1 Challenge audio 38
2.4.1.2 Fingerprint extraction and matching 40
2.4.1.3 Overall performances of M-ANP 44
2.4.2 SM-ANP Design 45
2.4.2.1 Challenge audio 45
2.4.2.2 Fingerprint extraction and matching 47
2.4.2.3 Parameters for SM-ANP 48
2.4.2.4 Overall performances of SM-ANP 49
2.4.3 Attacks Against M-ANP and SM-ANP 49
2.4.3.1 Fingerprint-Emulation Attack 50
2.4.4 Dynamic Challenge-Response for M-ANP 50
2.4.4.1 ANP M-Print space 51
2.4.4.2 System parameters 52
2.4.4.3 Security analysis 55
2.4.5 Dynamic Challenge-Response for SM-ANP 56
2.4.5.1 ANP SM-Print space. 56
2.4.5.2 System parameters 57
2.5 Comparison between FRC and ANP Authentication Systems 59
2.6 Related Work 60
2.7 Conclusion and Future Work 62

CHAPTER Page

3 DEEPJAM: DL-GUIDED JAMMING ATTACK ON CROSS-TECHNOLOGY IOT NETWORKS 64
3.1 Overview 64
3.2 Background 67
3.2.1 Zigbee MAC layer 67
3.2.2 Cross-Technology Interference (CTI) Illustration 68
3.3 System and Adversary Models 69
3.3.1 System Model 69
3.3.2 Adversary Model 70
3.4 Problem Formulation 71
3.4.1 Slot Duration 71
3.4.2 Slot Status 72
3.4.3 DeepJam Basics 74
3.5 DeepJam Details 75
3.5.1 Reinforcement Learning 75
3.5.1.1 Slot States 76
3.5.1.2 Actions 77
3.5.1.3 Rewards 77
3.5.2 RL Algorithm of DeepJam 78
3.6 Countermeasures 81
3.7 Evaluation 83
3.7.1 Evaluation Setup 83
3.7.2 Efficiency of DeepJam 85
3.7.2.1 Performance comparison 85

3.7.2.2 Impact of WiFi traffic loads 87
3.7.2.3 Impact of MAC and CCA methods 88
3.7.2.4 Impact of Zigbee devices' amount $\mathcal{N}_D$ 89
3.7.3 Latency 90
3.7.4 Efficiency of Dynamic Network Configuration 90
3.8 Related Work 91
3.9 Conclusion 92
4 DRONEKEY: DRONE-AIDED DL-POWERED SCALABLE GROUP-KEY ESTABLISHMENT FOR LARGE-SCALE IOT NETWORKS 93
4.1 Overview 93
4.2 Background and Feasibility Study 99
4.2.1 Background of CSI 99
4.2.2 Feasibility Study 100
4.3 System Overview and Adversary Model 102
4.3.1 System Model 102
4.3.2 DroneKey Workflow 103
4.3.3 Adversary Model 106
4.4 CSI Extraction and Processing 107
4.4.1 CSI Extraction 108
4.4.2 CSI Stream Processing 108
4.5 Key-generation DNN 111
4.5.1 GKG Dataset Generation 112
4.5.2 GKG DNN Training 114
4.6 Key Generation and Reconciliation 117

4.6.1 Key-source Vector Generation . . . 117
4.6.2 Key-source Vector Quantification . . . 118
4.6.3 Reconciliation . . . 120
4.7 Security Analysis . . . 121
4.7.1 Security against Malicious-drone Attack . . . 121
4.7.2 Security against Eavesdropping and Reproduction Attacks . . 121
4.8 Evaluation . . . 125
4.8.1 Implementation . . . 125
4.8.2 Performance Metrics . . . 126
4.8.2.1 Basic Experiment . . . 126
4.8.2.2 Impact of Network Scale . . . 128
4.8.2.3 Impact of Environmental Factors . . . 130
4.8.3 Overhead of DroneKey . . . 132
4.8.4 Whole-network Group-key Generation . . . 134
4.9 Related Work . . . 135
4.10 Conclusion . . . 138
5 CONCLUSION AND SUMMARY . . . 140
REFERENCES . . . 141

LIST OF TABLES

Table Page

1. Mobile Devices in Experiments. 44
2. The Overall Performance of M-ANP. 45
3. The Overall Performance of SM-ANP. 49
4. Pros & Cons of FRC and ANP Authentication Systems. 59
5. Performance with Different WiFi Traffic Loads. 87
6. Performance with Different MAC and CCA Methods. 88
7. Performance with Different Numbers of Zigbee Devices. 89
8. The Comparison between DroneKey and Representative prior Work. 94
9. The Impact of Environment Changes. 133

LIST OF FIGURES

Figure Page

1. A Generic Mobile Authentication System. 5
2. The Workflow of Proximity-Proof. 11
3. Attacks against Proximity-Proof. 14
4. Frequency Response Curves of the Speaker on a Samsung Galaxy S5, Measured by Two Nexus 7. 17
5. F_1 Scores for Different τ. 21
6. Illustration of Two-Way Acoustic Ranging. 22
7. The Acoustic Fingerprints of a Samsung Galaxy 5 Extracted by Different Devices. 27
8. Distance between One Device's Acoustic Fingerprints Extracted by Different Login Devices. 28
9. Distance between Acoustic Fingerprints of Different Devices. 29
10. Resilience to MiM Attacks. 29
11. Distance between an Original Fingerprint and the Fingerprint Extracted from a Replayed Audio. 30
12. Ranging Errors in Different Environments 32
13. Success Rates under Different Distance. 32
14. Success Rates of the Fingerprint-Emulation Attack. 36
15. F_1 Score Corresponding to Different κ and Threshold. 44
16. The Ratios of Distinguishable Elements. 55
17. Targeted CTI Context of DeepJam. 66
18. Zigbee and WiFi Channels. 68
19. Slot Status. 73

Figure Page

20. DeepJam DNN Architecture. 79

21. Comparison of DeepJam, Random Jamming, and Reactive Jamming. 86

22. The Correlated CSI Streams. 101

23. The System Model of DroneKey. 103

24. The Workflow of DroneKey. 104

25. Raw and Processed Mag and Phase Streams. 109

26. The Architecture of a Key-Generation DNN. 115

27. The Impact of the Group-Coverage Area. 130

Chapter 1

INTRODUCTION

Security guarantee is essential for the widespread adoption of IoT systems, and the recent advancement in AI brings both challenges and opportunities to IoT security. On the one hand, AI enables better security designs. On the other hand, AI-based advanced attacks are more threatening than traditional ones. This dissertation aims to study the dual effects of AI on IoT security, specifically IoT device security and IoT communication security.

Chapter 2 investigates the security and usability of acoustic mobile authentication. Mobile authentication explores the user's mobile device as a proof of the user's identity, and acoustic mobile authentication identifies the user's mobile device by fingerprinting the hardware its acoustic components. Different hardware characteristics have be investigated in previous acoustic fingerprinting schemes. This chapter try to answer the question which of them is more suitable for mobile authentication. In particular, this chapter proposes two acoustic mobile authentication schemes exploring acoustic components' linear and non-linear features, respectively. Their security, usability, and deployment are compared using experimental evaluations and theoretical analysis. In terms of security, this chapter identifies the novel fingerprint-emulation attack and proposes the dynamic challenge and response mechanism as an effective defense. Based on the comparison, this chapter concludes the suitable application scenarios for each of these two schemes.

Chapter 3 studies the jamming attack on cross-technology IoT networks. IoT networks in the future are expected to contain devices of different technologies such as WiFi, Bluetooth, and Zigbee. These technologies may have frequency overlapping channels in the free ISM band, and the Cross-Technology Interference (CTI) thus happens. This chapter identifies a new vulnerability of cross-technology IoT networks. In particular, this chapter proposes a new jamming attack named DeepJam. DeepJam explores Deep Reinforcement Learning (DRL) to predict the victim's transmissions not subject to CTI and can thus launch the jam attack in a more stealthy manner. The evaluation confirms DeepJam's advantages over traditional jamming attacks. This chapter proposes DL-based media access and dynamic network configuration as two effective countermeasures against DeepJam.

Chapter 4 studies the scalable group-key establishment in large-scale mission-critical IoT networks. Devices in a mission-critical IoT network frequently exchange sensitive broadcast/multicast messages through the open wireless channel. A group key is thus needed to authenticate and encrypt these communications. Existing group-key generation schemes, however, are not scalable with the network size. Particularly, their communication and computation overhead increases dramatically as the the network size increases. This chapter proposes a scalable solution named DroneKey. DroneKey actively induces correlated changes to the wireless signals received at a group of devices with a flying drone. The DL technique is used to extract a common key from these related wireless signals. The real-world evaluation with a prototype of DroneKey confirms that DroneKey significantly outperforms existing solutions in terms of scalability and the key-generation rate.

Chapter 2

THE SECURITY AND USABILITY OF ACOUSTIC MOBILE AUTHENTICATION

2.1 Overview

Acoustic fingerprinting aims to identify a mobile device by its internal microphone(s) and speaker(s). It is promising for two primary reasons. First, a typical mobile device, such as the smartphone and smartwatch, has at least one microphone and one speaker. Second, every microphone or speaker is a multi-stage audio signal processing system consisting of multiple hardware elements, so it can be quite unique due to the hardware imperfection introduced in the manufacturing process.

Different acoustic fingerprints have been explored. The Frequency Response Curve (FRC), which is a linear feature of a speak or microphone and refers to the normalized output gains over a given frequency range, was used in (Z. Zhou et al. 2014; Hristo et al. 2014; D. Chen et al. 2017). Das *et al.* used Mel-Frequency Cepstral Coefficients (MFCCs) of the output audio to identify a speaker or microphone (Das, Borisov, and Caesar 2014). NAuth (X. Zhou et al. 2019) distinguishes different speaker-microphone pairs with a nonlinear feature called Acoustic Nonlinear Pattern (ANP). The hardware features of a device's speaker, microphone, or speaker-microphone pair can be used as the acoustic fingerprint, and I term the corresponding fingerprints as S-Print, M-Print, and SM-Print, respectively.

Mobile authentication is one of the most appealing application scenarios of acoustic fingerprinting. A mobile authentication system considers a user as legal if he[1] can prove the possession of a registered mobile device. Fig. 1 shows a generic acoustic mobile authentication system. It consists of three parties: the prover $\mathcal{P}$ (the registered mobile device of the user), the verifier $\mathcal{V}$, and the server $\mathcal{S}$. Without ambiguity, I also denote the user owning the prover device by $\mathcal{P}$. $\mathcal{P}$ starts an authentication instance by sending a request with $\mathcal{P}$'s ID to $\mathcal{S}$. Then $\mathcal{S}$ sends a challenge to $\mathcal{P}$ via $\mathcal{V}$, and $\mathcal{P}$ returns a response corresponding to the challenge. $\mathcal{P}$ is authenticated if $\mathcal{S}$ verifies that the response is associated with one of the registered devices, and vice versa. The mobile device can act as both $\mathcal{P}$ and $\mathcal{V}$ in self-proof scenarios like the online account login on a mobile device, in which case $\mathcal{S}$ directly communicates with the mobile device.

Acoustic fingerprinting has been applied to mobile authentication in existing studies (D. Chen et al. 2017; X. Zhou et al. 2019), but there are still two open questions: 1) which acoustic fingerprint technique is most suitable for mobile authentication; 2) the fingerprint of which acoustic element(s) (the speaker, microphone, or speaker-microphone pair) should be used. To answer these two questions, the author identifies the following three essential requirements for a sound acoustic mobile authentication system.

- **Accurate**: the system can accurately identify mobile devices.
- **Deployable**: it is low-cost and can extract *verifier-agnostic* acoustic fingerprints. In particular, the fingerprint of a mobile device should not be tied to a specific verifier, which is very important in a large distributed system with many verifiers.

[1]No gender implication.

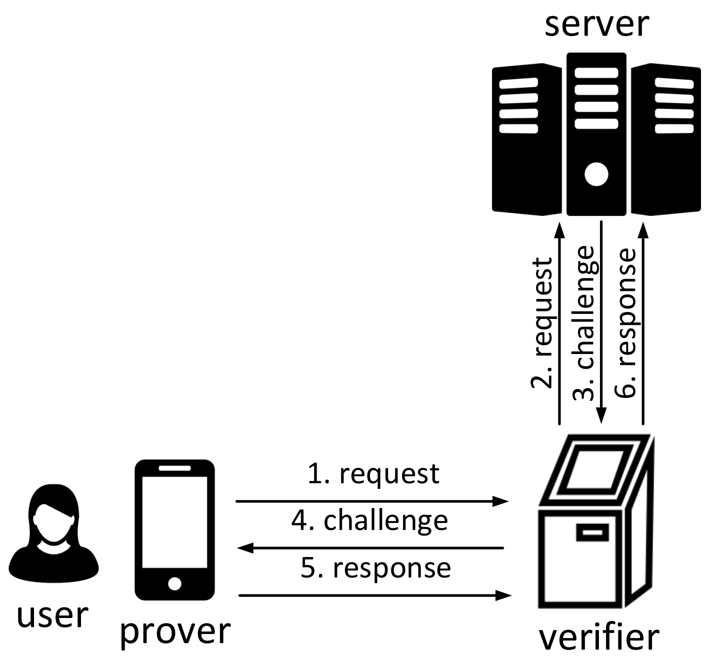


Figure 1. A generic mobile authentication system.

- **Secure**: it is highly resilient to possible attacks.

To extract a verifier-agnostic MFCC fingerprint of a prover, the verifier must be equipped with a high-fidelity speaker or microphone which usually costs a few hundred dollars or more. Since MFCC fingerprints do not satisfy the deployable requirement, this chapter focuses on studying FRC and ANP fingerprints henceforth.

2.2 System and Adversary Models

2.2.1 A Generic Acoustic Mobile Authentication Model

A generic acoustic mobile authentication system consists of the prover $\mathcal{P}$, the verifier $\mathcal{V}$, and the server $\mathcal{S}$, as shown in Fig. 1. $\mathcal{P}$ is the user's mobile device and is registered to the system in the initialization stage. The registration can only be conducted when the communication between $\mathcal{P}$ and $\mathcal{S}$ is guaranteed to be secure, i.e., when the attacker can neither overhear nor tamper with the communication. To register a mobile device, the user first sends $\mathcal{S}$ a registration request which contains an identification proof such as the username and password. $\mathcal{S}$ verifies the request and returns a challenge, and the user generates a response corresponding to the challenge with $\mathcal{P}$ and submits the response. $\mathcal{S}$ extracts $\mathcal{P}$'s acoustic fingerprint from the response and stores the fingerprint for future verification.

The author makes the following assumptions for the acoustic mobile authentication system. First, $\mathcal{V}$ can communicate with $\mathcal{S}$ through a secure wireless or wired channel. Second, $\mathcal{P}$ and $\mathcal{V}$ can communicate via a short-range wireless channel (e.g., Bluetooth, WiFi, or acoustic channels) which is not necessarily secure.

The system can identify $\mathcal{P}$ with the acoustic fingerprint of its speaker, microphone, or speaker-microphone pair which is termed as S-Print, M-Print, or SM-Print of $\mathcal{P}$ using different challenges and responses. If S-Print is used, the challenge specifies the input to $\mathcal{P}$'s speaker whose output audio is the response. $\mathcal{V}$ records the audio with a built-in microphone and forwards the response to $\mathcal{S}$. If $\mathcal{P}$ is identified with its M-Print, the challenge is an audio generated by $\mathcal{V}$'s speaker. $\mathcal{P}$ records the challenge audio with the its microphone and submits the recorded audio as the response to $\mathcal{S}$. Finally, SM-Print involves $\mathcal{P}$'s speakers and microphones. The challenge specifies the input to $\mathcal{P}$'s speaker, and $\mathcal{P}$ records the output audio with its microphone as response. In addition, the response in S-Print is an audio, and those in M-Print and SM-Print are audio files. To clarify the difference, the author terms the response in S-Print as an A-response and that in M-Print or SM-Print as an F-response.

S-Print, M-Print, and SM-Print target different authentication scenarios. S-Print and M-Print are suitable for proximity-based authentication systems in which a stand-alone verifier is available to verify the proximity of $\mathcal{P}$ to $\mathcal{V}$. The verifier can be a smart lock in an access control system or a login terminal such as a laptop with which the user tries to log into his online account. SM-Print is suitable for self-proof authentication scenarios in which $\mathcal{P}$ directly communicates with $\mathcal{S}$. For example, a user may log into his online account on the mobile device which is also used as his prover. In this case, the challenge audio specified by $\mathcal{S}$ is played by $\mathcal{P}$'s speaker and recorded by $\mathcal{P}$'s microphone.

2.2.2 Adversary Model

This chapter considers an attacker $\mathcal{A}$ who attempts to be authenticated as $\mathcal{P}$ by the system. The author has the following assumptions about $\mathcal{A}$: 1) $\mathcal{A}$ has no access to $\mathcal{P}$ and cannot compromise $\mathcal{P}$, $\mathcal{S}$, or $\mathcal{V}$; 2) $\mathcal{A}$ is aware of the used fingerprinting scheme and has acquired some fingerprint(s) of $\mathcal{P}$; 3) $\mathcal{A}$ can launch the attack with advanced equipment like high-fidelity speakers and microphones. This work focuses on the security of exploring acoustic fingerprints for mobile authentication, and other security mechanisms such as encryption and biometric-based verification are beyond the scope of this report. The author thus considers the authentication system compromised if $\mathcal{A}$ can bypass the acoustic-fingerprint verification.

$\mathcal{A}$ may obtain $\mathcal{P}$'s acoustic fingerprints through three practical ways. First, $\mathcal{A}$ can overhear the communication between $\mathcal{P}$ and $\mathcal{V}$ if the channel between them is insecure. For example, the prover in an S-Print authentication system transmits the response audio to $\mathcal{V}$ through the insecure acoustic channel, and the audio can be captured by any nearby microphones. $\mathcal{A}$ can thus obtain the response audio by deploying a microphone around $\mathcal{V}$ and then use it to infer $\mathcal{P}$'s acoustic fingerprint. In M-Print and SM-Print systems, $\mathcal{P}$ may communicate with $\mathcal{V}$ through Wi-Fi or Bluetooth, which are more secure than the acoustic channel. However, $\mathcal{A}$ still gets a chance to obtain the communication content by launching some advanced attacks such as the Man-in-the-Middle attack proposed in (Chen and Wu 2010). Second, $\mathcal{A}$ can deploy a phishing website or application which also adopts acoustic authentication and requires the user to reveal $\mathcal{P}$'s acoustic fingerprints. Finally, $\mathcal{S}$ must store $\mathcal{P}$'s fingerprint for verification which may be exposed to $\mathcal{A}$ due to data leakage. The author considers the following specific attacks in this reports.

- **Random Impersonation**. $\mathcal{A}$ impersonates $\mathcal{P}$ with his own mobile device $\hat{\mathcal{P}}$. $\mathcal{A}$ can obtain the model of $\mathcal{P}$ and uses a device of the same model to launch the attack.
- **Replay Attack**. $\mathcal{A}$ manages to obtain $\mathcal{P}$'s response and replays it to the system. In particular, $\mathcal{A}$ starts an authentication instance with his own device $\hat{\mathcal{P}}$ and sends a request containing $\mathcal{P}$'s ID to $\mathcal{S}$. When being asked for a response, $\mathcal{A}$ submits $\mathcal{P}$'s response to $\mathcal{S}$. An F-response, which is an audio file, can be directly submitted through the short range wireless channel between $\hat{\mathcal{P}}$ and $\mathcal{V}$. To submit an A-response, $\mathcal{A}$ plays the response audio to $\mathcal{V}$ with $\hat{\mathcal{P}}$'s speaker.
- **Co-located Attack**. $\mathcal{A}$ starts an authentication instance at a verifier that is close to $\mathcal{P}$. When being asked for the response, $\mathcal{A}$ triggers $\mathcal{P}$ to generate and submit the response, which is used for verification.
- **Fingerprint-Emulation Attack**.$\mathcal{A}$ manages to obtain one fingerprint of $\mathcal{P}$ and then emulates it with his own mobile device. Specifically, $\mathcal{A}$ first starts an authentication instance with his own mobile device and then submits a forged

response corresponding to the challenge to $\mathcal{S}$ for verification. If the target authentication system uses M-Print or SM-Print for verification, $\mathcal{A}$ can submit the forged F-response through the short-range wireless channel between $\mathcal{A}$'s mobile device and $\mathcal{V}$. If the target authentication system adopts S-Print, $\mathcal{A}$ plays the forged A-response with a high-fidelity speaker which does not distort the forged A-response. By perfectly emulating one fingerprint of the prover $\mathcal{P}$, $\mathcal{A}$ can be authenticated just as $\mathcal{P}$.

2.3 FRC-based Mobile Authentication: Proximity-Proof

This section adopts mobile two-factor authentication (2FA) as the example context to demonstrate FRC-based mobile authentication. Specifically, the author first proposes Proximity-Proof, an automatic 2FA system that adopts the FRC-based mobile authentication as the extra layer of security (Han et al. 2018) and then evaluate the security and usability of Proximity-Proof. With comprehensive evaluations, the author identifies the benefits and limitations of the FRC-based mobile authetnication.

Mobile 2FA adds the user's smartphone or other mobile devices as the second layer of security to secure online accounts, as passwords are increasingly easy to steal, guess, or hack (**RSASec12**; **RSABreach11**). When a user tries to log into an online system employing mobile 2FA, he enters username and password as usual. Then the online system will verify whether the user have the pre-registered mobile device and let him in if so. So mobile 2FA lets the user's mobile device serve as another proof of his identity and can keep the account safe even if the password is compromised.

Commercial mobile 2FA solutions such as Google 2-step verification (**Google2**), Duo (**Duo**), and Encap Security all require user involvement. For example, a Duo user needs to enroll his phone and install the Duo Mobile app there. There are three authentication methods for the online system to verify the user's possession of the enrolled phone. First, the system can send a notification (called Duo Push) that the user needs to approve in Duo Mobile. Second, the system can call the enrolled phone for the user to answer and press a key to approve the login. Finally, the user can enter a passcode on the login interface, which can be texted to the enrolled phone by the system or generated in Duo Mobile. Other mobile 2FA solutions all adopt similar authentication methods. Such demand for user interactions seriously affects the experience of mobile users (Weir et al. 2009; Gunson et al. 2011), especially senior citizens or those with disability such as blind and visually impaired users.

Proximity-Proof is motivated by the observation that the user response in each aforementioned mobile 2FA technique is equivalent to transmitting some information either directly or indirectly via the login device to the online system. The author refers to such user information as the 2FA response for convenience, which is the passcode in the third Duo authentication method or some unforgeable data incurred by the legitimate user's approval of the login attempt in the first and second Duo authentication methods. Zero user-phone interaction can thus be achieved by automatically generating and then transmitting the 2FA response to the server. The workflow of Proximity-Proof is shown in Fig. 2. The author assumes a general scenario in which a web server processes login requests via a browser-based interface. The server-browser communications are secured with traditional TLS-like mechanisms. Each legitimate user enrolls his phone and also install the Proximity-Proof app. In this scenario, the server $\mathcal{S}$ is the web server, the verifier $\mathcal{V}$ is the login device, and the prover $\mathcal{P}$ is still the user's phone.

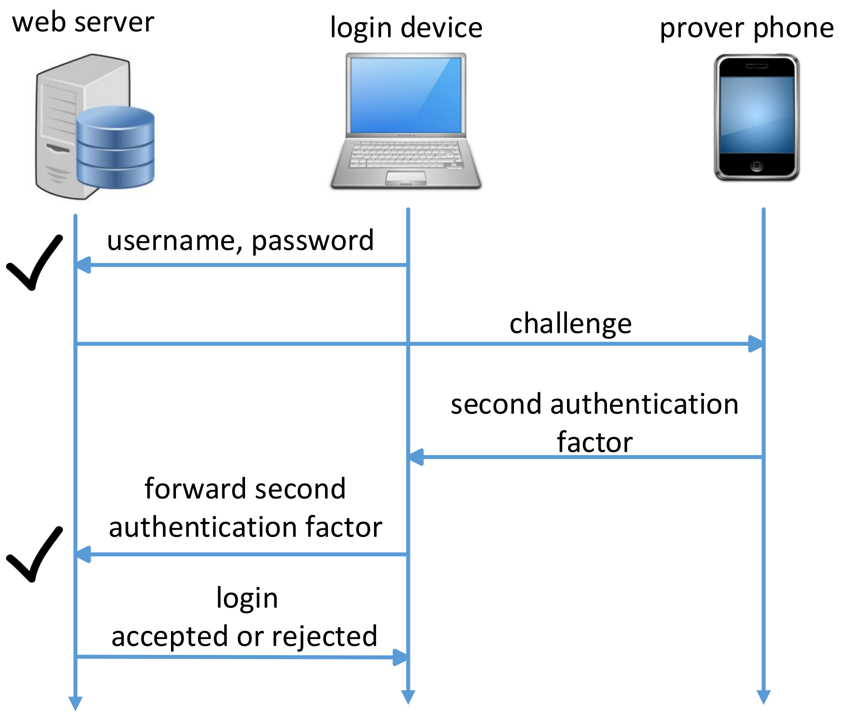

web server
login device
prover phone
username, password
challenge
second authentication factor
forward second authentication factor
login accepted or rejected

Figure 2. The workflow of Proximity-Proof.

Proximity-Proof leverages the prevalent acoustic components for 2FA response transmission. FRC fingerprinting is enabled to counteract the replay attack, and acoustic distance ranging method is adopted to defeat co-located attackers. Specifically, the web server stores the FRC fingerprint of the enrolled phone's microphone and speaker. After verifying the 2FA response from a mobile device, which can also be termed as prover, the login browser further involves a novel protocol developed by us to extract the speaker fingerprint and microphone fingerprint of the prover. If the extracted fingerprints match the stored copies, the web server considers that the 2FA response was generated by the enrolled phone. While extracting the speaker and microphone fingerprints of the prover, the browser further measures the distance to the prover by exchanging a few acoustic signals. If the estimated distance is above a user-chosen safety threshold, the browser considers that the co-located attack may have happened.

The web server only admits the attempted user when the 2FA response, the speaker and microphone fingerprints, and the distance measurement all pass verifications. Otherwise, it invokes the traditional mobile 2FA process as the fallback.

2.3.1 Attacks Against Proximity-Proof

This part discusses the attacks against Proximity-Proof and still considers the four types of attacks introduced in Section 2.2.2. The random impersonation and fingerprint-emulation attacks against Proximity-Proof are the same as demonstrated previously, so the author only details the replay attack and co-located attacks in this section. The author considers a more advanced replay attack against Proximity-Proof, which is termed as the Man-in-the-Middle (MiM) attack.

- **MiM attack**: Fig. 3a illustrates the MiM attack, in which the attacker $\mathcal{A}$ is far from the victim and his enrolled phone. But $\mathcal{A}$ sets up a high-speed, invisible channel between the enrolled phone and the adversarial login device, e.g., by having an accomplice or hidden eavesdropping device near the victim. When $\mathcal{A}$ attempts to log in, the web server triggers the enrolled phone to generate an automatic 2FA response which is relayed in real time to the login device via the adversarial channel.
- **Co-located attack**: As shown in Fig. 3b, $\mathcal{A}$ in this scenario is physically co-located with the victim such as in a library, a bar, a train, a campus cafeteria, or other often crowded public venues. $\mathcal{A}$'s attempted login again triggers an

automatic response from the enrolled phone, which can be directly received by $\mathcal{A}$'s login device.

The server considers the enrolled phone near the login device and then admits $\mathcal{A}$ by mistake under both MiM and co-located attacks. As mentioned in (Karapanos et al. 2015), the prior work (Shirvanian et al. 2014; Czeski et al. 2012; Karapanos et al. 2015) cannot deal with MiM and co-located attacks. In contrast, Proximity-Proof is designed to thwart them.

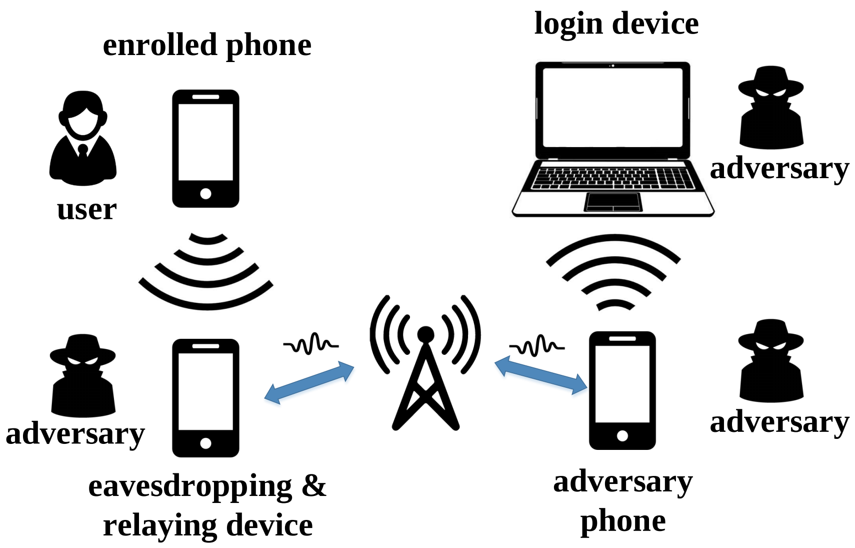
2.3.2 Proximity-Proof Design

In this section, the author details Proximity-Proof's key components: acoustic transmission, FRC fingerprinting, and cross-device ranging.

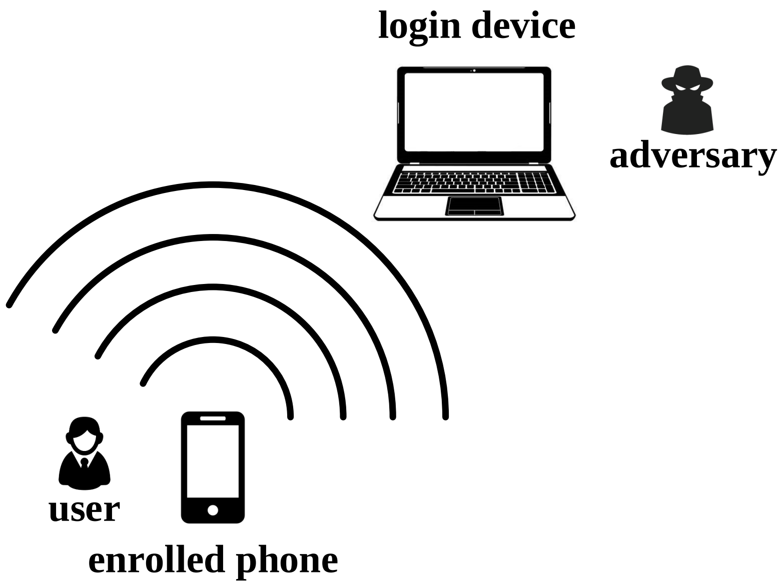
2.3.2.1 Acoustic Transmission of 2FA Response

Proximity-Proof transmits the 2FA response via acoustic signals emitted by the enrolled phone's speaker and received by the login device's microphone. Note that web browsers can access the host device's speaker and microphone via the standard Web Audio API. The author uses OFDM-based acoustic signals to cope with severe channel conditions.

Proximity-Proof uses high-frequency inaudible signals to avoid disturbing users and also explore the fact that the high-frequency band is usually very quiet according to the prior work (Z. Zhou et al. 2014). My implementation and experiments use the frequency band between 18 kHz and 20 kHz, which is thus used in my subsequent illustrations as



(a) MiM attack.



(b) Co-located attack.

Figure 3. Attacks against Proximity-Proof.

an example. Proximity-Proof divides [18, 20] kHz into 20 non-overlapping sub-channels with each spanning 100 Hz. The OFDM sub-carrier frequencies are $f_m = 18 + 0.1m$ kHz for $m \in [1, 20]$. As in (Wang et al. 2016), Proximity-Proof uses On-Off Keying as the modulation scheme for its simplicity, and the phone generates the n-th ($n \geq 1$) time-domain sample (Nandakumar et al. 2016) as

$$x_n = A \sum_{m=1}^{20} X_m \cos(2\pi n f_m) \ , \tag{2.1}$$

where A denotes the signal amplitude, and X_m is the m-th binary bit to transmit. x_n is sent via the phone speaker.

After receiving x_n via its microphone, the browser performs a Fast Fourier transform (FFT) to extract the amplitude of each sub-carrier signal component, denoted by I_m for sub-carrier f_m. Since no signal is transmitted at 18 kHZ, the author denotes the signal amplitude detected at 18 kHz by I_0 and use it as a reference. The browser then decodes X_m by comparing I_m with I_0. If the difference between I_m and I_0 exceeds a predefined system threshold (e.g., 10 dB in my experiments), X_m is decoded as bit-1 and otherwise bit-0.

The author constructs a virtual packet from the 2FA response, which consists of a preamble followed by data segments. The preamble is to help the login browser locate the beginning of the virtual packet. Similar to (Wang et al. 2016), the author uses a chirp signal (20ms long in my experiments) from 17 kHz to 19 kHz as the preamble. A silence period (20ms in my experiments) is also added after the preamble to avoid interference with the following data segment. The author also applies the Reed-Solomon code RS(15,11) to encode the raw 2FA response to mitigate transmission errors. The RS-coded 2FA response is further divided into data segments of 20 bits with one for each OFDM sub-carrier. Each data segment is converted into an OFDM symbol of duration 10ms, and a silence period of 10ms is added between adjacent

OFDM symbols to combat the inter-symbol-interference (ISI) and the multipath effect. The author found in my experiments that the audio is initially heavily distorted, so the author lets the speaker send a random audio signal of 20ms long before the preamble to "warm up" itself.

The performance of my 2FA transmission scheme above can be briefly analyzed as follows. Assume that the RS-coded 2FA response is L bits, where L is an integer multiple of 20 after possible padding. It takes $20+20+20+10*L/20+10*(L/20-1) = (50+L)$ms to transmit one virtual packet, corresponding to an effective data rate of $\frac{L}{50+L}$ kb/s. Suppose that the virtual packet can be successfully decoded with probability p. The phone speaker keeps sending the virtual packet for $m \geq 1$ times, where m is a system parameter. If the login browser still cannot successfully decode a virtual packet with probability $(1-p)^m$, it notifies the the authorb server to invoke the traditional mobile 2FA authentication method.

2.3.2.2 FRC Fingerprinting

Now the author presents a novel technique for the login browser to extract the speaker and microphone fingerprints of a mobile device which purports to be the enrolled phone.

The feasibility of speaker and microphone FRC fingerprinting is rooted in the imperfect manufacturing process that introduces unique mechanical and electronic features into each speaker (or microphone). So each speaker (or microphone) has a unique frequency response which measures the gain or attenuation at each frequency and can identify the affiliated mobile device. The prior work (Z. Zhou et al. 2014; Das, Borisov, and Caesar 2014) explores the frequency response as a hardware fingerprint

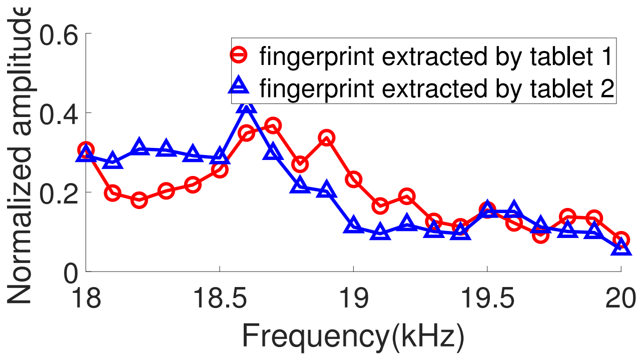


Figure 4. Frequency response curves of the speaker on a Samsung Galaxy S5, measured by two Nexus 7.

to identify a smartphone, but the extracted frequency response is associated with a speaker-microphone pair (i.e., the emitting speaker and the recording microphone) rather than with an individual speaker or microphone. The author highlights this issue with a simple experiment. Fig.4 shows the frequency responses of the speaker on a Samsung Galaxy S5 smartphone, measured by two Nexus 7 tablets with the same method in (Z. Zhou et al. 2014; Das, Borisov, and Caesar 2014). As I can see, the two microphones yield very different frequency responses for the same speaker.

In the mobile 2FA context, the speaker is on the enrolled phone, while the microphone is on an arbitrary login device available to the user (e.g., a personal computer or a shared one in a library). If I use the same method in (Z. Zhou et al. 2014; Das, Borisov, and Caesar 2014) to identify the enrolled phone, the extracted frequency response is tied to the speaker of the enrolled phone and the microphone of a particular login device. It follows that the online system must obtain the frequency response associated with the enrolled phone and every possible login device the user may use in the enrollment phase, which is highly unrealistic. So the prior work (Z. Zhou et al. 2014; Das, Borisov, and Caesar 2014) is not applicable to my context.

my fingerprinting technique explores the following acoustic propagation model for frequency f proposed in (Szabo 1994) and then refined in (D. Chen et al. 2017),

$$P(f,x) = L(f)L'(f)P_0(f)e^{\lambda(x)} + \text{noise} \tag{2.2}$$

where $P_0(f)$ represents the transmitted signal power, $P(f,x)$ denotes the received signal power at distance x from the speaker, $L(f)$ and $L'(f)$ denote the energy loss due to the speaker and microphone, respectively, and $\lambda(x)$ is a function of x that can be obtained by fitting experimental data.

The above propagation model can be further simplified. In particular, I have observed from my experiments that the ambient noise is insignificant at any frequency beyond 18 kHZ. I further conducted an experiment to evaluate the SNR in a noisy coffee house. I set the volume of a Samsung Galaxy S5 to 30 percent of its maximum volume and used a flat stimulation (to be explained shortly) as the input to its speaker. I used the other Samsung Galaxy S5, which was placed half a meter away (the expected maximum safe working distance of Proximity-Proof), to record the audio. I found that the received audio signal power is more than 20 dB higher than the ambient noise. To minimize the impact of noise, I the leverage AudioManager API to set the volume to the maximum.

I can obtain a refined acoustic propagation model as

$$P(f,x) \approx L(f)L'(f)P_0(f)e^{\lambda(x)}\ . \tag{2.3}$$

Proximity-Proof explores an interactive protocol for the login browser to extract the speaker and microphone fingerprints of the prover phone. My protocol uses a flat stimulation as the input to the speakers of both the prover phone and login device. The flat stimulation is composed of 20 sine waves whose frequencies range from 18.1 kHz to 20 kHz in an equal increase of 0.1 kHz. In particular, the speaker of the prover phone generates an audio to the flat stimulation, which is recorded by the microphones on both the prover phone and the login device; then the speaker of the login device generates an audio to the flat stimulation, which is recorded by the microphones on both the prover phone and the login device as well. Let D denote the prover phone and B the login device. I also use $P_{XY}(f)$ to denote the received power at frequency f of the audio signal emitted by X and recorded by Y, where X and Y can be either of B and D. Then I have the following equations

$$P_{DD}(f) = L_D(f)L'_D(f)P_D(f)e^{\lambda(x_{DD})}, \tag{2.4}$$

$$P_{DB}(f) = L_D(f)L'_B(f)P_D(f)e^{\lambda(x_{DB})}, \tag{2.5}$$

$$P_{BB}(f) = L_B(f)L'_B(f)P_B(f)e^{\lambda(x_{BB})}, \tag{2.6}$$

$$P_{BD}(f) = L_B(f)L'_D(f)P_B(f)e^{\lambda(x_{BD})}, \tag{2.7}$$

where P_X is the transmission power at frequency f on device X, and x_{XY} denote the distance between the speaker of device X and the microphone of device Y.

Each enrolled phone can be uniquely identified by a vector of $L_D(f)$ and $L'_D(f)$ values for each frequency f in the flat stimulation. Directly obtaining $L_D(f)$ and $L'_D(f)$ involves estimating $P_D(f)$, $P_B(f)$, x_{DD}, x_{DB}, x_{BB}, and x_{BD}. I use a special trick to avoid the error-prone parameter estimation. Let the signal measurements at a reference frequency 18 kHz be denoted by R_{DD}, R_{DB}, R_{BB}, and R_{BD}, respectively. I further use l_X and l'_X to denote the energy loss of the speaker and microphone of device X at 18 kHz, respectively. Then I have

$$R_{DD} = l_D l'_D P_D e^{f(x_{DD})}, \tag{2.8}$$

$$R_{DB} = l_D l'_B P_D e^{f(x_{DB})}, \tag{2.9}$$

$$R_{BB} = l_B l'_B P_B e^{f(x_{BB})}, \tag{2.10}$$

$$R_{BD} = l_B L'_l P_B e^{f(x_{BD})}. \tag{2.11}$$

By combining Equations (2.4) to (2.11), I have

$$P_{DD}(f)/R_{DD} = (L_D(f)/l_D)(L'_D(f)/l'_D), \tag{2.12}$$

$$P_{DB}(f)/R_{DB} = (L_D(f)/l_D)(L'_B(f)/l'_B), \tag{2.13}$$

$$P_{BB}(f)/R_{BB} = (L_B(f)/l_B)(L'_B(f)/l'_B), \tag{2.14}$$

$$P_{BD}(f)/R_{BD} = (L_B(f)/l_B)(L'_D(f)/l'_D). \tag{2.15}$$

The prover phone needs to report its signal measurements $P_{DD}(f)$, $P_{BD}(f)$, $R_{DD}(f)$, and $R_{BD}(f)$ to the login browser. By solving these equations, the login browser can get $S_i(f) = L_D(f)/l_D$ and $M_i(f) = L'_D(f)/l'_D$, based on which to obtain two 20-dimension vectors, denoted by S and M for the prover phone's speaker and microphone, respectively. Then I normalize S and M as

$$\hat{S} = \frac{S}{\sqrt{\sum_{f\in\{18.1,18.2,\ldots,20\}\text{kHz}} S_i^2(f)}}, \tag{2.16}$$

$$\hat{M} = \frac{M}{\sqrt{\sum_{f\in\{18.1,18.2,\ldots,20\}\text{kHz}} M_i^2(f)}},. \tag{2.17}$$

The above fingerprinting process can be executed multiple times to improve estimate accuracy, in which case the login browser uses the concatenation of average $\hat{S}$ and $\hat{M}$ as the acoustic fingerprint of the prover phone. If the Euclidean distance between the collected and legitimate acoustic fingerprints is above a threshold τ, the prover phone is considered an imposter and rejected access.

I set the threshold $\tau = 0.4$ in Proximity-Proof, which was obtained through experiments. In particular, I used one Samsung tablet as the login terminal, one Samsung S5 as the prover device, and five other devices as adversarial devices, including one Samsung S5, one Samsung Note 5, one Huawei Honor 8, and two Google Nexus 6. For each mobile device, I extracted its speaker and microphone fingerprints 20 times. I chose 20 values, ranging from 0.1 to 2 with a step of 0.1, as candidate threshold

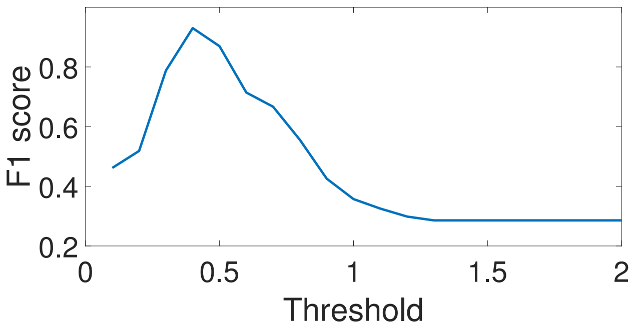


Figure 5. F_1 scores for different τ.

values. Then I used F-measurement to evaluate each value, and the F_1 scores were calculated using the following equations.

$$F_1 = \frac{2}{\dfrac{1}{Recall} + \dfrac{1}{Precision}} \tag{2.18}$$

$$Precision = \frac{TP}{TP + FP} \tag{2.19}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.20}$$

where TP and TF are the numbers of correctly recognized fingerprints of the prover device and malicious devices, respectively; FN and FP are the numbers of incorrectly recognized fingerprints of the prover device and malicious devices, respectively.

The F_1 score is an important metric to evaluate the accuracy of the binary classification method. A high F_1 score ensures that both precision and recall are high. The result demonstrated in Fig. 5 shows that $\tau = 0.4$ achieves the highest F_1 score. Therefore, I adopt 0.4 as the threshold in my experiments. In practice, the parameter τ can be further refined with more sophisticated machine learning algorithms and much more mobile devices.

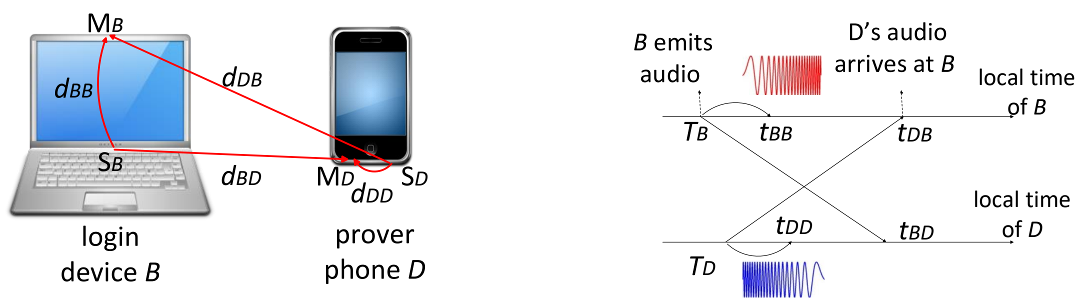


Figure 6. Illustration of two-way acoustic ranging.

2.3.2.3 Cross-Device Ranging

Proximity-Proof estimates the distance between the prover phone and the login device to withstand the co-located attack. The key motivation is that a user normally puts his phone closer to himself than anyone else in a crowded public environment (e.g., a library or cafeteria) where the co-located attack is more likely to occur. So the distance between the enrolled phone and login device of the co-located attacker should be sufficiently larger than that between the enrolled phone and login device used by the legitimate user.

There are many cross-device ranging methods. For example, Frequency Modulated Continuous Waveform (FMCW) has been used to accurately measure the distance between two synchronized devices (Kulpa 2006; Derham et al. 2007). However, cross-device synchronization is non-trivial (Wei and Zhang 2015; Wang and Shao 2013). Even a small synchronization deviation of 1ms will lead to a measurement error of 30cm. A FMCW variant is presented in (Mao, He, and Qiu 2016) and does not require cross-device synchronization; but this method is designed for devices equipped with at least two speakers, which are not available on many COTS phones and tablets.

Proximity-Proof adopts the two-way sensing method in (Peng et al. 2007) to measure the distance between two devices. Without the need for cross-device synchronization, this method only requires that both devices have one speaker and one microphone. Almost all COTS smartphones, tablets, laptops, and all-in-one PCs fulfill this requirement. Fig. 6 shows the process of the two-way ranging method for clarity. Here I assume that device D is the prover phone with microphone M_D and speaker S_D, and device B is the login device with microphone M_B and speaker S_B.

The ranging process involving B and D both transmitting and recording audio signals. Specifically, B sends short audios via S_B at time T_B, and so does the prover phone D via S_D at time T_D. Meanwhile, both M_B and M_D start audio recording. Then B analyzes the recorded audio to derive the arrival time of its own audio and D's audio, denoted by t_{BB} and t_{DB}, respectively. Similarly, D derives t_{BD} and t_{DD}. I further donate the speed of sound by c and the distance between device X's speaker and device Y's microphone by d_{XY}. The following equations are straightforward to obtain,

$$d_{BB} = c \cdot (t_{BB} - T_B), \tag{2.21}$$

$$d_{BD} = c \cdot (t_{DB} - T_B), \tag{2.22}$$

$$d_{DB} = c \cdot (t_{BD} - T_D), \tag{2.23}$$

$$d_{DD} = c \cdot (t_{DD} - T_D). \tag{2.24}$$

The distance $\bar{d}_{BD}$ between B and D is approximately equal to the average of d_{BD} and d_{DB}.

$$\begin{aligned}D &= \frac{1}{2} \cdot (d_{BD} + d_{DB}) \\ &= \frac{c}{2} \cdot ((t_{DB} - T_B) + (t_{BD} - T_D)) \\ &= \frac{c}{2} \cdot ((t_{DB} - t_{DD} - t_{BB} + t_{BD}) + \\ &\quad (t_{BB} - T_B) + (t_{DD} - T_D)) \\ &= \frac{c}{2} \cdot ((t_{DB} - t_{DD}) - (t_{BB} - t_{BD})) + \\ &\quad \frac{1}{2} \cdot (d_{BB} + d_{DD}),\end{aligned}$$

where d_{BB} is the distance between S_B and M_B, and d_{DD} is the distance between S_D and M_D. The speaker-microphone distance is often fixed for a specific mobile device model and can be known by checking the hardware specification. If $\bar{d}_{BD}$ is within a user-chosen safe threshold (say, 0.5m in my evaluation), the login browser (device) can ascertain that no co-located attack is present with overwhelming probability.

I use chirp audio signals to address interference and overlap. In particular, B and D emit up-chirp and down-chirp signals, respectively. The high auto-correlation and low cross-correlation of down and up chirps allow both devices to distinguish the audios from each other. To detect the audio arrival time, each device calculates the correlation between recorded audio and reference chirp signals. The "peak" point indicates the accurate arrival time.

In Proximity-Proof, the ranging and fingerprint procedures are conducted simultaneously. The frequency of the chirp signals used for ranging is between 16.5 kHz and 17.5 kHz. The frequency of the fingerprinting audios is between 18 kHz and 20 kHz. I transmit the ranging and fingerprinting audios at the same time. In doing so, Proximity-Proof can verify whether the ranging audio is from the enrolled phone.

2.3.2.4 Self-Proof Case

In Proximity-Proof, the login device is assumed to be different from the enrolled phone. But it is also very common that people use the browsers on their enrolled phones to access online accounts. Proximity-Proof can be easily modified to become Self-Proof for accommodating this scenario. Self-Proof uses the same processes in Proximity-Proof for automatic 2FA response transmission. However, with only one speaker and one microphone available, I cannot extract their individual fingerprints with the previous fingerprinting method in Section 2.3.2.2. Instead, I resort to the existing method in (Z. Zhou et al. 2014; Das, Borisov, and Caesar 2014) to fingerprint the speaker-microphone pair in each enrolled phone. More specifically, I can use the flat stimulation as the input to the speaker and use the microphone to record the audio. The login browser forwards the frequency response extracted from the audio to the web server for comparison with the stored copy associated with the provided username and password. Any significant difference above a system threshold will deny the account access and invoke the traditional mobile 2FA procedure. Since a different fingerprinting process is used in Self-Proof, a co-located attacker cannot overhear the fingerprint of the enrolled phone, thus eliminating the need for acoustic distance ranging in this context.

2.3.3 Evaluation of Proximity-Proof

In this section, I experimentally evaluate the effectiveness and security of Proximity-Proof.

2.3.3.1 Implementation

I implemented a prototype of Proximity-Proof. Specifically, I used one Lenovo E420 laptop as the login device and another Lenovo E420 laptop as the server. I chose Google Chrome as the browser and wrote the browser-side implementation in HTML5. I used thenavigator.mediaDevices.getUserMedia API to access the microphone and record audios. I also used the $HTML\langle audio\rangle$ element to access the speaker and played a pre-record chirp audio file in the WAV format. No plugin was needed for the browser.

2.3.3.2 Efficacy of Acoustic Fingerprinting

I used experiments to verify the uniqueness of acoustic fingerprints. Nine mobile devices were used, including two Samsung Galaxy S5, two Google Nexus 6, two Nexus 7 tablets, one Huawei Honor 8, one iPhone SE, and one iPhone 5. I first chose a Samsung Galaxy 5 as the user's device and extracted its fingerprint with every other device. The extracted acoustic fingerprints are shown in Fig. 7. As I can see, the fingerprints of the same device extracted by different devices are very similar.

Next, I used a Nexus 7 tablet as the login device in the enrollment phase and each of the other eight devices as a testing device. With the Nexus 7, I extracted the acoustic fingerprint of each testing device, which emulates its fingerprint stored at the web server. Then for each testing device, I used each other testing device as an ad-hoc login device to extract its fingerprint 20 times in three months, resulting in 140 runtime fingerprints for each testing device. In my experiments, the distance between each testing device and each login device was randomly chosen between 10cm

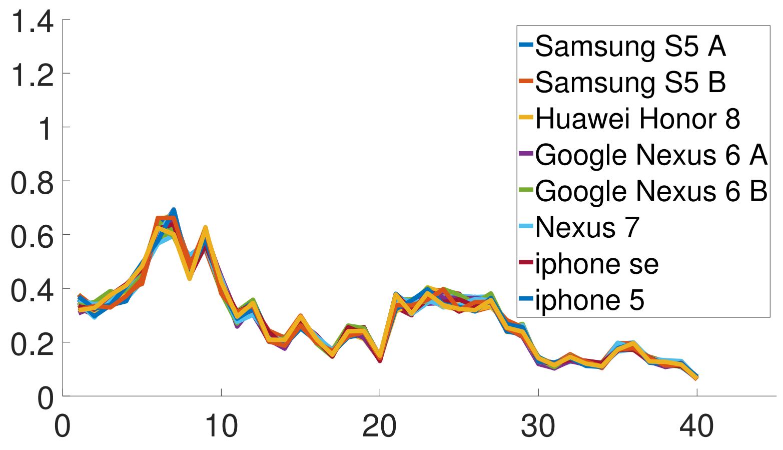


Figure 7. The acoustic fingerprints of a Samsung Galaxy 5 extracted by different devices.

to 50cm with arbitrary device orientation. The testing and login devices were place on the same table without any obstacle between them. Fig. 8 shows the Euclidean distance between each runtime fingerprint and its corresponding copy stored at the web server. In the box plot, the red bar inside each box depicts the median, and the lower and upper edges of the box are the first and third quartiles, respectively. The upper and lower ends of the whisker indicate the corresponding maximum and minimum values, respectively. Only three of the 1,120 runtime fingerprints are more than $\tau = 0.4$ away from the corresponding stored copies. This result further confirms that the web server can use any login device to extract the acoustic fingerprint of an enrolled phone. Besides, the fingerprint of each testing device does not change significantly in the three-month test window.

Fig.9 shows the Euclidean distance between the fingerprints of every two testing devices extracted by the initial Nexus 7 tablet. Since the distance is always larger than 0.4, acoustic fingerprints can effectively distinguish mobile devices of different types and also of the same type.

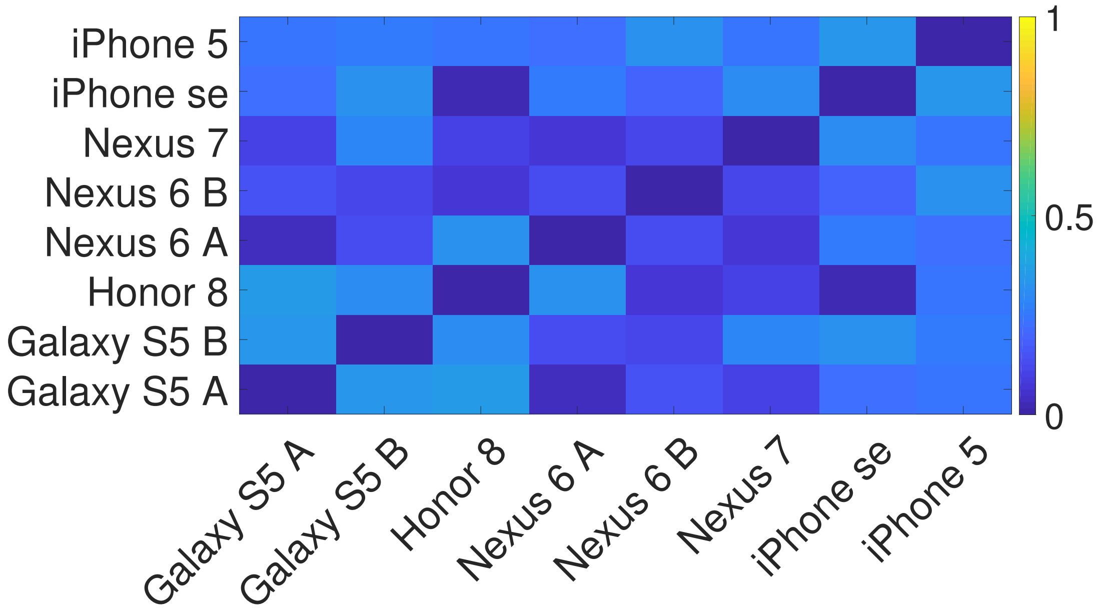


Figure 8. Distance between one device's acoustic fingerprints extracted by different login devices.

2.3.3.3 Security Against MiM attacks

I used one Samsung Galaxy S5 as the victim device and two Nexus 6 (one as the monitoring device and the other as the replay device) to conduct the MiM attack. The login device was a Nexus 7. However, apart from checking the one-time passcode, the login device also verified the fingerprint of the prover phone (i.e., the Nexus 7 acting as the replay device). Fig. 10 compares the real fingerprint of the victim device stored at the web server and the fingerprint extracted by the login browser. Since the later one is actually the fingerprint of the relaying Nexus 7, I can see the significant difference in Fig. 10, based on which the web server can easily deny the illegitimate login request.

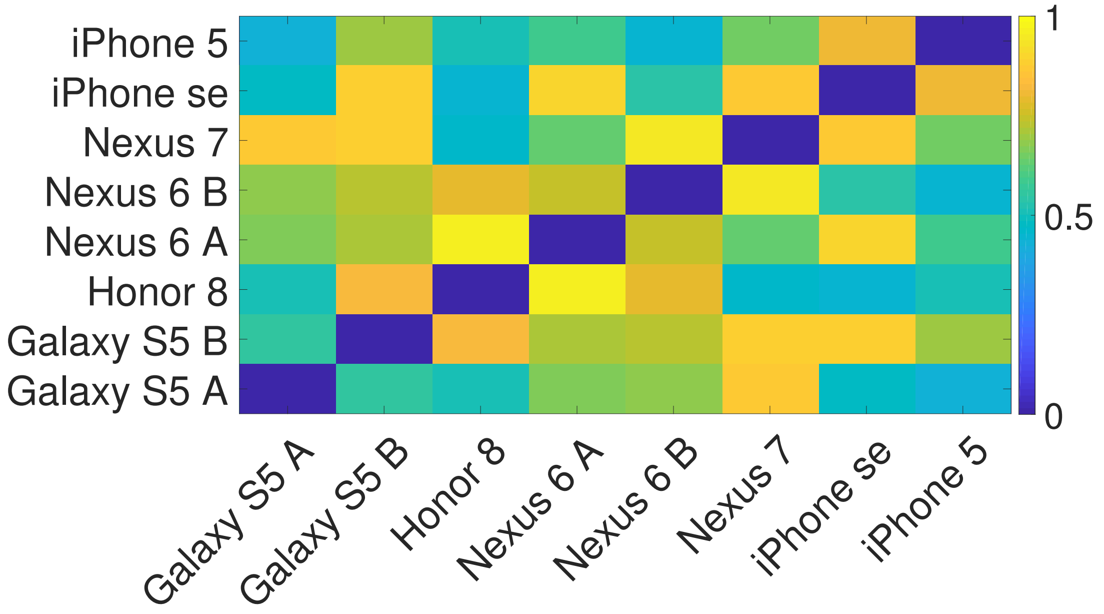


Figure 9. Distance between acoustic fingerprints of different devices.

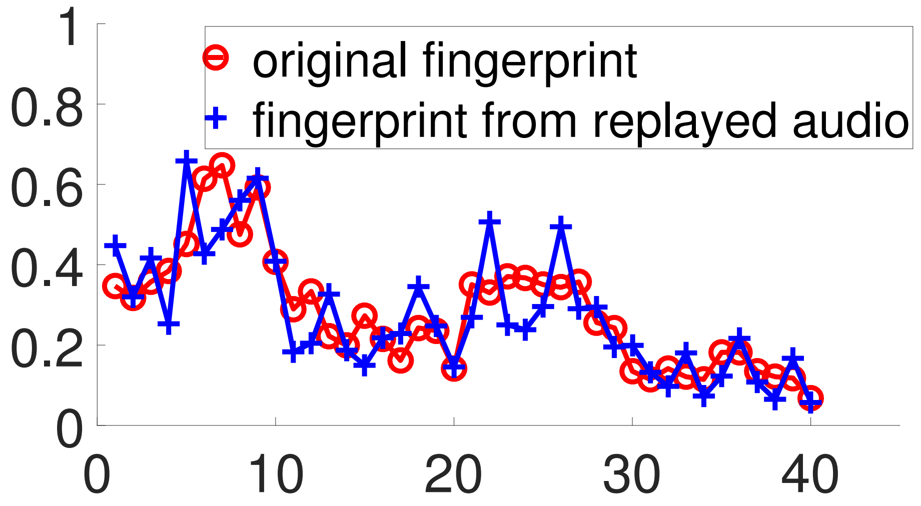


Figure 10. Resilience to MiM attacks.

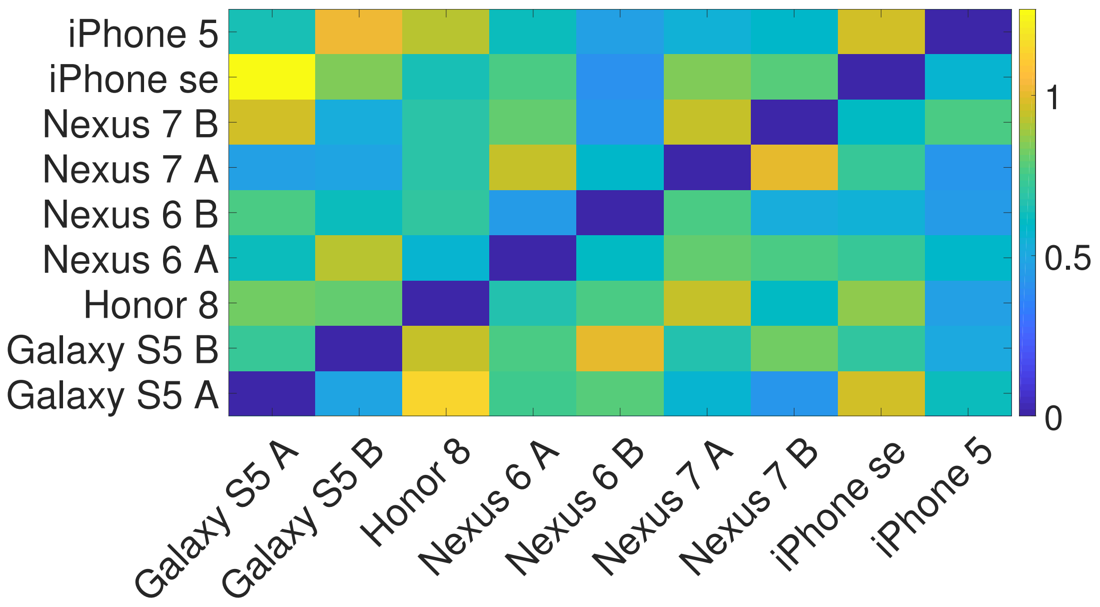


Figure 11. Distance between an original fingerprint and the fingerprint extracted from a replayed audio.

I further carried out the following experiment. For each pair of devices, say A and B, I used B to record the audio generated by A and replayed the audio to the login device. I then compared the fingerprint extracted from the replayed audio with the original fingerprint of device A. Note that I do not consider the fingerprint extracted from self-recorded audios because the attacker has no access to the user's device. As I can see from Fig. 11, the Euclidean distance is always larger than $\tau = 0.4$ for each pair of fingerprints, which indicate that Proximity-Proof can easily distinguish the original audio from the one replayed by illegitimate devices with a proper threshold τ. So Proximity-Proof can effectively defend against the MiM attack.

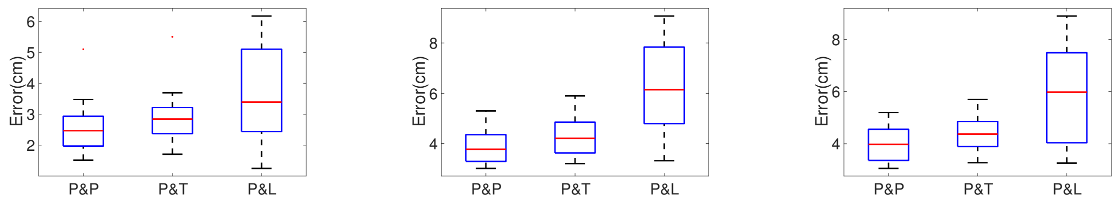
2.3.3.4 Security Against Co-Located Attacks

Now I report the accuracy of cross-device ranging and also the resilience of Proximity-Proof to co-located attacks.

Since Proximity-Proof is designed for different devices to work under diverse environments, I evaluated the accuracy of cross-device ranging in a wide range of scenarios. Specifically, I used the ranging method to measure the distance between a laptop-phone pair (L&P), a tablet-phone pair (T&P), and a phone-phone pair (P&P) in an office, a bookstore, and a coffee house. I used a Lenovo Thinkpad E420 as the laptop, a Nexus 7 as the tablet, and a Samsung Galaxy S5 as the phone. For each device pair in each environment, I set the ground-truth distance as 0.5m, which is Proximity-Proof's default maximum working distance. I then run the ranging method to measure their distance and calculate the distance errors for each case.

Fig. 12 shows the ranging errors in different environments, where the red points depict the outliers which fall more than 1.5 times the interquartile range above the third quartile or below the first quartile. I can see that the ranging accuracy for T&P and P&P is quite high with the average error in both cases below 5cm in all three environments. In contrast, the ranging accuracy for L&P is slightly lower with the average error around 4.2cm, 6.2cm, and 6.3cm in the office, bookstore, and coffee house, respectively. The reason is that the laptop's microphone is at the top of the screen, while its speaker is behind the keyboard. The distance between the laptop's speaker and microphone is affected by the screen-keyboard angle, which introduced additional errors into the ranging result in comparison with the other two cases.

I used a Lenovo E420 laptop as the login device and an Samsung Galaxy S5 as the user's device to evaluate Proximity-Proof's resilience to the co-located attack. The



(a) Office. (b) Bookstore. (c) Coffee house.

Figure 12. Ranging errors in different environments

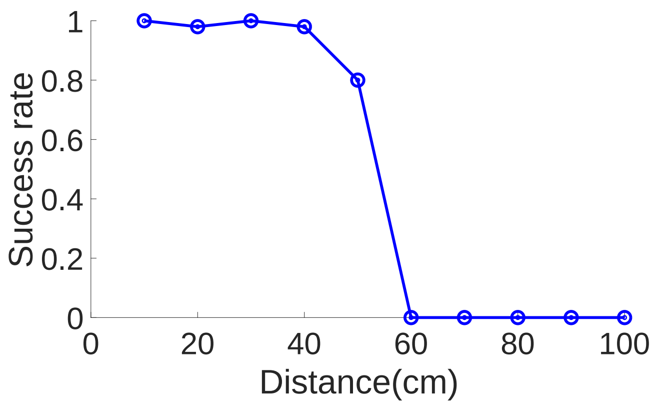


Figure 13. Success rates under different distance.

volume of the Galaxy S5 was set to 30 percent of its maximum volume. I varied the distance between the Galaxy S5 and the laptop from 10 cm to 1 m with a step length of 10cm and run the authentication procedure 50 times for each distance. As I can see from Fig. 13, when the distance is less than 40cm, the authentication attempt succeeds for at least 98% of the cases. When the distance is 50 cm, the successful authentication rate drops to around 80%, which is mainly caused by the ranging error. Moreover, if an attacker launches the co-located attack from a distance of 60cm or larger from the login device, almost none of his authentication attempts can succeed. These results show that Proximity-Proof is highly secure against the co-located attack.

2.3.3.5 Security Against Fingerprint-Emulation Attack

I conduct the fingerprint-emulation attack as following. The nine mobile devices were selected as the victim device one by one. For a chosen victim, I first used it to generate the benign response. The fingerprint extracted from the benign response was used as the fingerprint profile stored in the server. Then I inferred the response spectrum and used the ultrasound speaker to forge the response audio. Since the flat stimulation contains multiple tones at pre-selected frequencies with the same amplitude, the response audio contains tones at the same frequencies with the stimulation. The amplitude of each tone can be obtained by multiplying the amplitude of the stimulation tone by the gain factor of the speaker at the corresponding frequency, which can be obtained from the FRC fingerprint of the victim's speaker and is known to the attacker. I used the ultrasound speaker to reproduce the inferred response spectrum. I assume that the attacker has obtained the two audio files, which are recorded by the victim device, so the attacker can directly submit them to the server for authentication. I generated 200 forged response audios for each victim and obtained totally 1,800 forged response from which I extracted 1,800 forged fingerprint samples. I calculated the Euclidean distances between the 1,800 fingerprint samples and the corresponding fingerprint profile. Only two distances were larger than 0.4, which is the threshold adopted by Proximity-Proof to distinguish different devices. In other words, only two forged responses were correctly identified as illegal. The high-fidelity ultrasound speaker has a flat FRC in the frequency range used by acoustic fingerprints, and thus it can reproduce the response spectrum and emulate the FRC of $\mathcal{P}$'s speaker with only a subtle distortion. Therefore, the fingerprint-emulation attack can achieve a success rate as high as 99.8%.

2.3.3.6 Countermeasure Against Fingerprint-Emulation Attack

I propose a dynamic challenge-response mechanism as a defense against the fingerprint-emulation attack. A mobile device has multiple acoustic fingerprints corresponding to different challenges. The attack can thus be thwarted if the fingerprint used in each authentication session has never been used before. Specifically, the server stores multiple fingerprints of an enrolled device referred to as the fingerprint pool and randomly selects a fingerprint for each authentication instance. Every fingerprint can be used only once. The system can update the fingerprint pool when a secure channel between the enrolled phone and the server is available. To eliminate the risk of fingerprint exposure, the user must update the the fingerprint pool with fingerprints that are not revealed to any authentication system.

The amount of a mobile device's distinct fingerprints (referred to as the fingerprint space) is the primary concern about the dynamic challenge-response defense. People may use a mobile device for years, and thousands of authentication sessions may be conducted during this period. If the fingerprint space is not big enough, the dynamic challenge-response mechanism cannot be adopted.

Two distinct FRC fingerprints should not contain any common gain factor to be distinguishable. In what follows, I first demonstrate how I quantify distinct FRC fingerprints of the speaker and then provide the FRC fingerprint space of the speaker through similar processes.

Since the FRC fingerprint of the speaker contains the speaker's gain factors at multiple frequencies, I first investigate the minimal number $\mathcal{K}$ of gain factors needed for accurate device identification through experiments. In my experiments, I selected the first m gain factors as the speaker's FRC fingerprint and calculated the corresponding accuracy for different values of m. For each m, I extracted the speaker's FRC fingerprint for 20 times for each of the nine mobile devices and obtained 180 fingerprint samples. I then identify the device associated with each fingerprint sample and calculated the accuracy. I tested 20 values from 2 to 21 for m. The accuracy increases with m. When m is larger than 10, mobile devices can be identified with accuracy above 95%, and the benefit of further increasing m is insignificant when m exceeds 10. I therefore chose $\mathcal{K}$ to be 10.

Next, I investigate the number of distinct gain factors of a speaker. I assume that a fingerprint $\langle \alpha_1, \alpha_2, \ldots, \alpha_{10} \rangle$ is chosen by the system. Here, α_i is the ith gain factor contained in the fingerprint, and I denote the frequency corresponding to α_i by χ_i. Under the dynamic challenge-response mechanism, the attacker cannot obtain any α_i. However, the attacker may have obtained $\hat{\alpha_i}$ whose corresponding frequency $\hat{\chi_i}$ is close to χ_i and then use $\hat{\alpha_i}$ as α_i to launch the fingerprint-emulation attack. The difference between $\hat{\chi_i}$ and χ_i is denoted by $\Delta\chi_i$. Without loss of generality, I assume that $\Delta\chi_1 = \Delta\chi_2 = \cdots = \Delta\chi_n = \Delta\chi$.

The gain-factor variance of an acoustic component within a small frequency range is insignificant even in the high frequency domain (Beranek and Mellow 2012). If $\Delta\chi$ is not sufficiently large, the two fingerprints $\langle \alpha_1, \alpha_2, \ldots, \alpha_{10} \rangle$ and $\langle \hat{\alpha_1}, \hat{\alpha_2}, \ldots, \hat{\alpha_{10}} \rangle$ are very likely to be indistinguishable, and thus the attacker is identified as legal user and authenticated. I conducted an experiment to obtain the minimal $\Delta\chi$ to defeat the attack. I tested 10 values ranging from 10 Hz to 100 Hz with a step length of 10 Hz and measured the success rate of the attack for each value of $\Delta\chi$. More specifically, the nine devices were chosen as the victim one by one. For a chosen $\mathcal{P}$, I randomly selected 10 frequencies $(\chi_1, ..., \chi_{10})$ from the 21 frequencies used in Proximity-Proof. The gain factors of the victim's speaker on the selected frequencies were extracted

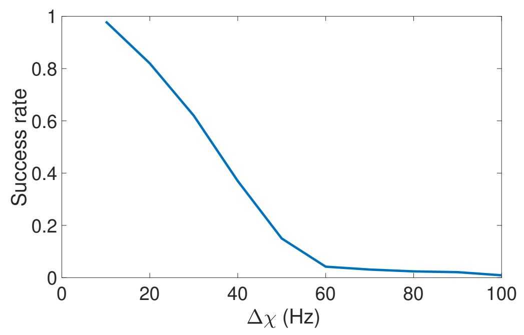


Figure 14. Success rates of the fingerprint-emulation attack.

as the fingerprint $\mathcal{F}$. I then extracted the speaker's fingerprint $\hat{\mathcal{F}}$ on frequencies $\langle \chi_1 + \Delta\chi, ..., \chi_{10} + \Delta\chi \rangle$. The experiment was repeated 100 times for each device, and I calculated the ratio that $\hat{\mathcal{F}}$ is not distinguishable from $\mathcal{F}$ (i.e., the success rate of the attack) for each $\Delta\chi$. Fig. 14 shows my experiment results. The success rate of the attack decreases with the increase of $\Delta\chi$, and the fingerprint-emulation attack can be defeated (i.e., the success rate below 5%) when $\Delta\chi$ is larger than 60 Hz. I meet the requirement for $\Delta\chi$ by choosing the fingerprint frequencies from a set of predetermined values with sufficient gaps. In particular, 66 frequencies ranging from 18 kHz to 21.96 kHz with a step length of 60 Hz are chosen as the candidate frequencies. For each authentication attempt, the system randomly selected 10 frequencies from the candidate frequencies. Since each candidate frequency can be chosen only once, the speaker has $\lfloor 66/10 \rfloor = 6$ distinct FRC fingerprints. This fingerprint space is obviously too small for mobile authentication.

I conducted similar experiments and derived the FRC fingerprint space a microphone is five. Therefore, FRC authentication systems are all still vulnerable to the fingerprint-emulation attack due to the very small fingerprint space.

2.4 ANP-based Mobile Authentication

In this section, I demonstrate ANP-based mobile authentication systems and experimentally evaluate their security and usability (Han et al. 2021). I still consider a system consisting of the prover $\mathcal{P}$, the verifier $\mathcal{V}$, and the server $\mathcal{S}$, as shown in Fig. 1.

ANP is a hardware feature of an acoustic element related to its nonlinear properties. Practical microphones and speakers on commodity mobile devices are only approximately linear in the audible range due to cost considerations and exhibit non-linearity in the non-audible range. In particular, I have

$$S_{\text{out}} = \sum_{i=1}^{\infty} g_i S_{\text{in}}^i, \tag{2.25}$$

where g_i is called the ith-order non-constant nonlinear coefficient. According to (Aurelle et al. 1996), $\{g_i | i \geq 1\}$ are sensitive to the frequencies in S_{in}, and $\{g_i | i \geq 2\}$ are also sensitive to the power of individual frequency components in S_{in}. Given a specific S_{in}, g_i is determined by the nonlinear characteristic of the acoustic element. Due to the nonlinear relation between the input and output signals, the output signal contains new frequency components not present in the input signal (Roy, Hassanieh, and Choudhury 2017; G. Zhang et al. 2017), and those new frequency components are referred to as distortion components. The subsequent discussion refers to ANP as the amplitudes of distortion components produced by the nonlinearity of the speaker, microphone, or both. Different speakers or microphones have distinct ANPs for the same input signals.

In the device-to-device authentication system NAuth (X. Zhou et al. 2019), ANP is used to distinguish different speaker-microphone pairs. NAuth is quite effective in the targeted application scenarios but is not verifier-agnostic. Here I extend NAuth (X. Zhou et al. 2019) by introducing two ANP authentication systems, M-ANP and

SM-ANP, which identify $\mathcal{P}$ using the ANP fingerprints of its microphone and speaker-microphone pair, respectively. I do not consider identifying $\mathcal{P}$ with its speaker's ANP fingerprint because it is difficult to extract verifier-agnostic ANP fingerprints of a speaker. In particular, in order to extract a speaker's fingerprint, a microphone must be used to capture the speaker's output audio. Most microphones, including high-quality ones, exhibit significant nonlinearity in the high frequency domain. The high-frequency audio that can invoke the speaker's nonlinear distortion can also cause distortion at the microphone. Therefore the distortion components in the recorded audio is affected by the microphone and cannot be used to fingerprint the speaker alone.

I consider random impersonation, replay, and fingerprint-emulation attacks against M-ANP and SM-ANP. As demonstrated in Section 2.3.3, the co-located attack can be easily defeated with acoustic distance ranging. So I does not discuss the co-located attack in this chapter.

2.4.1 M-ANP Design

2.4.1.1 Challenge audio

M-ANP uses a high-frequency audio with two tones as the challenge audio played by verifier $\mathcal{V}$ to prover $\mathcal{P}$. In particular, the challenge audio S_{in} is generated as

$$S_{\text{in}} = A_1 \cos(2\pi f_1 t) + A_2 \cos(2\pi f_2) \ . \tag{2.26}$$

The nonlinearity of the microphone in COTS smartphones and smartwatches is more significant in the high frequency range above 18 kHz (G. Zhang et al. 2017). So I require $f_2 > f_1 \geq 18$ kHz. Since the nonlinear coefficient g_i in Eq. (2.25) of a common microphone is negligible for $i \geq 3$ (Roy, Hassanieh, and Choudhury 2017), the nonlinear output of $\mathcal{A}$'s microphone before low-pass filtering can be approximated by

$$\begin{aligned} S_{\text{out}} &\approx g_1 S_{\text{in}} + g_2 S_{\text{in}}^2 \\ &= \frac{g_2}{2}(A_1^2 + A_2^2) + g_1 A_1 \cos(2\pi f_1 t) + g_1 A_2 \cos(2\pi f_2) \\ &\quad + \frac{g_2 A_1^2}{2} \cos(4\pi f_1 t) + \frac{g_2 A_2^2}{2} \cos(4\pi f_2 t) \\ &\quad + g_2 A_1 A_2 \Big(\cos(2\pi(f_2 + f_1)t) + \cos(2\pi(f_2 - f_1)t) \Big) \ . \end{aligned} \tag{2.27}$$

Since a typical microphone's cutoff frequency is 22 kHz, the frequency components at $2f_1$, $2f_2$, and $f_2 + f_1$ in S_{out} cannot be recorded. I additionally require $f_2 - f_1 < 18$ kHz so that the distortion component $g_2 A_1 A_2 \cos(2\pi(f_2 - f_1)t)$ can not only be recorded but also be differentiated from the two tones at f_1 and f_2, respectively. As I will see shortly, this distortion component is used to construct the ANP fingerprint of $\mathcal{P}$.

Verifier $\mathcal{V}$ cannot use an ordinary speaker to generate S_{in}. In particular, different COTS speakers exhibit distinct and significant nonlinearity in the high-frequency range above 18 kHz. So S_{in} would invoke the speaker's nonlinear distortion that would further result in many low-frequency distortion components in its output. Such unwanted distortion components can be recorded and mixed with those induced by $\mathcal{P}$'s microphone. The fingerprint extracted from the recorded audio would thus be tied to both $\mathcal{P}$'s microphone and $\mathcal{V}$'s speaker, which violates the verifier-agnostic requirement.

I propose a cost-effective solution based on COTS ultrasound transducers which each costs at most several US dollars. In particular, I let each verifier use two ultrasound transducers with each generating a unique tone in S_{in}. Although ultrasound transducers also exhibit nonlinearity, the resulting distortion components are in the high-frequency range above 22 kHz and thus cannot be recorded by $\mathcal{P}$'s microphone.

To see this more clearly, consider an arbitrary transducer $i \in [1, 2]$. The input to transducer i is an electrical signal $A'_i \cos(2\pi f_i t)$, and the corresponding nonlinear output can be modeled as

$$\begin{aligned} T_i &\approx g_{1,i} A'_i \cos(2\pi f_i t) + g_{2,i}(A'_i \cos(2\pi f_i t))^2 \\ &= g_{1,i} A'_i \cos(2\pi f_i t) + \frac{g_{2,i} A'^2_i}{2}(1 + \cos(4\pi f_i t)) \ , \end{aligned} \tag{2.28}$$

where $g_{1,i}$ and $g_{2,i}$ denote the first-order and second-order coefficients of transducer i, respectively. Since I require that $f_i \geq 18$ kHz, the distortion component at $2f_i$ cannot be recorded by $\mathcal{P}$'s microphone. In addition, the DC component can be easily filtered from the audio recording.

I further use a simple calibration to extract transducer-agnostic and thus verifier-agnostic fingerprints. In particular, each $g_{1,i}$ corresponds to the gain of transducer i which is a standard parameter in the technical specification of the transducer. Since different transducers may have distinct gain factors, I set $A'_i = A_i / g_{1,i}$. Therefore, the effective output from transducer i with regard to $\mathcal{A}$'s microphone is $g_{1,i} A'_i \cos(2\pi f_i t) = A_i \cos(2\pi f_i t)$, which is exactly the challenge tone T_i I need in Eq. (2.26).

2.4.1.2 Fingerprint extraction and matching

The absolute amplitude of the distortion component at frequency $f_2 - f_1$ cannot be directly used as $\mathcal{P}$'s fingerprint due to the Automatic Gain Control (AGC) system in common microphones. Specifically, the system automatically adjusts the microphone gain according to the perceived sound volume. So the measured amplitude at frequency $f_2 - f_1$ may vary considerably for different verifiers and/or verifier-$\mathcal{A}$ distances instead of equaling the ideal constant $g_2 A_1 A_2$.

Since the AGC system affects all the frequency components almost equally (**GuptaMet04**), I propose to use the relative amplitude as $\mathcal{P}$'s fingerprint. For this purpose, I add a reference tone $A_0 \cos(2\pi f_0 t)$ to S_{in}, which is played by an additional transducer at the verifier. Here, A_0 and f_0 are both system constants. I require f_0 much below 18 kHz and also any possible $f_2 - f_1$ so that $A_0 \cos(2\pi f_0 t)$ incurs negligible nonlinear distortion at the microphone. Then I define the fingerprint element as the absolute amplitude of frequency $f_2 - f_1$ divided by that of frequency f_0.

M-Print uses $\kappa \geq 1$ different challenge audios that differ in frequencies and/or amplitudes in each authentication session, leading to κ fingerprint elements. $\mathcal{P}$'s fingerprint is extracted as $\Theta_{\text{M}} = \langle \theta_1, ..., \theta_\kappa \rangle$, where θ_i denotes the fingerprint element corresponding to the ith challenge audio. The larger κ, the longer the authentication time, the higher distinguishable Θ_{M}, the more reliable the authentication result, and vice versa.

I also need to mitigate the impact of ambient noise to extract Θ_{M}. For this purpose, I let the verifier play each challenge audio for a duration of ω and then keep silent for ω. Meanwhile, $\mathcal{P}$'s microphone kept recording with a sampling frequency of 44.1 kHz. The ambient noise can be considered constant during this short duration (e.g., $\omega = 50$ ms in my experiment). After applying fast Fourier transform to the audio captured by $\mathcal{P}$'s microphone, I subtracted the noise spectrum in the silent period from the audio spectrum in the non-silent period. The resulting differential spectrum was used to extract the "noise-free" fingerprint for this challenge audio. This process was repeated multiple times, and the average result was used as Θ_{M} for final verification by the authentication server.

I use the scaled Euclidean distance to compare two fingerprints to avoid the dominance of large-valued elements. In particular, assume that the authentication server stores an authentic fingerprint Θ'_{M} for the κ challenge audios. It compares Θ'_{M} with the extracted Θ_{M} by computing

$$\mathsf{diff}(\Theta_{\mathrm{M}}, \Theta'_{\mathrm{M}}) = \sqrt{\sum_{l=1}^{\kappa} \left(\frac{\theta_l - \theta'_l}{\theta_l + \theta'_l} \right)^2} . \tag{2.29}$$

If $\mathsf{diff}(\Theta_{\mathrm{M}}, \Theta'_{\mathrm{M}})$ is no larger than a system threshold τ_{M}, the authentication server considers the responses from $\mathcal{P}$ and authenticate the request. κ and τ_{M} are obtained through experiments. I tested 20 candidate values ranging from 1 to 20 for κ. For each value, I obtained the corresponding τ_{M} and calculated the identification accuracy.

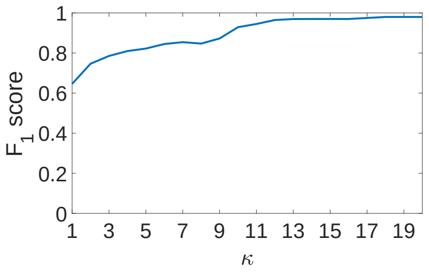
I use the F_1 score to obtain τ_{M} corresponding to a specific κ. Two Prowave 250ST160 transducer were used to generate the challenge, and two Agilent 33220A signal generators were used to power the transducers. I chose 10 challenges that each contains κ challenge audios. The frequencies of the challenge tones were randomly selected, and the output voltages of the signal generators were fixed as 10 V. For each of the 20 devices, I extracted its fingerprints corresponding to each of those 10 challenges for 20 times and totally got 4,000 testing fingerprint samples. The distance between transducers and the device's microphone, denoted by d, may also has impacts on the amplitudes of distortion components and thus affects the ANP fingerprint. I randomly chose a value between 10 cm and 25 cm as d in each experiment so that the obtained κ and τ_{M} are robust to slight changes of d. Then the 20 devices were chosen as $\mathcal{P}$ one by one. When a devices was chosen as $\mathcal{P}$, the rest 19 devices were considered unauthenticated, and I extracted $\mathcal{P}$'s fingerprints corresponding to the 10 selected challenges one more time as the reference fingerprints for later classification. Then I tried 20 values ranging from 0.05 to 1 with a step of 0.05 as τ_{M} to identify whether each testing sample comes from the prover. Based on the classification result, I calculated the F_1 scores corresponding to each τ_{M} as follows:

$$\begin{aligned}
F_1 \text{ score} &= \frac{2}{\dfrac{1}{\text{Recall}} + \dfrac{1}{\text{Precision}}} \\
\text{Precision} &= \frac{\text{TP}}{\text{TP} + \text{FP}} \\
\text{Recall} &= \frac{\text{TP}}{\text{TP} + \text{FN}}
\end{aligned} \tag{2.30}$$

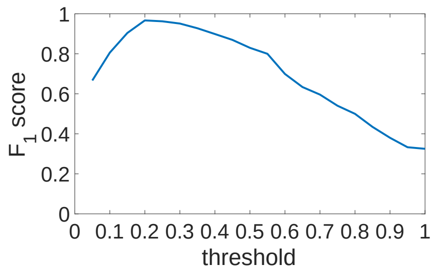
.

Here, TP denotes the number of fingerprint samples correctly recognized as being the fingerprints of $\mathcal{P}$; FP and FN denote the number of fingerprint samples incorrectly recognized as being and not being the fingerprints of $\mathcal{P}$, respectively. For one evaluated value of τ_{M}, I obtained 20 F_1 scores with different devices chosen as $\mathcal{P}$. I chose the evaluated value with the highest average F_1 score as τ_{M}.

The maximum average F_1 scores corresponding to different κ are shown in Fig. 15a. With the increase of κ, the maximum average F_1 score increases. The system can achieve an average F_1 score of 0.96 with an average Precision of 97.6% and an average Recall of 95.2% when κ is 12. However, the benefit of increasing κ is insignificant when κ is larger than 12. In particular, the system achieve average F_1 scores of 0.964 and 0.965 with κ equaling 13 and 14, respectively. Compared with the performance adopting $\kappa = 12$, the average F_1 score increases by less than 0.5%, while the challenge audio length increases by more than 8%. To avoid unnecessary time consumption, M-ANP adopts $\kappa = 12$. Fig. 15b shows the average F_1 scores corresponding to different thresholds when κ is 12. A threshold of 0.2 achieves the highest average F_1 score, so I adopt 0.2 as τ_{M}. With $\kappa = 12$ and $\tau_{\mathrm{M}} = 0.2$, I can identify the devices associated with the 4,000 fingerprint samples with accuracy of 96.4%.



(a) F_1 score corresponding to different κ.



(b) F_1 score corresponding to different threshold ($\kappa = 12$).

Figure 15. F_1 score corresponding to different κ and threshold.

2.4.1.3 Overall performances of M-ANP

I further evaluated M-ANP in three common scenarios with different noise volumes: the office, the store, and the restaurant. Due to hardware constraints, I can only conduct the experiment in the lab. So I recorded the ambient noise in those scenarios and played the recording when I conducted the experiments to emulate those scenarios.

Android devices (12)	Nexus 5 (2), Nexus 7 (2), Google Pixel 2 (2), Google Pixel 3 (2), Samsung S5 (2), and Samsung S7 (2)
iOS devices (8)	iPhone 5 (1), iPhone 5s (1), iPhone 6 (3), iPhone XR (1), iPad 2 (2), and iPad 4 (1)

Table 1. Mobile devices in experiments.

I used the 20 mobile devices listed in 1 to evaluate the performance of M-ANP. I randomly selected 10 challenges and extracted every device's fingerprints corresponding to each challenges for 80 times (20 times in each scenario). For each scenario, I had obtained a testing set that contained 400 fingerprint samples. The 20 mobile devices were selected as the prover one by one, and the rest 19 devices were considered unauthentic. I extracted the chosen devices fingerprints corresponding to those 10 challenges without playing the recorded noise and used the extracted fingerprints as

M-Print	Office	Store	Restaurant
Precision	97.2%	94.7%	94.1%
Recall	95.1%	93.7%	92.5 %
F_1 score	0.96	0.94	0.93

Table 2. The overall performance of M-ANP.

the references. Then I distinguished whether the fingerprints in the testing sets are associated to the prover or not and calculated the Precision and Recall. After all the 20 devices had been chosen as the prover, I calculated the averaged Precision and Recall for each scenario, and the results are shown in Table 2. M-ANP performs best in the office scenario. The performances in noisy scenarios, such as the store and restaurant scenarios, are comparable to that in the office.

2.4.2 SM-ANP Design

2.4.2.1 Challenge audio

SM-ANP adopts the AM modulated signal used in NAuth (X. Zhou et al. 2019) as the input to $\mathcal{P}$'s speaker. The challenge signal is obtained by modulating a baseband signal of frequency f_b upon a carrier signal of frequency f_c and is represented by

$$S_{\text{in}} = A_{f_c} \sin(2\pi f_c t)(1 + A_{f_b} \sin(2\pi f_b t)) \ . \tag{2.31}$$

The challenge in SM-ANP specifies f_c, f_b, A_{f_c}, and A_{f_b}. I invoke the speaker on $\mathcal{P}$ through a standard API which takes a discrete sequence of amplitude values sampled from an sound wave as input and outputs the corresponding audio. For this purpose, I sample S_{in} at a common speaker's maximum sample rate $f_s = 48$ kHz to obtain the following sequence as the input to the speaker API:

$$\hat{S}_{\text{in}}[i] = A_{f_c} \sin(2\pi f_c i/f_s)(1 + A_{f_b} \sin(2\pi f_b i/f_s)) \ , \tag{2.32}$$

where $\hat{S}_{\text{in}}[i]$ denotes the ith element for all $i = 1, 2, \ldots$. S_{in} can invoke significant nonlinear distortion of the speaker-microphone pair on $\mathcal{P}$, which results in many distortion components in the output of $\mathcal{P}$'s microphone with a cutoff frequency of 22 kHz. According to Eq. (2.25) and trigonometric expansion, these distortion components are at frequencies nf_c, mf_b, and $nf_c \pm mf_b$, where $k, m, n \in \mathbb{N}$ and $nf_c, mf_b, nf_c \pm mf_b < 22$ kHz (X. Zhou et al. 2019).

I carefully select f_c and f_b to enhance the nonlinear distortion components in the microphone's output. In particular, I find that many distortion frequencies are actually the same for different combinations of k, m, and n. For example, if $f_c = 20$ kHz and $f_b = 5$ kHz, I have $2f_b = f_c - 2f_b = 10$ kHz. Based on this observation, I can stack up the distortion components so that their combined effect is more profound. For this purpose, I set $f_c = nf_b$ ($n \in \mathbb{N}+$), resulting in $\lfloor 22 \text{ kHz}/f_b \rfloor$ distortion components at frequencies $\{kf_b | 1 \leq k \leq \lfloor 22 \text{ kHz}/f_b \rfloor\}$).

The next issue is to decide the feasible values for f_b and f_c. Assume that A_{f_c} and A_{f_b} are fixed for the time being. The microphone can record $\lfloor 22 \text{ kHz}/f_b \rfloor$ distortion components, each corresponding to one fingerprint element of prover $\mathcal{P}$. If f_b is set too large, very few fingerprint elements can be obtained and thus may be insufficient to distinguish a large number of devices. On the other hand, if f_b is set too small, there may be too many distortion components whose amplitudes may be too small given the fixed total power of the recorded audio, and such weak distortion components may be indistinguishable from noise and thus would dramatically decrease the identification accuracy. So I select f_b from a range $[f_{b,\min}, f_{b,\max})$. In addition, the nonlinearity of speakers and microphones is more significant in the inaudible domain, and the cutoff frequency of the speaker in common mobile devices is 24 kHz. So I set 18 kHz $\leq f_c <$ 24 kHz with f_c being a multiple of f_b.

In SM-ANP, prover $\mathcal{P}$ generates the audio with the highest possible volume to maximize the nonlinear distortion. In particular, the strength of nonlinear distortions is significantly affected by the modulation depth defined as $\zeta = A_{f_b}/A_{f_c}$ (G. Zhang et al. 2017; X. Zhou et al. 2019), which should neither be too large nor too small. SM-ANP selects ζ from a predetermined range $[\zeta_{\min}, \zeta_{\max})$, which can be empirically determined as well. Once ζ is chosen, I maximize A_{f_c} and A_{f_b} under the constraint that the maximum value of sample $\hat{S}_{\text{in}}[i]$ does not exceed the default peak value defined by the operating system (e.g., 32767 in Android).

2.4.2.2 Fingerprint extraction and matching

Let θ'_i denote the amplitude of the ith distortion component at frequency if_b for all $1 \leq i \leq \beta$. I define the fingerprint of $\mathcal{P}$ for specific f_b, f_c, and ζ as $\Theta_{\text{SM}} = \langle \theta_1, \ldots, \theta_\beta \rangle$, where

$$\theta_i = \frac{\theta'_i}{\sqrt{\sum_{j=1}^{\beta} {\theta'_j}^2}} \,. \tag{2.33}$$

I use the normalized amplitudes instead of absolute values to counteract the impact of the AGC system. I also adopt the method demonstrated in Section 2.4.1.2 to mitigate the impact of ambient noise.

SM-ANP also uses the scaled Euclidean distance as in Eq. (2.29) to measure the fingerprint similarity. If the calculated distance does not exceed a system threshold τ_{SM}, the authentication server considers prover $\mathcal{P}$ and thus authentic, and vice versa.

2.4.2.3 Parameters for SM-ANP

I now explain how to obtain the pre-determined parameters of SM-ANP.

$f_b^{\min}$, $f_b^{\max}$, **and** $\boldsymbol{\tau_{\mathrm{SM}}}$. The 12 android devices listed in Table 1 were used in the experiment. I fixed ζ as 100% and tried 30 frequencies ranging from 100 Hz to 3 kHz with a step length of 100 Hz as f_b. For each frequency f_b, I set $f_c = \lceil \frac{18\text{ kHz}}{f_b} \rceil f_b$. I obtained totally 30 challenges and used the F_1 *mesurement* as in M-ANP to obtain the corresponding τ_{SM} and average F_1 *score*. The average F_1 *score* is above 0.95 when f_b is between 700 Hz and 2 kHz and dramatically lower when f_b is out of this range. I therefore chose 700 Hz and 2 kHz as $f_b^{\min}$ and $f_b^{\max}$, respectively. Among the 16 frequencies within $[f_b^{\min},\ f_b^{\max}]$, I found that 0.15 is the optimal threshold for 10 of them, and the average F_1 *score* under this threshold is above 0.95 for the rest 6 frequencies as well. Based on this finding, I chose $\tau_{\mathrm{SM}} = 0.15$. With $\tau_{\mathrm{SM}} = 0.15$, I can identify the devices associated with the collected fingerprint with an accuracy of 95.3%.

$\zeta_{\min}$ **and** $\zeta_{\max}$. The same 12 android devices were used in this experiment. The impact of ζ is more significant when the amplitudes of the distortion components are small. So I fixed f_b and f_c as 700 Hz and 18.2 kHz, respectively. I increased ζ from 50% to 150 % with a step length of 5% and obtained the corresponding averaged F_1 *score* for each value. I found that the averaged F_1 *score* is above 0.95 when M_d is between 75% and 110% and therefore chose $\zeta_{\min} = 75\%$ and $\zeta_{\max} = 110\%$.

SM-Print	Office	Store	Restaurant
Precision	96.3%	92.6%	93.1%
Recall	95.2%	92.1%	91.7 %
F_1 score	0.96	0.92	0.92

Table 3. The overall performance of SM-ANP.

2.4.2.4 Overall performances of SM-ANP

I further evaluated SM-ANP in office, store, and restaurant scenarios with the 12 android devices. I randomly selected 10 challenges and used the same way as I did with M-ANP to obtain the Precision and Recall of SM-ANP in three scenarios. To avoid redundancy, I omit the description of the experiment details and only show the results in Table 3. Similarly to M-ANP, SM-ANP performs best in the office and comparably well in the store and restaurant. Since the speaker's power is weaker compared with the transducer's, the dynamic noise's impact to SM-ANP is more significant.

2.4.3 Attacks Against M-ANP and SM-ANP

I consider random impersonation, replay, and fingerprint-emulation attacks against M-ANP and SM-ANP. As demonstrated in Section 2.3.3, the co-located attack can be easily defeated with acoustic distance ranging. So I does not discuss the co-located attack in this chapter. The random impersonation is the same as demonstrated in Section 2.2.2, so I omit its detail. M-ANP and SM-ANP both adopt F-response, so the replay and fingerprint-emulation attacks are essentially the same. I only detail the fingerprint-emulation attack in this section.

2.4.3.1 Fingerprint-Emulation Attack

In S-ANP and SM-ANP, prover $\mathcal{P}$'s fingerprint partially reveals the spectrum of the response. In S-ANP, the fingerprint reveals the amplitude ratio θ of the distortion component to the reference tone. Attacker $\mathcal{A}$ can forge a response by setting the amplitude of the distortion component as θ multiplied by the reference tone's amplitude. In SM-ANP, the fingerprint reveals the normalized amplitudes of distortion components. Attacker $\mathcal{A}$ can forge the response by using each fingerprint element as the amplitude of the corresponding distortion component. S-ANP and SM-ANP both use F-response, so $\mathcal{A}$ can directly submit the forged response to the system through the short range wireless channel between $\mathcal{A}$ and $\mathcal{V}$.

The generation of the forged response is essentially the inverse process of fingerprint extraction, so the fingerprint extracted from the forged response is identical to $\mathcal{P}$'s fingerprint. The system identifies $\mathcal{A}$ as $\mathcal{P}$ and is thus compromised.

2.4.4 Dynamic Challenge-Response for M-ANP

A straightforward defense against both replay and fingerprint-emulation attacks is to let the authentication server issue unique challenge audios for different authentication sessions to prevent possibly exposed fingerprints from being used for launching fingerprint-emulation attack. Specifically, the authentication server randomly selects κ tone pairs as the challenge for each authentication session and never reuses the same set of κ tone pairs in the future. However, even a subset of reused tone pairs can be used to launch the fingerprint-emulation attack. In what follows, I first quantify the amount of distinct tone pairs and then analyze the resilience of the dynamic challenge-response M-ANP.

2.4.4.1 ANP M-Print space

To estimate the fingerprint space of M-Print, I first examine the impact of tone frequencies and amplitudes on the distortion components (or equivalently fingerprint elements). Consider two arbitrary challenge tones $A_1 \cos(2\pi f_1 t)$ and $A_2 \cos(2\pi f_2 t)$. Ideally, this tone pair would result in a distortion component at frequency $f_2 - f_1$ with amplitude $a_{i,j} = g_2 A_1 A_2$. However, I observe from experiments that g_2 is not a constant but depends on A_1, A_2, f_1, and f_2. Due to ambient noise and measurement errors, this distortion component may appear at a slightly different frequency and with a slightly different amplitude. Even worse, it may be very similar to the distortion component induced by a different tone pair, say $A'_1 \cos(2\pi f'_1 t)$ and $A'_2 \cos(2\pi f'_2 t)$.

To guarantee sufficient distinguishably among different distortion components, it is necessary to ensure that no two tone pairs are very similar in both frequencies and amplitudes. I thus have the following criteria: (1) $\max\{|f_1 - f'_1|, |f_2 - f'_2|\} \geq h_f$; (2) $\max\{|A_1 - A'_1|, |A_2 - A'_2|\} \geq h_a$. As long as at least one criterion is satisfied, the two resulting distortion components can be distinguished with overwhelming probability.

I meet the above requirements by choosing the frequency and amplitude of each challenge tone from a set of predetermined candidate tones with sufficient gaps. In particular, let $f_{\max}$ and $f_{\min}$ denote the highest and lowest acoustic frequencies that can induce significant nonlinear distortion of the microphone, respectively. For example, I can set $f_{\min} = 18$ kHz and $f_{\max} = 50$ kHz according to (Roy, Hassanieh, and Choudhury 2017). The number of possible tone frequencies is then $N_f = \lceil (f_{\max} - f_{\min})/h_f \rceil$. In addition, let $A_{\max}$ and $A_{\min}$ denote the highest and lowest possible tone amplitudes, respectively, leading to $N_a = \lceil (A_{\max} - A_{\min})/h_a \rceil$ possible amplitudes for each tone. $A_{\max}$ depends on the maximum working voltage of the transducer, and $A_{\min}$ must be sufficiently large to induce nontrivial nonlinear distortion and can be obtained through experiments.

Given that $f_2 - f_1$ must be smaller than 18 kHz, I estimate the size of the fingerprint space as follows. For simplicity, assume that 18 kHz can be divided by h_f such that $\lambda = \frac{18\text{kHz}}{h_f}$. When f_1 is smaller than $f_{\max} - 18$ kHz, all the λ frequencies within the range $[f_1, f_1 + 18 \text{ kHz}]$ can be used as f_2. When f_1 is larger than $f_{\max} - 18$ kHz, there are $\frac{f_{\max} - f_1}{h_f}$ frequencies that can be used as f_2. Therefore, there are total $\lambda N_f - \lambda(\lambda+1)/2$ tone-frequency pairs. Since each tone has N_a possible amplitudes, there are $\psi_{\mathrm{M}} = (\lambda N_f - \lambda(\lambda+1)/2)N_a^2$ distinct tone pairs, each leading to a unique distortion component (or fingerprint element). Prover $\mathcal{P}$ may have $N_{\text{mic}} \geq 1$ microphones. For example, iPhone models starting from 6s and 6s+ all have four microphones. So I can have $N_{\mathrm{M}} = N_{\text{mic}}\psi_{\mathrm{M}}$ unique fingerprint elements of $\mathcal{P}$, leading to $\binom{N_{\mathrm{M}}}{\kappa}$ distinct fingerprints in total for a challenge with κ audio.

2.4.4.2 System parameters

Now I discuss how I obtained $A_{\min}$, h_f, and h_a through experiments involving the same set of 20 devices shown in Table 1. Two Prowave 250ST160 transducers were used to generate the challenge audio, and two Agilent 33220A signal generators were used as the power supply.

I obtained the transducer's minimum input voltage $V_{\min}$ instead of $A_{\min}$. I tried 17 voltages ranging from 2 V to 10 V with a step length of 0.5 V as the transducer's input voltage V_{in}. For each V_{in} value, I generated 10 challenges. The tone frequencies of each challenge were randomly chosen, and the amplitudes of all the challenges tones were fixed to $g_1 V_{\text{in}}$, where g_1 denotes the gain factor of the transducer. I extracted the 20 devices' fingerprints corresponding to each of those challenges 20 times and obtained 4,000 testing samples. Then the 20 devices were chosen as $\mathcal{P}$ one by one. Given a chosen $\mathcal{P}$, I extracted its fingerprints corresponding to those 10 challenges one more time as the fingerprint profile stored in $\mathcal{S}$ which were used to classify the 4,000 testing samples. Based on the classification results, I calculated the F_1 score. I totally obtained 20 F_1 scores for each V_{in} value and calculated the average F_1 score. The results show that the average F_1 score increases as V_{in} increases and exceeds 0.95 when V_{in} is larger than 6 V. When V_{in} is lower than 5.5 V, the average F_1 score is below 0.78. Therefore, I chose $V_{\text{min}} = 6$ V and $A_{\text{min}} = g_1 V_{\text{min}}$.

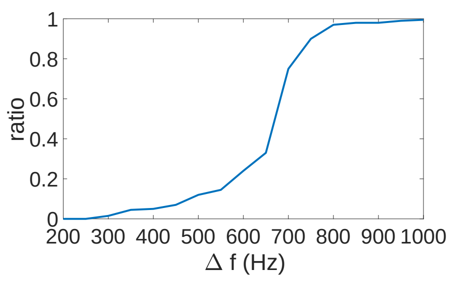
The choice of h_f and h_a should guarantee that a microphone's fingerprints with respect to different challenges are distinguishable, i.e., the distance between two fingerprints is larger than the threshold τ_{M}. Since the transducer is powered by the signal generator, the amplitude of the challenge tone is determined by the voltage of the signal generator. I denote the voltage corresponding to amplitude A_i by V_i. Since the second requirement for the amplitudes of challenge tones is equivalent to $\max\{|V_1 - V_1'|, |V_2 - V_2'|\} \geq h_v$, I obtained h_v instead of h_a. In M-ANP, the two most similar fingerprints, denoted by Θ and Θ', differ in only one element. Without loss of generality, I assume they differ in the first element and model the distance between the two fingerprints as

$$\mathsf{diff}(\Theta, \Theta') = \frac{\theta_1 - \theta_1'}{\theta_1 + \theta_1'}. \tag{2.34}$$

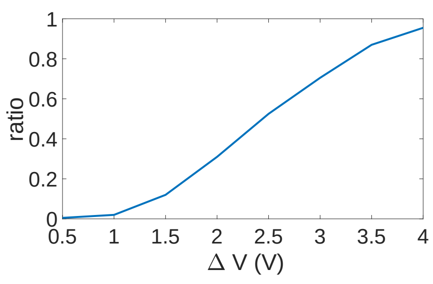
Therefore, two fingerprint elements should be distinguishable if the scaled distance between them is larger than τ_{M}. I seek to find the minimum values for Δf and ΔV so that the two elements corresponding to $\langle f_1,\ A_1,\ f_2,\ A_2\rangle$ and $\langle f_1,\ A_1,\ f_2+\Delta f,\ A_2\rangle$ or the two elements corresponding to $\langle f_1,\ A_1,\ f_2,\ A_2\rangle$ and $\langle f_1,\ A_1,\ f_2,\ A_2+g_1\Delta V\rangle$ are distinguishable.

I conducted an experiments with 20 mobile devices. I selected a base tone pair by randomly selecting a tone frequency pair and fixing the amplitudes of each tone to $A_{\min}$. The base tone pair is denoted by $\langle f_1,\ A_{\min},\ f_2,\ A_{\min}\rangle$. I extracted the fingerprint elements of the 20 mobile devices corresponding to the base tone pair as the reference elements. Then I increased Δf from 200 Hz to 1 kHz with a step length of 50 Hz and increased ΔV from 0.5 V to 4 V with a step length of 0.5 V. For each Δf and ΔV, I extracted the fingerprint elements corresponding to $\langle f_1,\ A_{\min},\ f_2+\Delta f,\ A_{\min}\rangle$ and $\langle f_1,\ A_{\min},\ f_2,\ A_{\min}+g_1\Delta V\rangle$ of each device for 20 times, where g_1 is the gain factor of the transducer. Totally 400 testing element samples for each individual tone pair were obtained. I repeated the whole process 10 times with different base tone pairs and calculated the scaled distances between each extracted element and the corresponding reference element. If the distance is larger than τ_{M}, the extracted element is considered distinguishable, and vice versa. Fig. 16a and Fig. 16b show the ratios of distinguishable elements corresponding to each Δf and ΔV, respectively. I can see that more than 96.3% of fingerprint elements are distinguishable when Δf is no less than 800 Hz, and more than 95.5% of fingerprint elements are distinguishable when ΔV is 4 V. So I adopt $h_f = 800$ Hz and $h_v = 4$ V.

Fingerprint space. Based on my experiment results, I estimate the fingerprint space as follows. There are total $N_f = \lceil(50 \text{ kHz} - 18 \text{ kHz})/800 \text{ Hz}\rceil = 40$ feasible tone frequencies. Since the maximum working voltage of the transducer is 20 V, there are total $N_a = \lceil(20 \text{ V} - 4 \text{ V})/5 \text{ V}\rceil = 4$ feasible tone amplitudes. Therefore, a



(a) Frequency.



(b) Voltage.

Figure 16. The ratios of distinguishable elements.

mobile device with two microphones, like Samsung S7, has about N_{M} =20,000 distinct fingerprint elements and around 8×10^{42} distinct fingerprints.

2.4.4.3 Security analysis

Now I analyze the resilience of dynamic challenge-response M-ANP to the fingerprint-emulation attack. Assume that $\mathcal{A}$ has acquired ϵ fingerprints of $\mathcal{P}$. $\mathcal{A}$ tries to impersonate $\mathcal{P}$ and starts an authentication instance at $\mathcal{V}$. $\mathcal{S}$ randomly selects a fingerprint $\mathcal{F}_s$ from the fingerprint pool, returns the corresponding challenge, and asks for the response. Since $\mathcal{P}$ fingerprints may contain common fingerprint elements, an element of $\mathcal{F}_s$ is known to $\mathcal{A}$ if another fingerprint containing the element has been exposed to $\mathcal{A}$. The probability P_e that a specific fingerprint element in $\mathcal{F}_s$ has been exposed can be estimated as

$$P_e = 1 - (1 - \frac{\kappa}{N_{\mathrm{M}}})^{\epsilon} \ . \tag{2.35}$$

As shown in Section 2.4.4.1, $\mathcal{A}$ can successfully emulate $\mathcal{F}_s$ only if all the κ elements in $\mathcal{F}_s$ are exposed. So the probability for this to occur is given by

$$P_{\text{success}} = P_e^{\kappa} = (1 - (1 - \frac{\kappa}{N_{\mathrm{M}}})^{\epsilon})^{\kappa}. \tag{2.36}$$

To achieve an attack success rate of 0.5, the attacker need obtain more than 5,000 fingerprints, which would take quite a long time and may not be feasible in practice. Therefore, the dynamic challenge-response scheme for M-ANP is resilient to the fingerprint-emulation attack.

2.4.5 Dynamic Challenge-Response for SM-ANP

Similar to M-ANP, SM-ANP can adopt the random challenge-response method to withstand the fingerprint-emulation attack. For this purpose, $\mathcal{S}$ maintains a set of fingerprints, referred to as a fingerprint pool, for each prover $\mathcal{P}$. For each authentication request concerning $\mathcal{P}$, the server randomly chooses one fingerprint from the pool and issues the corresponding challenge to $\mathcal{P}$.

2.4.5.1 ANP SM-Print space.

Now I discuss the ANP SM-Print space, i.e., the number of possible fingerprints for a single prover $\mathcal{P}$ in SM-ANP. Obviously, the scaled Euclidean distance between any two fingerprints should be larger than the threshold τ_{SM}. Since each challenge corresponds to a unique fingerprint, I can just estimate how many distinct challenges in Eq. (2.31) there can be. I first consider the impact of f_b. Although ideally all the distortion frequencies should be multiples of $f_b \in [f_{b,\min}, f_{b,\max})$ below 22 kHz, they may vary slightly due to noise and measurement errors. I thus require a minimum gap h_{f_b} between different f_bs to ensure that their corresponding distortion frequencies can be distinguished. This means that f_b can take $l_{f_b} = \lceil \frac{f_b^{\max} - f_b^{\min}}{h_{f_b}} \rceil$ values. Moreover, the carrier frequency f_c, A_{f_b}, and A_{f_c} all affect the amplitudes of distortion components.

For each given f_b, I require f_c to be a multiple of f_b in $[18, 24)$ kHz, so f_c can take $l_{f_c} = \lceil \frac{6\text{ kHz}}{f_b} \rceil$ possible values. A_{f_b} and A_{f_c} are determined once the modulation depth $\zeta = A_{f_b}/A_{f_c}$ is chosen from $[\zeta_{\min}, \zeta_{\max})$. I thus introduce a minimum gap h_ζ between different modulation depths so that the corresponding distortion components at the same frequencies can have sufficiently different amplitudes. This means that ζ can take $\lceil \frac{f_b^{\max} - f_b^{\min}}{h_{f_b}} \rceil$ values. Finally, I estimate the number of distinct SM-Print fingerprints for each speaker-microphone pair on $\mathcal{P}$ as

$$\psi_{\text{SM}} = \lceil \frac{f_b^{\max} - f_b^{\min}}{h_{f_b}} \rceil \times \sum_{i=1}^{l_{f_b}} \lceil \frac{6\text{ kHz}}{f_{b,\min} + i h_{f_b}} \rceil \,. \tag{2.37}$$

As in latest smartphones or smartwatches, prover $\mathcal{P}$ may have $m \geq 1$ speaker-microphone pairs, leading to $N_{\text{SM}} = m\psi_{\text{SM}}$ distinct fingerprints in total.

2.4.5.2 System parameters

I conducted experiments to obtain $\zeta_{\min}$, $\zeta_{\max}$, h_{f_b}, and h_ζ. 12 Android devices shown in Table 1 were used in experiments.

$\zeta_{\min}$ **and** $\zeta_{\max}$. Since the impact of ζ on the distortion component's amplitude is more significant when the distortion components' amplitudes are small, I chose f_b and f_c to be 700 Hz and 18.2 kHz, respectively. I increased ζ from 50% to 150 % with a step length of 5% and obtained the corresponding F_1 score for each ζ value. The F_1 score is above 0.95 when M_d is between 75% and 110%. As a result, I chose $\zeta_{\min} = 75\%$ and $\zeta_{\max} = 110\%$.

h_{f_b} **and** h_ζ. I obtained the minimum values of Δf_b and $\Delta\zeta$ so that the a device's fingerprints corresponding to $\langle f_b + \Delta f_b,\ \zeta\rangle$ and $\langle f_b,\ \zeta + \Delta\zeta\rangle$ are distinguishable from the fingerprint corresponding to $\langle f_b,\ \zeta\rangle$. Since f_b is within [700 Hz, 2 kHz], the length of the fingerprint, i.e., the number of distortion components, is between 11 and 31. I did not consider the length of 11 because 2 kHz is the only available f_b for this length. I first selected 20 challenges whose corresponding fingerprints are of different length. The f_b for the ith challenge is $\lfloor \frac{22\text{kHz}}{11+i} \rfloor$ and denoted by f_b^i, and the corresponding f_c^i is $\lceil \frac{18\text{ kHz}}{f_b} \rceil f_b$. The length of the fingerprint corresponding to the ith challenge is $11 + i$. The modulation depths of all the challenges are fixed as $\zeta_{\min}$. I iteratively chose one of the 12 devices as prover $\mathcal{P}$ and extracted its fingerprints corresponding to those challenges as the reference fingerprints. I then extracted $\mathcal{P}$'s fingerprints corresponding to $\langle f_b^i + \Delta f,\ \zeta_{\min} \rangle$. I tested 20 values of Δf ranging from 20 Hz to 400 Hz with a step length of 20 Hz. For each challenge, I extracted the $\mathcal{P}$'s fingerprints 20 times. If the extracted fingerprint has a different length from the reference fingerprint or the distance between the extracted fingerprint and the reference fingerprint is larger than τ_{SM}, the extracted fingerprint is considered distinguishable from the reference fingerprint. Next, I extracted the $\mathcal{P}$'s fingerprints corresponding to $\langle f_b^i,\ \zeta_{\min + \Delta\zeta} \rangle$ 20 times. I tested 20 values of $\Delta\zeta$ ranging from 1% to 20% with a step length of 1%. The results show that 97% fingerprints are distinguishable when Δf is larger than 120 Hz, and 95% fingerprints are distinguishable when $\Delta\zeta$ is larger than 6%. Therefore, I choose $h_{f_b} = 120$ Hz and $h_\zeta = 6\%$.

Fingerprint space. Based on the obtained parameters, I estimate that a speaker-microphone pair has approximately 580 distinguishable fingerprints. A mobile device with two speakers and two microphones has approximately 2,320 fingerprints. The fingerprint space is much smaller than that of ANP M-Print and may not be sufficiently large for long-term mobile authentication. The main reason is that the mobile device's speaker has limited power and frequency ranges, leading to a relatively small number of distinguishable challenges.

2.5 Comparison between FRC and ANP Authentication Systems

I summarize the pros and cons of FRC and ANP authentication systems in Table 4. The accuracy is evaluated based on the ratio of acoustic fingerprints whose corresponding devices are correctly identified. The deployability is evaluated based on the hardware with which $\mathcal{V}$ must be equipped. The security is evaluated based on whether the system is resilient to specific attacks (indexed as 'yes' or 'no') and whether the dynamic challenge-response (shortened as dynamic C-R in the table) is adoptable.

Fingerprint scheme		Proxmity-Proof (cross-device)	Proximity-Proof (self-device)	M-ANP	SM-ANP
Accuracy		99.5%	99.3%	96.4%	95.3%
Deployability		COTS microphones and speakers	None	Ultrasound transducers	None
Security	Impersonation	Yes			
	Replay	Yes	No	No	No
	Co-located	Yes (with acoustic distance ranging)			
	Fingerprint emulation	No			
	Dynamic C-R	Not adoptable	Not adoptable	Adoptable	Not adoptable

Table 4. Pros & cons of FRC and ANP authentication systems.

For the cross-device authentication scenarios, M-ANP is more secure than Proximity-Proof. Thanks to the large fingerprint space, the dynamic challenge and response mechanism can be enabled in M-ANP, and the most powerful fingerprint-emulation attack can thus be defeated. However, Proximity-Proof is more deployable in some application scenarios. Particularly, the verifier in an Proximity-Proof must has a speaker and a microphone. In some application scenarios, speakers and microphones are already installed in the device which can act as the verifier. For example, many commercial smart lockers have speakers and microphones for the communication purpose. In this case, Proximity-Proof can be integrated to the existing authentication system without any hardware modification. In contrast, M-ANP requires several ultrasound transducers to be installed on the verifier. Ultrasound transducers are less common compared with commercial speakers and microphone. Hardware modification is almost unavoidable to integrate M-ANP to an existing authentication system. Besides, M-ANP is more disturbing and less resilient to noise compared with Proximity-Proof since it involves audible distortion components.

In the self-device authentication scenarios, SM-ANP and Proximity-Proof are both accurate and deployable. They are robust to the random impersonation and co-located attack but are vulnerable to the replay and fingerprint-emulation attacks, which are actually the same. Although acoustic authentication is not guaranteed to be secure in the self-device authentication scenarios, it can raise the bar for launching possible attacks.

2.6 Related Work

Fingerprinting a mobile device with the unique features of its hardware components has been a hot topic in recent years. The features of motion sensors are used to identify the mobile device in (Dey et al. 2014; Das and Borisov 2016; Bojinov et al. 2014). Dey *et al.* proposed to fingerprint the accelerometer with the bias of its reading (Dey et al. 2014). Bojinove *et al.* used the calibration error of the accelerometer as its fingerprint. Das *et al.* combined the features of the accelerometer and gyroscope to identify the mobile device (Das and Borisov 2016). Ba *et al.* proposed to use the

Photo-Response Non-Uniformity of the camera as the mobile device's fingerprint (Ba et al. 2018). Researcher have also leveraged the imperfection of the WiFi chipset to identify the mobile device (Brik et al. 2008; Polak, Dolatshahi, and Goeckel 2011; Remley et al. 2005).

The author investigates identifying the mobile device with its acoustic elements. There have been many studies on fingerprinting the acoustic elements. Zhou *et al.*, Chen *et al.*, and Han *et al.* all proposed to used the frequency response as the fingerprint of the acoustic element (Z. Zhou et al. 2014; D. Chen et al. 2017; Han et al. 2018). Das *et al.* proposed to use the mel-frequency cepstral coefficients to identify an acoustic element (Das, Borisov, and Caesar 2014). The author's study is complementary to existing work, and the dynamic challenge-response mechanism proposed in this chapter can also be applied to existing schemes to defend them against the fingerprint exposure attack.

The nonlinearity of the acoustic element has been used for different purposes in previous studies. Roy *et al.* studies the feasibility of leveraging the nonlinear distortion of the microphone to record ultrasonic sounds (Roy, Hassanieh, and Choudhury 2017). Zhang *et al.* and Roy *et al.* utilized the nonlinearity of microphones to issue inaudible commands to the voice control system (G. Zhang et al. 2017; Roy et al. 2018). Lin *et al.* proposed an ultrasonic positioning system for mobile devices using the nonlinearity of the microphone (Lin, An, and Yang 2019). The most related work is NAuth (X. Zhou et al. 2019). Zhou *et al.* proposed using ANP to verify the consistency of the audio source in the device-to-device authentication context. Their scheme is quite efficient in the targeted context, but it does not fulfill the *verifier-agnostic* requirement of the distributed authentication system.

2.7 Conclusion and Future Work

In this report, I investigated the suitability of existing acoustic fingerprinting schemes for mobile authentication in terms of security and usability. While I found that all the schemes achieve sufficiently high identification accuracy for mobile authentication, MFCC acoustic fingerprint schemes incur a prohibitive deployment cost due to the need for expensive acoustic elements. In contrast, FRC and ANP authentication systems are both low-cost and verifier-agnostic but are both vulnerable to the fingerprint-emulation attack. To address these limitations, I proposed a dynamic challenge-response mechanism as a strong defense. The proposed system can thwart the fingerprint-emulation attack by not reusing acoustic fingerprints across different authentication sessions. To evaluate whether the proposed mechanism can be integrated into FRC and ANP authentication systems, I quantify the space of FRC and ANP fingerprints of the speaker, microphone, and speaker-microphone pair on the prover device. My experiment results show that ANP M-Print is the only scheme with a sufficiently large fingerprint space to support dynamic challenge-response to withstand the fingerprint-emulation attack.

Although the M-ANP fingerprint space is large enough for one authentication system, it may not be sufficient if a user register the same mobile device for multiple authentication systems. To defeat the fingerprint-emulation attack, a fingerprint used in one authentication system cannot be reused in others, and the fingerprint space assigned to each system dramatically decrease with the increase of system numbers. For future work, I plan to adopt the zero-knowledge-proof technique to reduce the risk of fingerprint and response leakage. Specifically, the system compare the fingerprint associated with a request device and the stored copy without involving any transmission of the fingerprints or responses, making it harder for the attacker to obtain the prover device's fingerprints and responses.

Chapter 3

DEEPJAM: DL-GUIDED JAMMING ATTACK ON CROSS-TECHNOLOGY IOT NETWORKS

3.1 Overview

Jamming is a critical threat against wireless communications. Different jamming attacks have been proposed for attackers with various capabilities(Y. Chen et al. 2009; Wilhelm et al. 2011; Zhao et al. 2019), and there have also been many studies on defeating or detecting jamming attacks (Yan et al. 2016; Chen, Zeng, and Mohapatra 2010). The current wireless environment has become much more complex than before due to the surge of IoT devices all over the world, which results in new threats to wireless security as shown in previous studies. The emerging deep learning technique has demonstrated its efficacy in compromising wireless security (Shi, Davaslioglu, and Sagduyu 2019; Shi, Erpek, et al. 2018). It is thus meaningful to investigate whether the attacker can exploit the complex wireless environment and deep learning technique to launch more effective jamming attacks and also devise corresponding countermeasures.

The unlicensed frequency bands are now crowded with devices of different wireless technologies. For example, WiFi, Zigbee, and Bluetooth all use the 2.4 GHz ISM band. The Cross-Technology Interference (CTI) happens when different kinds of wireless networks on the same frequency band are deployed at proximity. For instance, previous work shows that the throughput of a Zigbee network may drop by more than 60% if a WiFi network coexists (Garroppo et al. 2011; Musaloiu-E and Terzis 2008).

I proposes DeepJam, a new deep learning-guided jamming attack that exploits CTI in the complex wireless environment. I illustrate the basic idea of DeepJam with an example CTI context shown in Fig. 17, where a WiFi network and a Zigbee network coexist. I consider this context because of the prevalence of WiFi and Zigbee networks, and my work can be extended to other scenarios in which CTI exists. The WiFi network contains one Access Point (AP) and multiple WiFi devices, which can be mobile devices, computers, or smart home devices. The Zigbee network contains one coordinator and multiple Zigbee devices, and the network can be the alarm system in a house or the temperature monitoring system in a factory. The adversary aims to disrupt the Zigbee traffic from a specific device in an efficient and stealthy manner. In this context, the victim's transmission may fail with a high probability when CTI happens. Therefore, reactive jamming, which generates a jamming signal upon detection of any Zigbee preamble, is inefficient because it wastes significant energy jamming the victim's transmissions already disrupted by CTI and also the transmissions of other Zigbee devices. Random jamming that transmits a jamming signal regardless of the presence or absence of the victim's transmissions is not only more energy-inefficient but also easier to detect. By comparison, DeepJam can significantly reduce the victim's throughput with far fewer jamming signals by only jamming the victim's transmissions which are not subject to CTI and thus achieve much more stealthy jamming than conventional random and reactive jamming strategies.

It is challenging to accurately predict when jamming is necessary. The random backoff periods and the asynchronous clocks of the WiFi and Zigbee networks result in chaotic wireless traffic, making it difficult to capture the temporal traffic patterns. I first investigate the Zigbee MAC protocol and propose a slotted formulation of

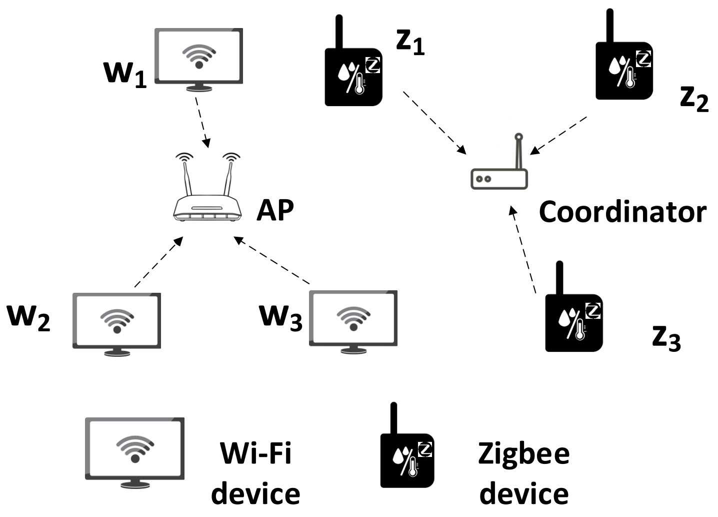


Figure 17. Targeted CTI context of DeepJam.

the jamming attack. By properly choosing the slot duration and carefully defining the status and actions, I convert the jamming attack to a time series process. Then I propose a deep learning-guided strategy to predict the attacker's optimal action in the coming slot according to sniffed traffic in the past. Deep learning has been proven more efficient in solving time series problems than traditional machine learning methods and thus is more suitable for DeepJam.

I evaluate the performance of DeepJam with comprehensive comparison with random jamming and reactive jamming in different scenarios. The results show that DeepJam significantly outperforms random jamming in all scenarios. Although reactive jamming performs better than DeepJam in a simple system with one Zigbee device, DeepJam outperforms reactive jamming in terms of efficiency and stealth when there are multiple Zigbee devices, and its advantage becomes more significant with the increase of the network complexity. For example, in a system that contains five Zigbee devices, 42% of DeepJam's jamming actions are necessary, while only 13% of reactive jamming actions are necessary. Due to the random backoff mechanism of the Zigbee network, DeepJam cannot fully jam the traffic of the victim device. But I show that DeepJam can decrease the throughput of the victim by 60% in the worst case and by 78% in the best case. I also propose two simple countermeasures against DeepJam and evaluate their efficacy.

3.2 Background

3.2.1 Zigbee MAC layer

The Zigbee MAC layer is defined in IEEE 802.15.4. The network can be beacon-disabled or beacon-enabled. The former adopts unslotted Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) for media access control, and the latter adopts slotted CSMA/CA. The acknowledgment (ACK) is optional in Zigbee MAC layer. In a beacon-disabled Zigbee network, a device first conducts the Clear Channel Assessment (CCA) to determine the channel status when it attempts to transmit a MAC frame. The device immediately transmits the MAC frame when the channel is idle. Otherwise, the device waits for a period and tries again. The backoff time B_z is defined by an exponential backoff algorithm. In a beacon-enabled network, the coordinator periodically generates beacons which divide the channel to superframes of the same duration. A superframe contains a mandatory active period and an optional inactive period, and the active period is equally divided into 16 slots. During the active period, a device accesses the channel in a similar manner as in the beacon-disabled Zigbee network, while the frame transmission must start at the beginning of one slot.

A Zigbee network can adopt Energy Detection (ED) or Carrier Sensing (CS) for CCA. ED considers the channel busy if the energy level is above a predefined threshold, while CS considers the channel busy only if Zigbee signals are detected. Payload encryption is optional in Zigbee MAC layer, and the Frame Check Sum (FCS) occupying the last 2 bytes of a MAC frame is used to detect bit errors. The MAC frame header, which contains the source and destination MAC addresses, and FCS are transmitted in plaintext.

3.2.2 Cross-Technology Interference (CTI) Illustration

As shown in Fig. 18, the Zigbee and WiFi channels overlap in frequency. For example, the WiFi channel 6 uses the same frequency band as Zigbee channels 16-19. Consequently, a WiFi network working on channel 6 may interfere with a Zigbee network working on channel 18 if those two networks are deployed at proximity.

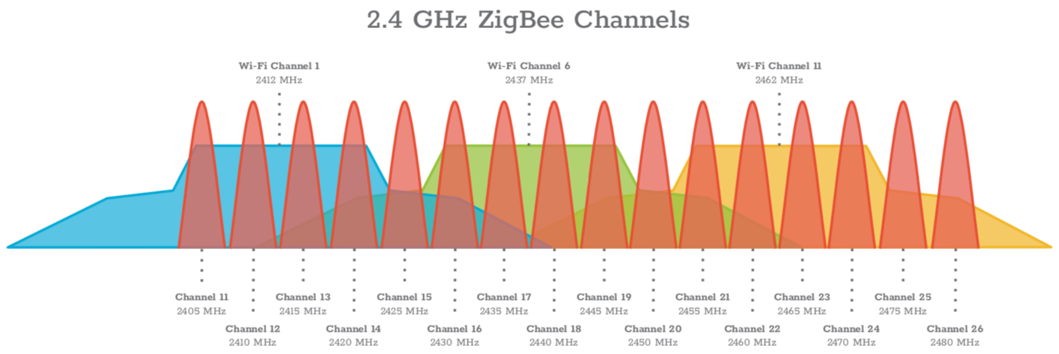


Figure 18. Zigbee and WiFi channels.

Previous studies have experimentally evaluated the interference between WiFi and Zigbee networks (Garroppo et al. 2011; Musaloiu-E and Terzis 2008). Since a WiFi transmitter's power is more than 30 times larger than a Zigbee transmitter's, Zigbee signals' impact on WiFi transmissions is insignificant. In contrast, WiFi signals have significant impact on Zigbee transmissions. Although WiFi signals do not fully jam Zigbee signals, the throughput of the Zigbee network may drop by 60% due to CTI.

3.3 System and Adversary Models

3.3.1 System Model

I consider a system in which a Zigbee network and a WiFi network coexist, and the system model is shown in Fig. 17. The experimental result in (Garroppo et al. 2011) shows that the cross-technology interference between Zigbee and Bluetooth networks is insignificant, so I do not consider Bluetooth networks in my system. The WiFi network follows IEEE 802.11 and contains one AP and multiple WiFi devices. The Zigbee network follows IEEE 802.15.4 and contains one coordinator and multiple Zigbee devices. Without loss of generality, I assume that the WiFi network keeps working on WiFi channel 6 (with a center frequency of 2,437 MHz and bandwidth of 20 MHz), and the Zigbee network keeps working on channel 18 (with a center frequency of 2,440 MHz and a bandwidth of 2 MHz). In this case, the WiFi network interferes with the Zigbee network. IEEE 802.15.4 does not adopt frequency hopping, so the Zigbee network does not hop to another channel due to the interference. Although the WiFi network adopts channel hopping, the Zigbee signal's power is too low to cause significant interference to the WiFi network which thus does not change the channel either.

I make the following assumptions about the Zigbee network. First, the Zigbee network is busy and all the MAC frames are of the maximum length $\mathcal{L}_z$ to maximize the data transmission rate. $\mathcal{L}_z$ equals 127 bytes as defined in 802.15.4. Second, I assume that the network's traffic load and pattern are both stable within a short period (less than 1 min). For simplicity, I assume that the number of Zigbee MAC frames generated (not transmitted) by a Zigbee device within a short period follows the Poisson distribution, and the possibility that a device generates k_z new MAC frames within τ sec is formulated as

$$P(k_z) = \frac{e^{-\lambda_z \tau}(\lambda_z \tau)^{k_z}}{k_z!}, \tag{3.1}$$

where λ_z is the arrival rate. The arrival rate varies with the device's ongoing task, so I only assume that λ_z is stable within a short period.

3.3.2 Adversary Model

I consider an attacker who is aware of the Zigbee channel by passive eavesdropping and attempts to disrupt the traffic from a specific Zigbee device in a stealthy manner. Particularly, the objective of the attacker is to reduce the throughput of the victim device by a target percentage with as few jamming signals as possible. Without loss of generality, I assume that the victim is z_1 whose MAC address is known to the attacker.

To launch the DeepJam attack, the attacker installs a monitor, which can be a COTS Zigbee device, around the victim device. The monitor sniffs the Zigbee traffic, measures the power level within the Zigbee channel, and also decodes any overheard Zigbee MAC frame. Since the header and FCS of a Zigbee MAC frame are not encrypted, the attacker is aware of the source device of the MAC frame and can also verify whether the frame is corrupted by checking its FCS. If the attacker detects an uncorrupted MAC frame of z_1, it considers the frame being successfully transmitted.

Since the Zigbee traffic is assumed stable within a short period, the attacker attempts to predict the victim's traffic in the near future from the signal overheard in the short past period. Specifically, the attacker uses the deep learning model to predict when z_1 may successfully transmit a MAC frame and then generates a predefined jamming signal, which is fixed Gaussian noise in the targeted Zigbee channel, during the predicted period. The power of the jamming signal is five times larger than that of the Zigbee signal, which is sufficient to disrupt Zigbee transmissions but still much weaker than the WiFi signal's power. So the attacker's action has no significant impact on the WiFi network.

After taking a jamming action, the attacker can determine whether the action is necessary. The signal captured by the adversarial monitor is mixed with the jamming signals, but the attacker can subtract the jamming signal from the sniffed signal to restore the original signal. The jamming action is necessary if the restored signal contains z_1's uncorrupted MAC frame and is unnecessary otherwise.

3.4 Problem Formulation

I provide a slotted formulation of the jamming attack and an overview of DeepJam in this section.

3.4.1 Slot Duration

I determine the slot duration based on the maximum MAC frame length $\mathcal{L}_z$ of the Zigbee network. In particular, the slot duration $\mathcal{T}_s$ equals $\mathcal{T}_z/2$, where $\mathcal{T}_z$ denotes the time consumed to transmit a Zigbee MAC frame of length $\mathcal{L}_z$.

The choice of the slot duration is critical. First, the slot must be short enough so that the slot status space is of small size. Multiple MAC frames of the victim may appear in one slot, and the slot's status is defined based on the status of those frames (corrupted or not). A proper slot duration must be short so that only a few MAC frames of the victim may be present in one slot. Second, the slot must be long enough so that the optimal action in a future slot is significantly impacted by only a small number of passing slots.

A slot duration of $\mathcal{T}_z/2$ satisfies the need. Since at most two MAC frames of the victim may appear in a slot, the slot has only three possible status which are detailed in Section 3.4.2. As demonstrated in Section 3.7, the number of slots that may have significant impact on the coming slot is at most 10. The evaluation results show that it is feasible to launch DeepJam in real time with a slot duration of $\mathcal{T}_z/2$.

3.4.2 Slot Status

I define three slot status: **IDLE**, **TRANSMISSION**, and **OCCUPY**, based on the attacker's observation, and I use an example in Fig. 19 to illustrate how I determine the slot status. Fig. 19 shows an example of the wireless traffic within 10 slots. The device transmits during blue periods and keeps silent during white periods. For simplicity, I assume that only three devices, z_1 (the victim Zigbee device), z_2 (another Zigbee device), and w_1 (a WiFi device), transmit within those 10 slots. The attacker monitors the wireless signal within the Zigbee channel, measures the power level within each slot, and decodes the signal captured in each slot.

In slot$_{i-9}$, the power level is always below a threshold, and the attacker can conclude that there is no wireless transmission within this slot and considers the slot **IDLE**. In slot$_{i-8}$ and slot$_{i-1}$, the attacker can detect a MAC frame header containing z_1's address, but the frame does not fully occupy the slot. Those two slots are also considered **IDLE**. If the attacker detects a complete and uncorrupted MAC

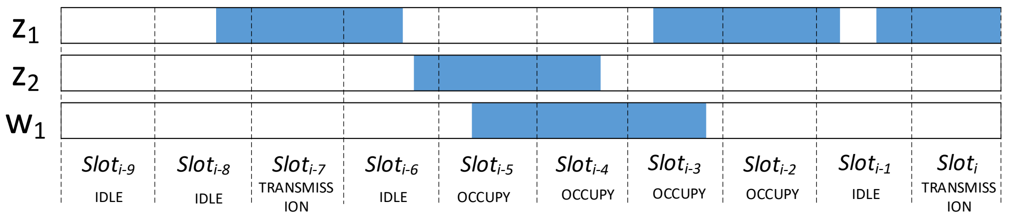


Figure 19. Slot status.

frame of z_1, the first slot fully occupied by the frame, such as slot_{i-7}, is considered **TRANSMISSION**, and the following slot, such as slot_{i-6}, is considered **IDLE**. Apart from **IDLE** and **TRANSMISSION** slots, all the rest slots are considered **OCCUPY**.

It is worth noting that the attacker may be uncertain about the status of a slot in some cases. For example, the attacker cannot determine the status of slot_i because FCS has not been received yet. In this case, the attacker temporally considers slot_i a **TRANSMISSION** slot.

Three cases can result in an **OCCUPY** slot. First, z_1 does not transmit, but Wi-Fi devices or other Zigbee devices transmit in this slot. slot_{i-5} and slot_{i-4} belong to the first case. Second, z_1 transmits, but Wi-Fi devices transmit simultaneously. Due to interference, the frame of z_1 cannot be correctly decoded. slot_{i-3} belongs to this case. Third, only z_1 transmits in this slot, but the frame's header is within the previous slot and cannot be decoded due to interference. Consequently, the monitor, as well as the destination device, cannot decode the source address correctly to recognize the frame as z_1's. slot_{i-2} belongs to this case.

3.4.3 DeepJam Basics

The main idea of DeepJam is predicting the optimal action in a future slot based on the observations in past slots. It takes time to process the captured signal and compute the prediction. Therefore, the attacker cannot immediately obtain the status of a slot and make a proper prediction at the beginning of the next slot. The experimental result in Section 3.7 shows that the signal processing and prediction computation can be done within one slot. Therefore, I add a gap of one slot between the most recent channel observation and the prediction. To be more specific, within the ith slot, the attacker first processes the signal captured in slot $i-1$. Then the attacker uses the observed channel status before and in slot $i-1$ to predict the optimal action he should take in slot $i+1$.

The attacker can take two actions, **WAIT** or **JAM**, in each slot. If the attacker predicts that the coming slot is a **TRANSMISSION** slot, the attacker takes action **JAM** by sending jamming signals in this slot. Otherwise, the attacker takes action **WAIT** and keeps silent. Since no error correction code is adopted, jamming half of the MAC frame, i.e., one slot, can disrupt the frame with a high probability. Attacker's actions also impact the channel status. If the attacker takes action **JAM** in slot i, the channel status is **OCCUPY** no matter whether there is wireless traffic or not. As I discussed in Section 3.3, the attacker can restore the original signal, so he can determine whether his prediction for an ended slot is correct and thus amend the prediction model.

3.5 DeepJam Details

DeepJam follows an online-learning process. In particular, the attacker takes actions based on a prediction model and updates the model based on the results of actions. It is well known that Reinforcement Learning (RL) (Sutton and Barto 2018) is an effective solution to online-learning problems, so I adopts RL in DeepJam. This section first briefly introduces RL and explains how I convert DeepJam to a typical RL problem. Then I detail the DeepJam RL algorithm.

3.5.1 Reinforcement Learning

RL considers an agent who interacts with the environment in a sequence of time slots and tries to maximize some notion of a cumulative reward (Sutton and Barto 2018). More specifically, the agent observes the state of the environment in the ith slot, denoted by s_i. Based on the observation, the agent takes an action $a_i \in \mathcal{A}_{s_i}$, where $\mathcal{A}_{s_i}$ is the set of possible actions in state s_i. As a result of the action, the agent gets a reward r_{i+1}, and the environment state changes to s_{i+1}. The goal of the agent is to maximize the cumulative reward $R_i \triangleq \sum_{j=i}^{\infty} \gamma^{j-i} r_{j+1}$, where $\gamma \in (0, 1]$ is a discount factor.

To convert DeepJam to a typical RL problem, I define the slot states, the attacker's actions, and the rewards as follows.

3.5.1.1 Slot States

Due to the MAC method, the channel status and the corresponding optimal action of the ith slot are related to the previous slots. I thus abuse the notation and use s_i to represent the state of slot i and refer to the channel status of all the slots that significantly impact the optimal action in slot i. As shown in Section 3.4, the attacker cannot immediately obtain the status of a slot and complete the prediction at the beginning of the next slot. So s_i does not contain the channel status in slot $i-1$. If I consider N_R slots before slot i except slot $i-1$, s_i can be formulated as a $1\times N_R$ vector $< u_{i-N_R-1}, \cdots, u_{i-2} >$. Here $u_j \in \{1,2,3\}$ indicates the channel status of slot j and is defined as follows:

$$u_j = \begin{cases} 1, & \text{if slot } j \text{ is } \textbf{IDLE}; \\ 2, & \text{if slot } j \text{ is } \textbf{OCCUPY}; \\ 3, & \text{if slot } j \text{ is } \textbf{TRANSMISSION}. \end{cases} \tag{3.2}$$

A large N_R can achieve a high prediction accuracy but results in long converge time. Since the temporal pattern of the wireless traffic comes from the exponential backoff algorithm, I choose N_R based on the maximum backoff periods of the WiFi and Zigbee networks. Particularly, I calculate N_R as:

$$N_R = \lceil \max(\mathcal{D}_w, \mathcal{D}_z)/\mathcal{T}_s \rceil - 1, \tag{3.3}$$

where $\mathcal{D}_w$ and $\mathcal{D}_z$ denote the maximum backoff periods in the WiFi and Zigbee networks, respectively; $\mathcal{T}_s$ denotes the duration of a time slot.

3.5.1.2 Actions

I denote the attacker's action in slot i with a_i, where $a_i \in \{\mathbf{JAM}, \mathbf{WAIT}\}$. The details of **JAM** and **WAIT** actions have been given in Section 3.4.

3.5.1.3 Rewards

I define the reward of action a_i based on the channel status of slot i. To be more specific, if $a_i = \mathbf{JAM}$ and slot i is **TRANSMISSION**, the attacker gets a reward $r_{i+1} = R_j \in (0, 1]$ for successfully jamming a MAC frame of the victim. If $a_i = \mathbf{WAIT}$ and slot i is **IDLE** or **OCCUPY**, the attacker gets a reward $r_{i+1} = R_s \in [0, 1])$ for saving energy. If $a_i = \mathbf{WAIT}$ and slot i is **TRANSMISSION**, the attacker misses a successfully transmitted MAC frame of the victim and receives a negative reward $r_{i+1} = R_m \in [-1, 0)$. If $a_i = \mathbf{JAM}$ and slot i is **IDLE** or **OCCUPY**, the attacker wastes energy on an unnecessary jamming and gets a negative reward $r_{i+1} = R_w \in [-1, 0]$.

Attackers can adjust R_j, R_s, R_m, and R_w based on their own constraints. For example, if the energy limitation is a big concern, the attacker can adopt large absolute values for R_w and R_s and adopt a small absolute value for R_m. Since hindering the wireless communication should always be the primary goal of the attacker, R_j and R_m cannot be zero.

3.5.2 RL Algorithm of DeepJam

DeepJam adopts Q-Learning (QL) (Watkins 1989), which is a popular RL algorithm, to determine the optimal action in a future slot. QL aims to obtain a Q-function $Q(s, a) \triangleq \mathbb{E}[R_i | s_i = s, a_i = a]$ (also called Q-value) to calculate the expected maximum cumulative reward of taking action a at state s. The action with the maximum Q-value is considered the optimal.

Traditional QL obtains Q-function in a tabular manner which results in long converge time and thus is not suitable for DeepJam. More specifically, the QL algorithm must go through all the combinations of slot states and actions to obtain the Q-function. For example, consider the scenario in Section 3.7 where the slot state contains the channel status of 10 slots with the slot duration around 2 ms. The QL algorithm takes at least $2 \text{ ms} * 2 * 3^{10} \approx 4$ min to converge and obtain the Q-function. The wireless traffic is highly dynamic, and the temporal pattern is very likely to have changed before the algorithm converges. So DeepJam cannot use traditional QL.

I adopt Deep Q-Learning (DQL) (Goodfellow, Bengio, and Courville 2016) to deal with the large state space and approximate the Q-function with a tailored Deep Neural Network (DNN). The input to the DNN is the slot state, and the outputs are the Q-values of taking **JAM** and **WAIT** actions in the slot. Fig. 20 shows the structure of DeepJam DNN that contains one LSTM cell and two Fully Connected (FC) Layers. I also show the input and output dimensions of each layer in the figure.

The LSTM cell (Hochreiter and Schmidhuber 1997) can capture long-time temporal patterns of the wireless traffic. As mentioned in Section 3.5.1.1, the slot state s_i contains the channel status of slots $i - N_R - 1$ to $i - 2$. Since the slots earlier than slot $i - N_R - 1$ may also have non-trivial impact on the optimal action in slot i, I use

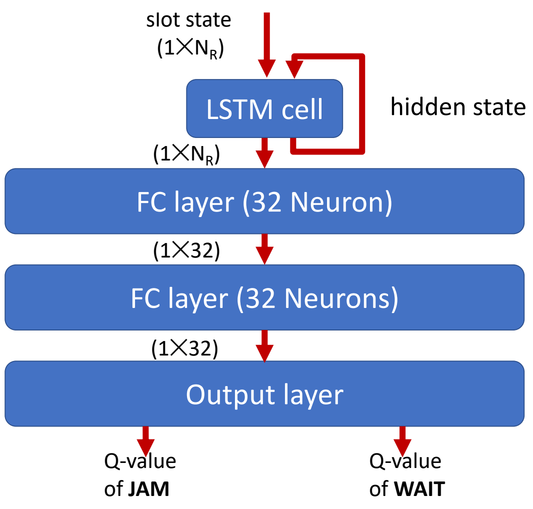


Figure 20. DeepJam DNN architecture.

the LSTM cell to memorize those early slots without increasing the dimension of the slot state. In particular, the LSTM cell contains four gates to maintain a hidden state and calculate the output (Hochreiter and Schmidhuber 1997). The hidden state is affected by the long-time history of inputs and is updated iteratively based on the newly coming input. The four gates control how the hidden state is affected by the newly coming input and how the output is affected by the hidden state. Therefore, the long-time history of the wireless traffic is memorized by the hidden state of the LSTM cell and contributes to the prediction of optimal actions. The input and output of the LSTM cell are all $1 \times N_R$ vectors. The output of the LSTM cell is fed into two FC layers, both of which contain 32 neurons and adopt the Rectified Liner Unit (ReLU) function as the active functions. The output layer is a linear FC layer that outputs the Q-values of **JAM** and **WAIT** actions.

DeepJam DNN is an estimation of the Q-function, and the training process iteratively reduces the estimation error. The observation on slot i, including the state, the action, and the reward, is one training sample, which is denoted by $o_i =< s_i, a_i, r_{i+1} >$. I adopt the techniques of separate target network and batch gradient descent to stabilize the training process. I maintain a target network whose structure is the same as that of DeepJam DNN. The Q-values given by DeepJam DNN and the target network are denoted by $q(s, a; \Theta_Q)$ and $q'(s, a; \Theta'_Q)$, respectively. Here Θ_Q and Θ'_Q denote the parameter vectors of DeepJam DNN and the target network, respectively; s denotes the slot state; a denotes the action. With o_i as the training sample, the loss function is defined as follows:

$$L_Q(i,\ \Theta_Q) = (r_{i+1} + \gamma \max_a q'(s_{i+1}, a; \Theta'_Q) - q(s_i, a_i; \Theta_Q))^2 \tag{3.4}$$

$$\mathrm{Loss} = (\mathrm{Reward_{i+1}} + \mathrm{D} \cdot \max_{\mathrm{action}} \mathrm{Q}'(\mathrm{s_{i+1}}, \mathrm{action}) - \mathrm{Q}(\mathrm{s_i}, \mathrm{action_i}))^2 \tag{3.5}$$

where $\gamma \in (0, 1]$ is the discount factor. In each iteration, I update DeepJam DNN's parameter vector with a batch containing five training samples, and the batch is denote by $b =< o_{I(1)}, o_{I(2)}, o_{I(3)}, o_{I(4)}, o_{I(5)} >$, where $I(j)$ is the slot index of the jth training sample. I update Θ_Q as

$$\Theta_Q \leftarrow \Theta_Q - \frac{\rho_Q}{5} \sum_{j=1}^{5} \frac{\partial L_Q(I(j),\ \Theta_Q)}{\partial \Theta_Q} \tag{3.6}$$

every iteration and replace Θ'_Q with Θ_Q every N_Θ iterations. Here ρ_Q is the learning rate.

I adopt experience replay (Mnih et al. 2015) to accelerate the training process. More specifically, I maintain a memory pool containing the observations on the latest 500 slots. In the training process, I select 20 batches that are continuous in time from the memory pool. To be more specific, if the first batch is $b_1 =< o_{I(1)}, o_{I(2)}, o_{I(3)}, o_{I(4)}, o_{I(5)} >$, the mth batch is $b_m =< o_{I(1)+m-1}, o_{I(2)+m-1}, o_{I(3)+m-1}, o_{I(4)+m-1}, o_{I(5)+m-1} >$. I use the 20 batches to update the DNN parameter vector successively, and the whole process is referred to as one epoch. An attacker can conduct multiple epochs in one slot according to his computational capacity. To capture the long-time temporal pattern, I only initialize the hidden state of the LSTM cell at the beginning of each epoch. I also update the target network at the beginning of each epoch and keep it stable within one epoch. The memory pool is updated at the beginning of each slot. I only take **WAIT** actions in the first $21 + N_R$ slots to collect training samples, and I use all the past slots as the memory pool from slot $21 + N_R$ to slot $501 + N_R$. Here N_R is still the aforementioned dimension of the slot state.

In case that DeepJam DNN gets stuck before converging to the optimal estimation of the Q-function, I choose actions in an ϵ-greedy manner. In particular, I take action $a = \arg\max_a q(s_i, a; \Theta_Q)$ in slot i with a probability of $1 - \epsilon$ and take the other action with a probability of ϵ. The hidden state of the LSTM cell is calculated with the states of the 20 slots that are ahead of slot i. The pseudocode of the DeepJam RL algorithm is given in Algorithm 1.

3.6 Countermeasures

I propose two countermeasures against DeepJam for networks with different capabilities.

Networks with advanced devices which have powerful computing capacity can defeat DeepJam by adopting deep learning-based MAC protocols. The victim device can learn the attacker's behavior pattern with DNN and thus predict the slots that are less likely to be jammed. A more powerful victim can train a defense DNN in advance with Generative Adversarial Network (GAN) (Goodfellow et al. 2014) which can even mislead or manipulate the attacker's behaviors. Specifically, the victim jointly trains

Algorithm 1: RL Algorithm of DeepJam

```
Initialize $\epsilon$, $\rho_Q$, $\gamma$, $N_R$
Initialize memory pool $M$
Initialize LSTM hidden state $h$, DNN parameter vector $\Theta_Q$
Copy $\Theta_Q$ to target network parameter vector $\Theta'_Q$
for slot $i$ in DeepJam do
  if $i \leq 21 + N_R$ then
    take action WAIT
  else
    Calculate LSTM hidden state $h_i$
    Input $s_i$ to DNN and output $q(s_i, a; \Theta_Q)$ for $a \in \{\mathbf{JAM}, \mathbf{WAIT}\}$
    Take action according to $\epsilon$-greedy policy
    for each epoch do
      Initialize LSTM hidden state
      Randomly selected 20 continuous batches $\mathcal{B}$
      for each batch in $\mathcal{B}$ do
        $\Theta_Q \leftarrow \Theta_Q - \frac{\rho_Q}{5} \sum_{j=1}^{5} \frac{\partial L_Q(I(j),\ \Theta_Q)}{\partial \Theta_Q}$
      end for
      $\Theta'_Q \leftarrow \Theta_Q$
    end for
  end if
  Process the signal obtained in slot $i - 1$
  Update $s_{i-1}$, $a_{i-1}$, and $r_i$ to $M$
end for
```

two DNNs, including an attack DNN and a defense DNN, in an adversarial manner. Both DNNs take the traffic history as inputs. The attack DNN simulates the attacker and predicts the optimal time to generate the next jamming signal just as DeepJam does. The defense DNN predicts the optimal time to transmit the next MAC frame and attempts to maximize the throughput of the victim. After sufficient rounds of competitions, the defense DNN can capture the behavior pattern of the attacker, and the victim can partially mitigate the attack's impact by acting according to the defense DNN's prediction. However, most COTS IoT devices only have limited computing capacity and thus cannot adopt the DNN-based countermeasure.

I also propose the dynamic network configuration as a simple yet effective countermeasure against DeepJam. As demonstrated in Section 3.7, a sudden change of wireless traffic can immediately and significantly harm the performance of DeepJam, and it takes time for DeepJam DNN to converge again. The victim network can introduce sudden changes to the wireless traffic by changing the network configuration frequently. Consequently, the DeepJam DNN may never converge, and thus the impact of DeepJam is weakened. my evaluation results show that the dynamic network configuration can indeed harm the performance of DeepJam. However, the dynamic network configuration may bring extra management burden to the network.

3.7 Evaluation

This section first introduces the evaluation setup, including the hardware and software, evaluation metrics, parameter setting, and comparison method. Then I evaluate my scheme in various scenarios. Since Zigbee networks in different application scenarios may differ in MAC and CCA methods, I evaluate DeepJam's performance under different Zigbee network configurations. I also evaluate the impacts of the WiFi network's traffic load and the Zigbee network size. Finally, I evaluate the latency of DeepJam and the efficiency of the countermeasure.

3.7.1 Evaluation Setup

I implemented DeepJam DNN with PyTorch 1.4 (*PyTorch* 2004). To make minimum assumptions about the attacker's ability, I conducted all the experiments on a COTS personal computer. The computer is equipped with an Intel Core i7-3770 3.4 GHz CPU, where all the computations Ire conducted.

I used two metrics to evaluate DeepJam. The first is **hit rate** (HR) defined as

$$\mathrm{HR} = \mathcal{N}_h/\mathcal{N}_j, \tag{3.7}$$

where $\mathcal{N}_j$ denotes the number of slots that are predicted to be **TRANSMISSION** and thus jammed by the attacker; $\mathcal{N}_h$ denotes the number of the victim's MAC frames that are corrupted not by the wireless signal of other devices but by the jamming signal. In other words, $\mathcal{N}_h$ predictions among the $\mathcal{N}_j$ **TRANSMISSION** predictions are correct, so HR reflects the prediction accuracy of the DeepJam DNN.

I define the second metric **jam rate** (JR) as

$$\mathrm{JR} = \mathcal{N}_h/(\mathcal{N}_h + \mathcal{N}_s), \tag{3.8}$$

where $\mathcal{N}_s$ denotes the number of the victim's MAC frames that are successfully transmitted.

HR measures the ratio of necessary jamming actions, and JR measures the victim's throughput decrease due to jamming. The jamming attack is said to be efficient if both HR and JR are high; i.e., the attacker can significantly reduce the victim's throughput with limited energy consumption.

I adopted 0.8, 0.1, -0.8, and -0.3 as R_j, R_s, R_m, and R_w, respectively. The discount factor and the learning rate are 0.9 and 0.01, respectively. I trained the DNN for five epochs in each slot. The slot duration $\mathcal{T}_s$ equals 2,128 μs, and slot state's dimension N_R equals 10.

I compare DeepJam with conventional random jamming and reactive jamming as follows. I first launch DeepJam for 10,000 slots among which $\mathcal{N}_j^d$ slots are jammed and calculate the HR and JR, denoted by HR_d and JR_d. Then I launch random jamming and reactive jamming each for 10,000 slots, aiming to achieve the same jam rate as DeepJam's. To be more specific, I jam each slot with a probability of JR_d in random jamming and jam the slot that follows a slot containing a Zigbee preamble with a probability of JR_d in reactive jamming. The number of jamming actions token in random jamming and reactive jamming are denoted by $\mathcal{N}_j^{ra}$ and $\mathcal{N}_j^{re}$, respectively. I also calculate the hit rates of random jamming and reactive jamming, denoted by HR_{ra} and HR_{re}, respectively. I define a metric **jamming-efficiency gain** (JEG) for more straightforward comparison. The JEGs of DeepJam over random jamming and reactive jamming are defined as $\mathrm{JEG}_{d/ra} = \mathcal{N}_j^{ra}/\mathcal{N}_j^{d}$ and $\mathrm{JEG}_{d/re} = \mathcal{N}_j^{re}/\mathcal{N}_j^{d}$, respectively. JEG measures the advantage of DeepJam over random jamming and reactive jamming in terms of the jamming actions' efficiency.

3.7.2 Efficiency of DeepJam

In this subsection, I first compare DeepJam, random jamming, and reactive jamming in a specific scenario. Then I evaluate the impact of multiple parameters.

3.7.2.1 Performance comparison

The initial configuration of the experiment was as follows. The arrival rate of the WiFi network was $\lambda_w = 500$ fps. The Zigbee network contained three Zigbee devices, was beacon-disabled, and adopted CS for CCA. I changed the configuration at slot 5,000 by adjusting λ_w to 300 fps. The HRs and JRs calculated for every 200 slots are shown in Fig. 21.

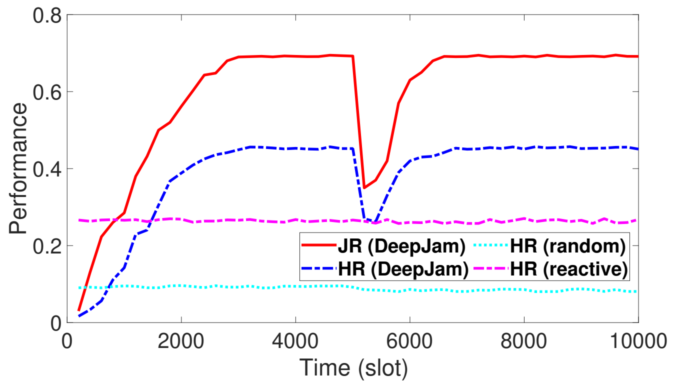


Figure 21. Comparison of DeepJam, random jamming, and reactive jamming.

With the initial configuration, DeepJam converged after around 3,100 slots (about 6.3 sec) and achieved an HR_d of 0.43 and a JR_d of 0.68. The performance of DeepJam dramatically decreased around slot 5,000 due to the sudden change of wireless traffic, but DeepJam converged again after about 3.3 sec and achieved an HR_d of 0.44 and a JR_d of 0.69. With short converge time, DeepJam can handle the highly dynamic wireless traffic. DeepJam significantly outperformed random jamming and reactive jamming with an $\text{JEG}_{d/ra}$ of 4.9 and an $\text{JEG}_{d/re}$ of 1.7.

To better understand the comparison between DeepJam and reactive jamming, it is worth noting that reactive jamming is triggered by the detection of Zigbee preambles. So reactive jamming wastes energy on jamming Zigbee traffic which is not from the victim device. Besides, some corrupted Zigbee packets contain correct preambles which also trigger a reactive jammer to launch unnecessary jamming. Therefore, DeepJam improved the HR of reactive jamming by more than 1.7 times because it can identify the uncorrupted packets of the victim more accurately.

3.7.2.2 Impact of WiFi traffic loads

In the experiment, the Zigbee network contained three Zigbee devices, was beacon-disabled, and adopted CS for CCA. I first evaluated DeepJam with a congesting WiFi network ($\lambda_w = 1,000$ fps) and then repeated the evaluation with the busy and idle WiFi networks ($\lambda_w = 500$ fps and $\lambda_w = 100$ fps, respectively). For each evaluation, I launched DeepJam for 50 times. The average converge time and the average HR_d and JR_d after convergence are shown in Table 5. The table also lists HR_{ra}, HR_{re}, $\text{JEG}_{d/ra}$, and $\text{JEG}_{d/re}$ for comparison.

λ_w	1,000 fps	500 fps	100 fps
HR_d	0.12	0.43	0.45
JR_d	0.30	0.67	0.67
Converge time (sec)	15.5	6.9	6.7
HR_{ra}	0.03	0.07	0.09
HR_{re}	0.14	0.23	0.26
$\text{JEG}_{d/re}$	4.0	6.1	5.0
$\text{JEG}_{d/re}$	0.86	1.8	1.7

Table 5. Performance with different WiFi traffic loads.

DeepJam performs well with the busy and idle WiFi networks, but its performance in a congested WiFi network is less satisfactory. I found that 95% of the victim's frames were corrupted due to the interference of the congested WiFi network. So the training samples, i.e., slots, containing uncorrupted transmissions were few, which resulted in long converge time and poor performance. However, the throughput of the victim is extremely low when it coexists with a congested WiFi network, so it does not make sense for the attacker to launch the jamming attack in such a scenario. DeepJam performs similarly with busy and idle WiFi networks. So I only considered the busy WiFi network hereafter, which is more common in practice.

3.7.2.3 Impact of MAC and CCA methods

The Zigbee network in this experience contained three Zigbee devices, and the WiFi network was busy. I considered four scenarios: a beacon-enabled Zigbee network adopting CS (scenario 1), a beacon-enabled Zigbee network adopting ED (scenario 2), a beacon-disabled Zigbee network adopting CS (scenario 3), and a beacon-disabled Zigbee network adopting ED (scenario 4). I repeated DeepJam 50 times in each scenario and calculated the average converge time, HRs, and JRs. The results are shown in Table 6. The table also lists the corresponding HR_{ra}, HR_{re}, $\text{JEG}_{d/ra}$, and $\text{JEG}_{d/re}$ for comparison.

	Scenario 1	Scenario 2	Scenario 3	Scenario 4
HR_d	0.37	0.40	0.45	0.44
JR_d	0.65	0.64	0.72	0.72
Converge time (sec)	7.6	7.1	6.7	6.7
HR_{ra}	0.05	0.05	0.08	0.07
HR_{re}	0.23	0.23	0.25	0.24
$\text{JEG}_{d/ra}$	7.4	8.0	5.6	6.2
$\text{JEG}_{d/re}$	1.6	1.7	1.8	1.8

Table 6. Performance with different MAC and CCA methods.

In all scenarios, DeepJam significantly outperformed random jamming and reactive jamming in terms of energy efficiency and converged within 8 sec. CCA methods had no significant impact on DeepJam, while the performance of DeepJam with beacon-disabled Zigbee networks was slightly better than that with beacon-enabled ones. The main reason is that slotted CSMA/CA is more complex than unslotted CSMA/CA. In particular, the end of the backoff period is synchronized with slot boundaries, and the device may conduct CCA multiple times before transmission even though the channel is idle, resulting in more chaotic wireless traffic. So DeepJam converged slower with beacon-enabled Zigbee networks, and HR_d and JR_d were both lower as well. To evaluate DeepJam in the worst case, I considered beacon-enabled Zigbee networks adopting CS hereafter.

3.7.2.4 Impact of Zigbee devices' amount $\mathcal{N}_D$

This experiment considered a beacon-enabled Zigbee network adopting CS and an busy WiFi network. I evaluated $\mathcal{N}_D$ equal to 1, 2, 3, 4, or 5. For each value, I launched DeepJam 50 times. The average HRs, JRs, and converge time are shown in Table 7. The table also lists the corresponding HR_{ra}, HR_{re}, $\mathrm{JEG}_{d/ra}$, and $\mathrm{JEG}_{d/re}$ for comparison.

$\mathcal{N}_D$	1	2	3	4	5
HR_d	0.39	0.41	0.37	0.39	0.37
JR_d	0.62	0.65	0.65	0.62	0.60
Converge time (sec)	6.3	6.3	6.7	6.7	6.3
HR_{ra}	0.23	0.19	0.08	0.06	0.05
HR_{re}	0.71	0.32	0.24	0.18	0.13
$\mathrm{JEG}_{d/ra}$	1.7	2.2	5.6	6.5	7.4
$\mathrm{JEG}_{d/re}$	0.55	1.3	1.8	2.2	2.8

Table 7. Performance with different numbers of Zigbee devices.

DeepJam outperformed random jamming in all scenarios, and the advantage of DeepJam over reactive jamming became more significant with the increase of $\mathcal{N}_D$ (especially for $\mathcal{N}_D \geq 3$). DeepJam achieved an $\mathrm{JEG}_{d/re}$ of 2.8 with $\mathcal{N}_D$ equaled 5. With the increase of $\mathcal{N}_D$, reactive jamming wasted more energy on jamming Zigbee traffic that was not from the victim, thus decreasing the HR_{re}. In contrast, the impact of $\mathcal{N}_D$ on DeepJam was negligible. HR_d, JR_d, and the converge time of DeepJam were stable for different $\mathcal{N}_D$.

3.7.3 Latency

This section evaluated the latency of DeepJam. To launch DeepJam in real time, the attacker needs to process the data captured in the previous slot, calculate the output of the deep neural network, and update the deep neural network within one slot, i.e., 2,128 μs. I denote the time taken to process the captured data, calculate the DNN output, and update the DNN by T_p, T_c, and T_u, respectively. As mentioned in Section 3.5, the attacker can finish the three tasks in parallel, so DeepJam is feasible in real time if $\max(T_p,\ T_c,\ T_u) \leq 2,128\ \mu s$. As defined in 802.15.4, the interval between two continuous MAC frames is 40 symbols during which the Zigbee device can process a MAC frame. I assume that the attacker uses a COTS Zigbee receiver as the monitor, so T_p is no more than 40 symbols, i.e., 640 μs. In all the aforementioned experiments, the maximum values of T_c and T_u are 10 μs and 1,870 μs, respectively. Both are less than 2,128 μs. Therefore, the real-time DeepJam attack is feasible.

3.7.4 Efficiency of Dynamic Network Configuration

Finally, I evaluated the efficacy of the dynamic network configuration countermeasure. This section reports the results for two configurations, the backoff unit and the superframe length. I considered a beacon-enable Zigbee network that adopts CS and contained 3 Zigbee devices in this experiment.

I first evaluated the efficacy of dynamic backoff units. The backoff units of the Zigbee network were chosen from $\{20, 40, 60, 80, 100\}$ samples, and the network randomly changed the backoff unit every 2 sec. I launched DeepJam in this scenario for 20 min. The average HR_d and JR_d were only 0.21 and 0.37, respectively; and the

JEG$_{d/ra}$ and JEG$_{d/re}$ were 2.3 and 0.78, respectively. Then I evaluated the dynamic superframe length, which was randomly chosen from {10,240, 15,360, 20,480, 25,600} μs every 2 sec. I also launched DeepJam in this scenario for 20 min. The average HR$_d$ and JR$_d$ were only 0.16 and 0.23, respectively; and JEG$_{d/ra}$ and JEG$_{d/re}$ were 1.7 and 0.59, respectively.

The performance of DeepJam with dynamic network configurations was much worse than that with stable network configurations. Due to the network-configuration changes, the wireless traffic pattern always changed before DeepJam DNN converged, which effectively weakened DeepJam's impact. The evaluation results show that dynamic network configuration is indeed an effective countermeasure against DeepJam.

3.8 Related Work

With the surge of IoT devices, the cross-technology interference has become a critical problem. There have been many studies evaluating the impact of the cross-technology interference (Garroppo et al. 2011; Musaloiu-E and Terzis 2008) or exploring new media access control schemes to mitigate it (Yu, Wang, and Liew 2019; Zhang and Shin 2011). In this chapter, I focus on another negative consequence of the cross-technology interference, which makes the wireless network more vulnerable to the jamming attack.

There has been significant research on jamming attacks and defenses. Traditional jammers can be classified into constant jammers, deceptive jammers, random jammers, and reactive jammers (Pelechrinis, Iliofotou, and Krishnamurthy 2010; W. Xu et al. 2005). A reactive jammer only disrupts the targeted channel upon detection of a specific signal such as the Zigbee MAC frame header, so it significantly outperforms other categories of jammers in terms of efficiency. In spite of the strict real-time requirement, researchers have successfully implemented the reactive jammer in the real environment (Wilhelm et al. 2011). However, the reactive jammer becomes less efficient in the CTI context. Specifically, the detection-based jamming strategy would waste energy on jamming the victim's transmissions which have been disrupted by CTI and also the transmissions of other devices.

With the surge of deep learning, there have been recent studies adopting deep neural networks to launch or mitigate the jamming attack (Shi, Sagduyu, et al. 2018; Slimeni et al. 2015; Machuzak and Jayaweera 2016). The work in (Shi, Sagduyu, et al. 2018) is the most related to DeepJam. Shi *et al.* (Shi, Sagduyu, et al. 2018) considered a slotted cognitive network in which the victim accesses the channel with a deep learning-based media access control method. They proposed a generative adversarial network-based jamming strategy which is efficient as shown in their experimental result. Unlike the scenario considered in (Shi, Sagduyu, et al. 2018), I consider a more realistic, chaotic scenario with cross-technology interference, in which an unslotted Wi-Fi network coexists with an unslotted Zigbee network.

3.9 Conclusion

This chapter presents the design of DeepJam, a deep learning-guided jamming strategy which exploits CTI in complex wireless environments. Detailed evaluations confirm that DeepJam is more stealthy and energy-efficient than conventional random and reactive jamming strategies. I also propose two effective countermeasures against DeepJam.

Chapter 4

DRONEKEY: DRONE-AIDED DL-POWERED SCALABLE GROUP-KEY ESTABLISHMENT FOR LARGE-SCALE IOT NETWORKS

4.1 Overview

The Internet of Things (IoT) networks are finding massive applications in mission-critical contexts, such as critical infrastructure monitoring, border control and protection, military reconnaissance, and surveillance. For example, an IoT network containing thousands of devices is being used to improve the management effectiveness of the Pendjari National Park over 2,755 km^2. Intelligent factories, such as the Tesla factory, use IoT networks with thousands of or more devices in a large factory area to facilitate manufacturing process workflows. Moreover, the military is actively exploring IoT networks for battlefield reconnaissance and border control. These IoT networks are all large-scale in terms of both the coverage area and device count. In this chapter, I consider such a large mission-critical IoT network formed by many distributed groups, each comprising many densely deployed nearby IoT devices. These devices communicate over the open wireless channel and frequently exchange broadcast/multicast messages with group peers. So it is necessary to explore a unique group key to secure broadcast/multicast messages in each group. This chapter focuses on investigating sound schemes to establish/update the group key of each individual group in such a large-scale IoT network.

Designing group-key generation (GKG) schemes for large-scale mission-critical IoT networks faces some essential challenges. First, each group may contain several

	Group size	**Key-generation rate**	**Randomness test**	**Real-world evaluation**
DroneKey	10	89.5 bit/sec	Passed	Yes
	50	50.0 bit/sec		
	100	36.2 bit/sec		
Liu et al.(Liu et al. 2014)	10	40 bit/sec	Passed	Yes
Wei et al.(Wei, Zhu, and Ni 2012)	3	80 bit/sec	Unknown	Yes
Thai et al.(Thai, Lee, and Quek 2015)	4	12 bit/sec	Unknown	No
Xu et al. (P. Xu et al. 2016)	4	9.4 bit/sec	Unknown	No

Table 8. The comparison between DroneKey and representative prior work.

tens to hundreds of IoT nodes in a densely deployed network. So the GKG scheme should be highly *scalable* to an arbitrary group size. Second, the IoT network can be totally unattended in remote non-accessible areas, last very long time, and transmit time-sensitive information. So the GKG scheme should be *fast* in quickly updating all the group keys, which can translate into the requirement for a high group-key generation rate. Third, but not the last, the IoT devices are mostly likely to be battery-powered. So the GKG scheme should be *efficient* with low communication and computation overhead.

Key generation based on wireless physical-layer (PHY) channel characteristics has received tremendous attention as alternative methods to cryptographic techniques. Most PHY-based methods such as (Mathur et al. 2008; Jana et al. 2009; Liu et al. 2013) target pairwise-key generation between two wireless devices. In contrast to cryptographic techniques based on computational hardness assumptions, these methods rely on *channel reciprocity* which refers to that two wireless devices can observe highly correlated variations of the wireless channel between them. Such correlated channel variations can be used as a common randomness factor for extracting a pairwise key between the two devices. As long as the eavesdropper is at least half wavelength away from legitimate devices, it cannot observe the same channel variations for inferring the pairwise key (Sayeed and Perrig 2008; Tope and McEachen 2001).

PHY-based GKG has also been studied as alternative methods to cryptographic group-key generation. The most intuitive solution is to generate many PHY-based pairwise keys to spread a group key across the group. The number of involved pairwise keys must be greater than the number of devices and can be as large as hundreds in my context. To defeat sophisticated attacks, each group key in a large-scale mission-critical IoT network must be frequently updated, so are the involved massive pairwise keys. The resulting computation and communication overhead is very high, making this solution inefficient and impractical. Efficient PHY-based GKG is challenging because the wireless channel variations between any two devices cannot be measured by any other device more than half-wavelength away from both devices due to channel reciprocity. It is nearly impossible to find a common wireless channel for many distributed devices to extract a group key. Some recent schemes (Liu et al. 2014; Thai, Lee, and Quek 2015; Jagadeesh, Joshi, and Rao 2021) try different ways to spread the measurements of selected channels to the entire group of devices. These schemes all require each device to transmit at least one probe packet, and all the packet transmissions must be finished within the short channel-coherence time which refers to the time duration in which the channel condition is considered non-varying. So these schemes are not scalable to a large group. For example, the largest group size reported in (Liu et al. 2014) is only 10. In addition, the group-key generation rate highly depends on the channel randomness and is often not satisfactory. The technique in (Liu et al. 2014) is the only one I beware that has passed the NIST randomness test and been evaluated in real environments. Its group-key generation rate is about 40 bit/sec according to their experimental setup with only 10 devices.

In this chapter, I propose **DroneKey**, a novel drone-aided PHY-based GKG scheme for large-scale mission-critical IoT networks, which is highly scalable, fast, and efficient. DroneKey explores drones which are increasingly popular and widely expected to be prevalent equipment in mission-critical IoT systems. Whenever group-key establishment/rekeying is needed, one or a few drones are dispatched to fly over the IoT network area and perform random 3D movements while broadcasting wireless signals to each group. Most drones have embedded WiFi transceivers, which can be used for signal broadcasting. For a drone without one, a lightweight battery-powered WiFi router attached to the drone can perform the broadcasting task. So my scheme is practical in terms of hardware requirements. A group of devices can receive the same broadcast signals and each extract a Channel State Information (CSI) stream. A device's CSI stream characterizes the dynamic variations of the unique wireless channel between the device and drone, which is mainly induced by fast drone movement. Although the CSI streams of different devices in a group depend on their respective channels with the drone, they are all correlated with the drone's trajectory and thus indirectly correlated with each other. DroneKey aims to quickly establish/refresh the group key of each individual group by mining this hidden CSI correlation.

The design of DroneKey faces two critical challenges. First, the relation among different CSI streams is highly complex and affected by many factors such as device locations, hardware features, channel shadowing and fading, and multipath signal propagation. So it is challenging to extract the hidden CSI correlation for establishing a group key in a distributed fashion. Second, since the drone's 3D trajectory is the dominating randomness factor for the group key, a powerful adversary may video-tape the drone movement to reconstruct the drone trajectory and then the group key. This trajectory-reconstruction threat is unique to DroneKey.

DroneKey adopts a deep-learning approach with an obfuscation function to address the above challenges. Within a group, one device is designated as the *group head* which can be chosen based on any sophisticated cluster-head selection algorithm in multi-hop wireless networks (Kang and Nguyen 2012; Thein and Thein 2010). All the non-head devices in a group are called *peer* devices. Although some existing correlation measurement algorithms can be used to measure the correlation between the head's and a peer's CSI streams, they cannot extract a common secret key because their output is just a single number between -1 and 1 indicating the correlation (Zar 2005; Abdi 2007). Recent studies show that the Deep Neural Network (DNN) can capture the correlation between two signals in a more sophisticated manner. For example, Wu *et al.* explore DNN to capture the correlation between the gait observations of two wearables on the same body but at different locations to infer one gait observation from the other (Wu et al. 2020). So I are motivated to adopt DNN in the DroneKey design.

DroneKey involves a one-time DNN training process in each group during the network-initialization phase. In particular, after IoT devices are deployed, one or a few drones are dispatched to traverse the network along random 3D trajectories while broadcasting wireless signals that can be received by all devices in each group. Each group head trains a unique DNN for each of its peer devices based on a confidential *obfuscation function*, its own CSI stream, and the CSI stream submitted by the peer device. The group head then sends the trained DNNs to the corresponding peer devices over a secure channel (Section 4.3.2).[2] Since the DNN of each peer device is closely tied to its relative location to the group head, different peer devices have diverse DNNs. But these DNNs are trained in a special way that each device in the same group can obtain the same output as the group key by feeding its CSI stream into its DNN. Since the obfuscation function is confidential, DroneKey guarantees that a passive eavesdropper cannot infer the group key from reconstructed CSI streams through the advanced trajectory-reconstruction attack. The training and distribution of DNNs are conducted only once during network initialization and not needed in each subsequent GKG instance. Whenever a new group key is needed, a drone is dispatched to perform 3D random movement while broadcasting wireless signals to IoT devices. Each device in a group just autonomously feeds the fresh CSI streams extracted from new drone signals into their respective DNNs to obtain a new group key without any interaction with each other.

Although DroneKey involves deep learning, the computational load for each IoT device is still lightweight. In particular, the one-time DNN training process during network initialization can be offloaded to a remote server if needed. In each subsequent GKG instance, each group head only needs to perform one matrix multiplication and a simple quantification operation to obtain the group key, and each peer device needs to conduct one DNN forward computation and a similar quantification operation to obtain the group key. Therefore, DroneKey is a feasible solution for large-scale IoT networks, which may contain many resource-constrained devices.

I prototype DroneKey and thoroughly evaluate its performance in both indoor and outdoor settings. The experimental results show that DroneKey can achieve key-

[2]Note that I only assume the availability of this secure channel during the network-initialization phase, which cannot be used to distribute new group keys during network operations (Section 4.3.2).

generation rates above 75 bit/sec in all the evaluated scenarios, and all the generated keys can pass the NIST randomness test. Table 8 shows the brief comparisons of DroneKey with the prior work. DroneKey outperforms the state-of-the-art PHY-based scheme (Liu et al. 2013) by more than 100% in terms of the key-generation rate for networks of the same size and can achieve a compatible key-generation rate for a large-scale IoT network 10 times larger than the networks considered in previous studies. In addition, I estimate that DroneKey can update all the 256-bit group keys in a large IoT network of 20,000 devices over a 1 km$\times$1 km area within 42 minutes with just one drone and 10 minutes with five drones. I also theoretically show that DroneKey is robust to the RF eavesdropping attack and also the drone-trajectory-reconstruction attack.

4.2 Background and Feasibility Study

4.2.1 Background of CSI

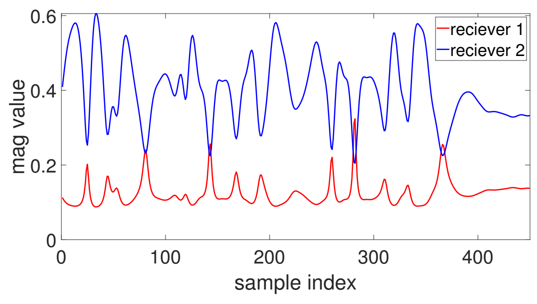
The PHY wireless channel characteristic at a specific frequency can be represented by the Channel Frequency Response (CFR). Given a transmitted signal whose frequency-domain representation is $X(f,t)$, the received signal can be represented as $Y(f,t) = H(f,t) \times X(f,t) + N(f,t)$, where $H(f,t)$ is the CFR at frequency f measured at time t, and $N(f,t)$ is the noise. $H(f,t)$ is a complex value and can be represented as $H(f,t) = a(f,t)e^{2\pi\phi(f,t)j}$, where $a(f,t)$ and $\phi(f,t)$ denote the magnitude attenuation and phase shift values, respectively. For brevity, I shall abbreviate magnitude attenuation and phase shift to mag and phase, respectively.

CSI measures the CFRs of a wireless channel at the carrier frequency or multiple subcarrier frequencies. Researchers have proposed many data-aided CSI-estimation schemes that work with all prevalent wireless techniques (Coleri et al. 2002; Morelli and Mengali 2001). In those schemes, a predefined pilot signal which is known to the receiver is transmitted, and the receiver estimates the CSI from its received signal. By adopting the existing generic CSI-estimation schemes, DroneKey can work with all prevalent wireless techniques for IoT systems.

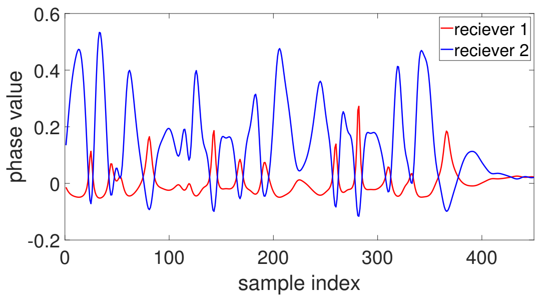
4.2.2 Feasibility Study

DroneKey depends on the premise that the CSI streams extracted from the same broadcast signals but by different receivers are correlated. I use a preliminary experiment to verify this feasibility. This experiment uses an N210 USRP, which is attached to a DJI Matrice 100 drone, as the transmitter and two other B210 USRPs placed three meters apart as two receivers. All the experiments in this chapter use this DJI drone, so I omit the drone model hereafter. I fly the drone back and forth between two receivers while the attached USRP keeps broadcasting WiFi packets. The two receivers keep extracting CSI from each received packet. The two resulting CSI streams are shown in Fig. 22. For a clear illustration, I process the raw streams with methods demonstrated in Section 4.4 and only show the processed streams of the 6th subcarrier here. There are obvious correlations between the two mag streams and also the two phase streams. So it is quite feasible to extract a common key from the two CSI streams.

In this chapter, DroneKey assumes wireless techniques based on orthogonal frequency-division multiplexing (OFDM) which is implemented in all WiFi stan-



(a) Mag streams.



(b) Phase stream.

Figure 22. The correlated CSI streams.

dards post 802.11b. DroneKey can be easily extended to all other prevalent wireless techniques because they all support the extraction of CSI. The CSI of OFDM-based wireless networks contains information about multiple subcarriers at different frequencies. In this scenario, the drone movement is the dominating reason for the CSI changes, and the CSI streams of different subcarriers at the same device are highly correlated. So DroneKey only uses one subcarrier for group-key generation to ensure sufficient key randomness. Hereafter, the term "CSI" refers to the CFR of the selected subcarrier, and each CSI sample is represented with a mag value and a phase value.

4.3 System Overview and Adversary Model

4.3.1 System Model

I consider a large-scale IoT network containing thousands of static IoT devices deployed over a large area for a mission-critical operation. The network is divided into many distributed groups with each containing several tens to hundreds of nearby devices. Once the network is deployed, it can be fully unattended during the long network lifetime.

A drone denoted by $\mathcal{D}$ is periodically dispatched to traverse the entire network along planned routes to refresh the group key of each device group in the network. Subsequent illustrations focus on one group of devices as shown in Fig. 23. DroneKey can support an arbitrary number of devices in each group. Without loss of generality, I assume that the IoT network adopts OFDM-based WiFi for group communications on the 2.4 GHz band as in the experimental evaluations, but DroneKey can be easily extended to any other wireless technique (e.g., Bluetooth, Zigbee, and Lora) because CSI measurement is universally supported.

One device in the group is selected as the group head with any sophisticated cluster-head selection algorithm in multi-hop wireless networks, and the rest devices in the group are called peer devices. The group head is denoted by $\mathcal{H}$, and the ith peer device is denoted by $\mathcal{P}_i$. $\mathcal{H}$ needs to be within the communication range of all the peer devices, but the peer devices do not need. In addition, I assume that the drone has a large communication range that covers the entire group.

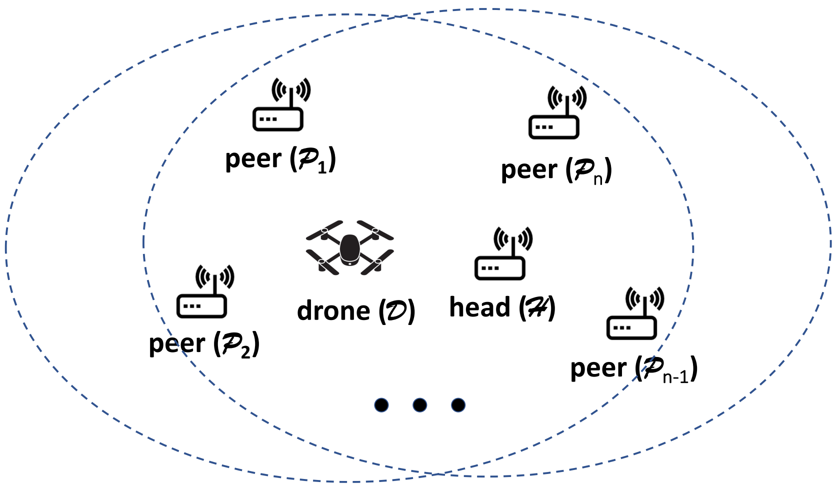


Figure 23. The system model of DroneKey.

4.3.2 DroneKey Workflow

DroneKey consists of a one-time initialization stage and subsequent group-key generation stages, as shown in Fig. 24. The black solid arrows indicate the timelines of the corresponding devices, and the dashed arrows represent the data exchanges between devices. The blue color indicates that a data transmission is secured with a pairwise secret key, and the red color means that a data transmission is in plain text. The workflows of all the peer devices are the same. So I use the first peer device $\mathcal{P}_1$ as the example to demonstrate DroneKey hereafter.

In the initialization stage, $\mathcal{D}$ flies randomly within a predefined area while continuously broadcasting WiFi packets. $\mathcal{P}_1$ and $\mathcal{H}$ each extracts a CSI stream from the broadcast signal and processes the stream with methods detailed in Section 4.4.2. I denote these two processed CSI streams of $\mathcal{P}_1$ and $\mathcal{H}$ by C_1^R and C_H^R, respectively. After obtaining C_H^R, $\mathcal{H}$ acquires C_1^R from $\mathcal{P}_1$, generates a training dataset from C_1^R and

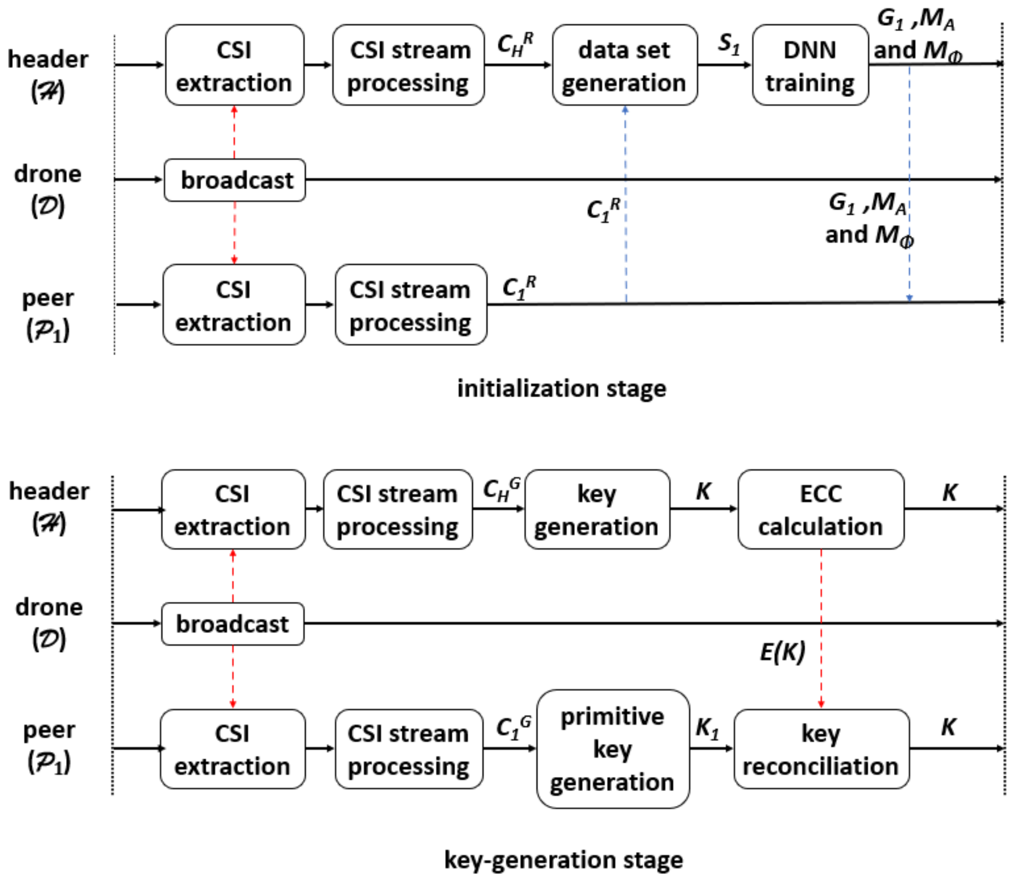


Figure 24. The workflow of DroneKey.

C_H^R, and finally trains a DNN. The training process involves an obfuscation function that can enhance DroneKey's security. This DNN is for the group-key generation at $\mathcal{P}_1$, and I denote it by G_1. More details regarding the dataset generation and DNN training are given in Section 4.5. Moreover, $\mathcal{H}$ determines the numbers of quantification bins from C_H^R and C_1^R, which is critical for the group-key generation stage; and the details are demonstrated in Section 4.6.2. Finally, $\mathcal{H}$ sends G_1 and the quantification-bin numbers to $\mathcal{P}_1$. The transmissions of C_1^R, G_1, and the quantification-bin number are secured with the pairwise key that can be established with any cryptographic or PHY-based method (Du et al. 2005; Liu et al. 2013; Diffie and Hellman 1976).

There are two remarks to make for network initialization. First, the training and distribution of DNNs is a one-time process. Second, I only assume the security of pairwise keys in the very short initialization phase to obviate the need for secure pairwise-key update schemes. This means that such pairwise keys are unavailable in subsequent network operations for securely delivering a new group key randomly selected by the group head to its peer devices.

In each subsequent key-generation stage, $\mathcal{P}_1$ acquires the group key K by generating its own primitive key and adjusting it according to the Error Correction Code (ECC) broadcast by $\mathcal{H}$. Particularly, $\mathcal{D}$ broadcasts WiFi packets while moving randomly, and $\mathcal{P}_1$ and $\mathcal{H}$ obtain their processed CSI streams from the broadcast signals. I denote these two CSI streams of $\mathcal{P}_1$ and $\mathcal{H}$ by C_1^G and C_H^G, respectively. Then $\mathcal{H}$ generates K from C_H^G and broadcasts the ECC of K. Next, $\mathcal{P}_1$ uses C_1^G as the inputs to G_1 and generates its primitive group key K_1 from G_1's output. Due to the impacts of the ambient noise, hardware flaws, and DNN estimation errors, K_1 may not be identical to K. So $\mathcal{P}_1$ adjusts K_1 according to the ECC broadcast by $\mathcal{H}$ in the final reconciliation step. In the key-generation stage, $\mathcal{H}$ and all the peer devices must extract their CSI streams simultaneously so that their CSI streams are correlated and can be used to generate a common secret key. Since I use the indexes of WiFi packets to synchronize different devices' CSI streams, $\mathcal{H}$ and the peer devices do not need to have synchronous clocks, which is also an advantage of DroneKey over prior work. DroneKey can renew the group key as needed by repeating the group-key generation process.

4.3.3 Adversary Model

Like all prior PHY-based GKG schemes (Liu et al. 2014; Thai, Lee, and Quek 2015; Jagadeesh, Joshi, and Rao 2021), I focus on establishing/updating group keys among randomly deployed wireless devices to secure wireless broadcast/multicast messages. Without loss of generality, I assume that the first peer device $\mathcal{P}_1$ is the attacker's target device, and the attacker—denoted by $\mathcal{A}$—aims to obtain the group key K. I assume a very powerful $\mathcal{A}$ with the following capabilities. First, $\mathcal{A}$ can deploy wireless monitors close to $\mathcal{H}$ and $\mathcal{P}_1$, with which $\mathcal{A}$ can overhear all the wireless signals and may be able to obtain similar copies of $\mathcal{H}$'s and $\mathcal{P}_1$'s CSI streams. Second, $\mathcal{A}$ is aware of $\mathcal{H}$'s and $\mathcal{P}_1$'s locations and can obtain the trajectories of $\mathcal{D}$ with a spy camera.

As demonstrated later, the one-time initialization stage is as short as several minutes, so it is very hard for the attacker to compromise DroneKey in this short time window. In addition, the deployment of peer devices usually involves human effort, and the initialization of a peer device is executed right after it is deployed. It is reasonable to assume human aid available in the initialization stage. Therefore, human-involved authentication schemes such as the code-based Bluetooth-like pairing can be adopted to defeat attacks against the initialization stage. Note that the security of the short network-initialization phase has been assumed in the extensive literature such as random key predistribution schemes for sensor networks (Eschenauer and Gligor 2002). I also assume that $\mathcal{A}$ cannot compromise $\mathcal{H}$ or $\mathcal{P}_1$, which is the same with all the existing GKG studies. Otherwise, $\mathcal{A}$ can directly obtain the group key. However, I have minimal assumptions about the security of $\mathcal{D}$. In particular, $\mathcal{A}$ can compromise $\mathcal{D}$ and control $\mathcal{D}$'s trajectory and the broadcast signal. $\mathcal{A}$ can also impersonate $\mathcal{D}$ with a malicious drone. Under these assumptions, I consider three specific attacks as follows.

- *Malicious-drone attack.* $\mathcal{A}$ compromises $\mathcal{D}$ and fully controls $\mathcal{D}$'s trajectory and the broadcast signal. $\mathcal{A}$ can also use a malicious drone to mimic $\mathcal{D}$. By manipulating the drone's trajectory and broadcast signal, $\mathcal{A}$ hopes to infer some information about the group key K.
- *Eavesdropping attack.* $\mathcal{A}$ eavesdrops on the WiFi channel used by DroneKey and tries to infer the group key K from the overheard signals.
- *Reproduction attack (or drone-trajectory-reconstruction attack).* $\mathcal{A}$ sets up an environment similar to DroneKey's and repeat $\mathcal{P}_1$'s initialization and key-generation processes to reproduce K. Specifically, $\mathcal{A}$ flies a drone in the arranged environment in the same trajectory as $\mathcal{D}$'s and measures CSI streams at the locations corresponding to those of $\mathcal{H}$ and $\mathcal{P}_1$. With the obtained CSI streams, $\mathcal{A}$ generates dataset, trains DNN, and generates a secret key as DroneKey does, hoping that the produced key is identical to K.

DroneKey cannot work when the jamming attack is launched. However, $\mathcal{A}$ cannot obtain K via the jamming attack, and DroneKey still works once the jamming signals are not present. Therefore, I do not consider the jamming attack in this chapter.

4.4 CSI Extraction and Processing

This section demonstrates the details of CSI extraction and processing. $\mathcal{H}$ and $\mathcal{P}_1$ use the same method to extract and process their CSI streams in both stages of DroneKey. Without loss of generality, I choose the 6th WiFi subcarrier for key generation, and the CSI sample only contains the mag and phase values of this subcarrier.

4.4.1 CSI Extraction

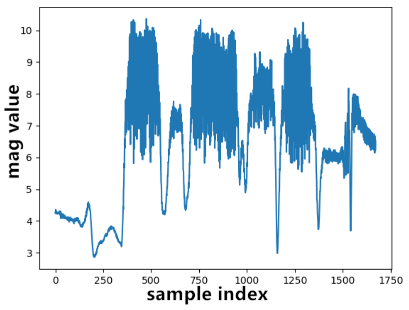
Among many existing CSI estimation schemes, DroneKey adopts the least-square equalizer for its low computational complexity (Coleri et al. 2002; Morelli and Mengali 2001). Specifically, $\mathcal{D}$ broadcasts successive WiFi packets containing only packet indexes. The preamble of each packet contains two copies of a predefined training sample, denoted as X_T. Given a received packet, the corresponding CSI sample is denoted by c and estimated as

$$c = \frac{Y_1 + Y_2}{2X_T}, \tag{4.1}$$

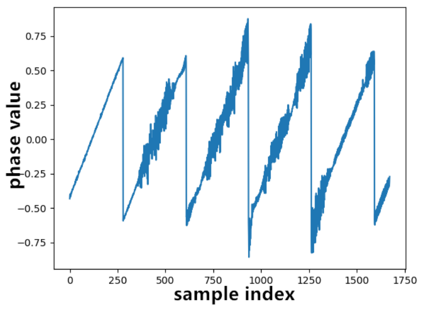
where Y_1 and Y_2 are the two received training samples. I use the index of this packet as c's index, which can be used to synchronize different devices' CSI streams. The CSI sample rate λ, i.e., the number of CSI samples extracted within one second, equals the packet transmission rate. The network works on the 2.4 GHz band and can transmit around 140 continuous WiFi packets within one second. So λ is around 140 sample/sec.

4.4.2 CSI Stream Processing

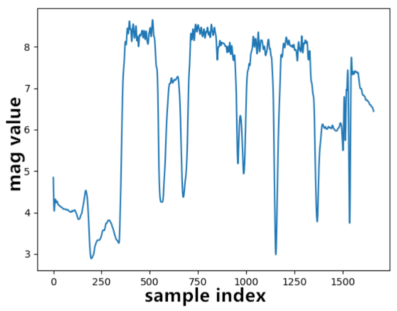
$\mathcal{H}$'s and $\mathcal{P}_1$'s raw CSI streams are heavily distorted due to the ambient noise and hardware flaws. Figs. 25a and 25b show $\mathcal{P}_1$'s raw mag and phase streams extracted in one experiment, in which $\mathcal{D}$ first keeps static and then moves randomly. The mag and phase streams are both distorted by high-frequency noise, and there is a periodically



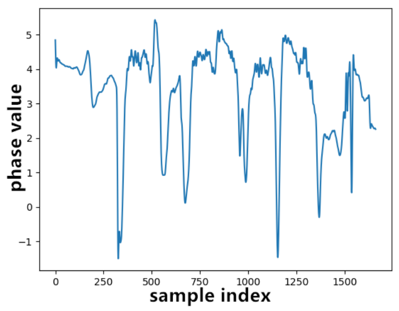
(a) Raw mag stream.



(b) Raw phase stream.



(c) Processed mag stream.



(d) Processed phase stream.

Figure 25. Raw and processed mag and phase streams.

changing offset in the phase stream. Moreover, some CSI samples corresponding to heavily interpreted WiFi packets are missing, so the mag and phase streams are not continuous in time. DroneKey processes the raw mag and phase streams with the following three steps.

Missing-sample estimation. I detect missing CSI samples based on sample indexes, estimate a missing sample with the uninterrupted samples around it, and insert the estimated samples to the original stream.

Phase calibration. After obtaining the continuous CSI stream, I calibrate the phase stream to remove the phase offset. The phase offset arises from the carrier-frequency offsets and asynchronous clocks of the transmitter (i.e., drone $\mathcal{D}$) and receiver. As shown in Fig. 25b, the phase offset periodically changes with time. Therefore, I use the phase samples extracted when the transmitter is static as a profile to calibrate the phase stream. Specifically, I first estimate the phase-offset changing cycle, denoted as T, from the profile and then calibrate the phase stream as

$$\phi_n = \phi_n - \phi_{(n \bmod T)}, \tag{4.2}$$

where ϕ_n is the nth phase sample in the stream.

For the convenience of phase calibration, I let $\mathcal{D}$ keep static for a period before moving randomly when DroneKey conducts CSI extraction. Experimental results show that T is always less than two seconds. So I set the static period of $\mathcal{D}$ as two seconds. The profile samples are extracted when $\mathcal{D}$ is static and contain little information useful for group-key generation. So I remove the profile samples from the CSI stream after the phase calibration.

Low-pass filter. Finally, I remove the high-frequency noise in the mag and phase streams with low-pass filters. The phase stream is more sensitive to noise and environmental changes than the mag stream. So the mag and phase filters use different cutoff frequencies. I denote the mag and phase cutoff frequencies by f_A and f_ϕ, respectively, which are obtained through experiments. In particular, I use an N210 USRP attached to the drone as the transmitter and two B210 USRPs placed together as two receivers. The distance between the two receivers is less than three centimeters. I fly the drone back and forth to the receivers randomly for 20 seconds while the drone keeps broadcasting WiFi packets. Each receiver extracts one CSI stream and processes it with missing-sample estimation and phase calibration. I denote these two processed CSI streams by $< A_1, \Phi_1 >$ and $< A_2, \Phi_2 >$, where A_i and Φ_i are the mag and phase streams of the ith receiver, respectively. I first determine f_A with A_1 and A_2. The CSI sampling rate is 140 sample/sec. According to the Nyquist–Shannon sampling theorem (Shannon 2001), the highest frequency of the signals contained in the mag stream is 70 Hz. So I test 70 values ranging from 1 Hz to 70 Hz with a step of 1 Hz for f_A. I use each tested value as f_A to filter A_1 and A_2 and calculate the correlation between the two filtered mag streams. The distance between the two receivers is less than half wavelength of the WiFi signal, and their CSI streams should be highly correlated without the noise's impact. So a good cutoff frequency should achieve a high correlation value. However, a low cutoff frequency may filter out too much information useful for key generation and thus reduce the key-generation rate. Therefore, I adopt 15 Hz as f_A because it is the maximum one among the values that achieve correlations above 0.9. Similarly, I determine f_Φ with Φ_1 and Φ_2 and adopt 20 Hz as f_Φ. The processed mag and phase streams are shown in Figs. 25c and 25d, respectively, which are continuous in time, smooth, free of noise, and can thus be used for key generation.

4.5 Key-generation DNN

In the peer device $\mathcal{P}_1$'s initialization stage, the group head $\mathcal{H}$ trains a DNN for $\mathcal{P}_1$ after obtaining $\mathcal{P}_1$'s CSI stream C_1^R. Specifically, $\mathcal{H}$ first generates a GKG dataset S_1 from C_1^R and $\mathcal{H}$'s own CSI stream C_H^R. It then trains $\mathcal{P}_1$'s DNN G_1 with S_1. Apart from generating the dataset and training the DNN, $\mathcal{H}$ also determines the number of quantification bins, which is critical for subsequent key-generation stages. However, the determination process is closely related to the details of the key-generation stage, so I defer its details to Section 4.6.2. This section first demonstrates the generation of S_1 and then details the training process of G_1.

4.5.1 GKG Dataset Generation

I first discuss G_1's function, which determines how the GKG dataset S_1 is generated. In the key-generation stage, the group key K is generated from $\mathcal{H}$'s CSI stream C_H^G. Device $\mathcal{P}_1$ uses its CSI stream C_1^G as G_1's input and generates a primitive group key K_1 from G_1's output. Therefore, G_1 should be able to estimate some information related to C_H^G from C_1^G, so K_1 can be similar to and can be adjusted to K in the final reconciliation step. To achieve this goal, a training sample for G_1 should contain two elements generated from $\mathcal{P}_1$'s and $\mathcal{H}$'s CSI streams, respectively. In the training process, the element related to $\mathcal{P}_1$ is used as G_1's input, and the element related to $\mathcal{H}$ is the target output. For consistence with machine learning concepts, I term the elements related to $\mathcal{P}_1$ and $\mathcal{H}$ as the feature and label, respectively.

Now I illustrate the generation of one training sample, and S_1 can be obtained by repeating this process. The training sample contains a feature and a label, which are two vectors denoted by V^f and V^l, respectively. I first randomly select a one-second segment, i.e., 140 continuous CSI samples, from C_1^R and represent it by $\tilde{C}_1^R$. I use the mag and phase values of the selected samples as the elements of V^f which can be represented as $[a_{(1,1)}, a_{(1,2)}, ..., a_{(1,140)}, \phi_{(1,1)}, \phi_{(1,2)}, ..., \phi_{(1,140)}]$, where $a_{(1,n)}$ and $\phi_{(1,n)}$ are the mag and phase values of the nth CSI sample in $\tilde{C}_1^R$, respectively. Then I select a one-second segment which is extracted simultaneously with $\tilde{C}_1^R$ from C_H^R. I use the sample indexes to synchronize C_H^R and C_1^R, and $\tilde{C}_1^R$ can be easily obtained. I denote the selected segment of C_H^R by $\tilde{C}_H^R$ and generate V^l from $\tilde{C}_H^R$ through a more complex process.

$\tilde{C}_H^R$ cannot be directly used as the label for two reasons. First, it is hard for G_1 to accurately estimate the fine-grained channel information contained in $\tilde{C}_H^R$. Second, directly using $\tilde{C}_H^R$ as the label is not secure. If I use $\tilde{C}_H^R$ as the label, G_1's output is the estimation of $\mathcal{H}$'s CSI stream. As demonstrated later in Section 4.6, $\mathcal{P}_1$ obtains its primitive group key K_1 by quantifying G_1's output. Accordingly, I must generate the group key by quantifying $\mathcal{H}$'s CSI stream. In this case, a powerful attacker who has obtained a similar copy of $\mathcal{H}$'s CSI stream can easily infer the group key.

To address the aforementioned concerns, I generate V^l from $\tilde{C}_H^R$ through down-sampling and obfuscation. Specifically, I first down-sample $\tilde{C}_H^R$ with a ratio of 1 to 10, and the down-sampled segment can be represented as a vector $V_d = [a_{(H,5)}, a_{(H,15)}, ..., a_{(H,135)}, \phi_{(H,5)}, \phi_{(H,15)}, ..., \phi_{(H,135)}]$, where $a_{H,n}$ and $\phi_{H,n}$ are the mag and phase values of the nth CSI sample in $\tilde{C}_H^R$, respectively. Then I apply an obfuscation function Λ to V_d to obtain the label vector $V^l = \Lambda(V_d)$. The obfuscation function can be represented as

$$\begin{aligned}\Lambda(V_d) &= V_d O \\ &= V_d \begin{bmatrix} O_A & 0 \\ 0 & O_\Phi \end{bmatrix} \\ &= \begin{bmatrix} A_H O_A & \Phi_H O_\Phi \end{bmatrix}\end{aligned} \tag{4.3}$$

Here, O is a 28×28 matrix and termed as an obfuscation matrix. O_A and O_Φ are both 14×14 matrices. A_H and Φ_H are the first and second half parts of V_d, respectively. The generated label V^l is a 1×28 vector. I term the whole process converting $\tilde{C}_H^R$ to V^l as label generation and denote it by $\mathcal{Z}(\tilde{C}_H^R, O)$. $\mathcal{H}$ uses the same obfuscation matrix to generate the group key-training datasets for all the peer devices.

I have two requirements for Λ. First, the information contained in each element of V_d must be inherited by $\Lambda(V_d)$. To satisfy this requirement, O_A and O_Φ must be full-rank matrices. Second, for the convenience of key quantification demonstrated later, the ranges of all the elements in $A_H O_A$ must be the same, and $\Phi_H O_\Phi$ should fulfill the same requirement. Since the movement of $\mathcal{D}$ is random, the elements in A_H are independent and identically distributed variables with the minimum value Min_A and the maximum value Max_A. The elements in A_Φ are also independent and identically distributed variables with the minimum value Min_Φ and the maximum value Max_Φ. The second requirement can thus be satisfied by requiring that the sum of O_A's elements and the sum of O_Φ's elements in the same column both equal one. Besides, I require all the elements in O to be non-negative. With the carefully designed O_A and O_Φ, the elements in $A_H O_A$ are within the range [Min_A, Max_A], and $\Phi_H O_\Phi$'s elements are within the range $[\text{Min}_\Phi, \text{Max}_\Phi]$.

4.5.2 GKG DNN Training

I adopt a Convolutional Neural Network (CNN) as G_1. Due to the random movement of $\mathcal{D}$, the relations between different CSI samples in the same CSI stream are not significant, and CNN is thus more suitable for DroneKey than the Recurrent Neural Network (RNN). G_1 contains four hidden layers and one output layer, and its architecture is shown in Fig. 26. The first hidden layer is a fully connected layer containing 280 neurons. The second hidden layer is a 1D convolutional layer without padding. The kernel size and step of the second hidden layer are 1×5 and 5, respectively. The third layer is another fully connected layer containing 112 neurons, and the last hidden layer is a no-padding 1D convolutional layer whose kernel size and step are 1×4 and 2, respectively. The four hidden layers all use the ReLU function as their activation functions. The fully connected output layer contains 28 neurons. The dimensions of each layer's input and output are also shown in Fig.26.

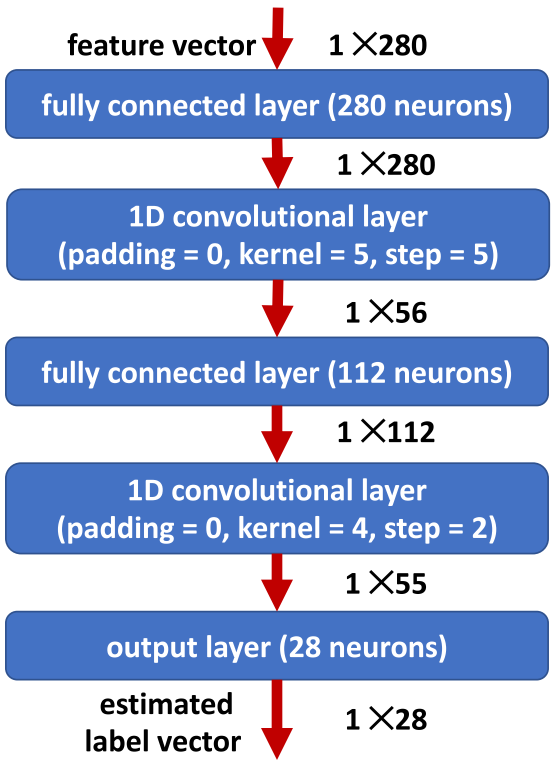

feature vector
1 ╳280
fully connected layer (280 neurons)
1 ╳280
1D convolutional layer
(padding = 0, kernel = 5, step = 5)
1 ╳56
fully connected layer (112 neurons)
1 ╳112
1D convolutional layer
(padding = 0, kernel = 4, step = 2)
1 ╳55
output layer (28 neurons)
estimated
label vector
1 ╳28

Figure 26. The architecture of a key-generation DNN.

In the training of G_1, I adopt the scaled Mean Square Error Loss (MSELoss) as the loss function. As demonstrated later in Section 4.6.2, $\mathcal{H}$ generates the group key K by quantifying the elements of a key-source vector, denoted as Θ. G_1's output, denoted as Θ_1, is the estimation of Θ. $\mathcal{P}_1$ generates its primitive group key by quantifying Θ_1's elements. Whether a GKG instance can succeed depends on Θ_1's worst element, i.e., the element with the largest estimation error. Therefore, I adopt MSELoss to balance the errors of different elements in G_1's output. Besides, the first and second half parts of Θ are generated from the mag and phase values, respectively, and they have different ranges. So does G_1's output. Therefore, I scale the loss function so that the first and second half parts of G_1's output evenly contribute to the loss value. In particular, given a training sample whose feature and label vectors are $V^f = [v_1^f, ...v_{280}^f]$ and

$V^l = [v_1^l, ..., v_{28}^l]$, respectively, G_1's output is represented as $G_i(V^f) = [\hat{v}_1^l, ..., \hat{v}_{28}^l]$, and the scaled MSELoss $l(V^l, G_i(V^f))$ is calculated as

$$\begin{aligned} l(V^l, G_1(V^f)) = & \frac{1}{14}\sum_{n=1}^{14}(\frac{v_n^l - \hat{v}_n^l}{\mathrm{Max}_A - \mathrm{Min}_A})^2 \\ & + \frac{1}{14}\sum_{m=15}^{28}(\frac{v_m^l - \hat{v}_m^l}{\mathrm{Max}_\Phi - \mathrm{Min}_\Phi})^2. \end{aligned} \tag{4.4}$$

$$\mathrm{Loss} = \frac{1}{14}\sum_{n=1}^{14}(\frac{\mathrm{v_n - v'_n}}{\mathrm{Max_A - Min_A}}) + \frac{1}{14}\sum_{m=15}^{28}(\frac{\mathrm{v_m - v'_m}}{\mathrm{Max_P - Min_P}}) \tag{4.5}$$

Here, Min_A and Max_A are the minimum and maximum mag values of $\mathcal{H}$'s CSI samples, respectively. Min_Φ and Max_Φ are the minimum and maximum phase values of $\mathcal{H}$'s CSI samples, respectively.

The training process of G_1 consists of multiple epochs. I adopt the Stochastic Gradient Descent (SGD) optimizer to update G_1's parameters in each epoch and adopt cross-validation to avoid overfitting. In particular, I equally divide the training dataset into 10 subsets and randomly select one of them as the validation set in each epoch. Once the validation set is chosen, I iteratively use every training sample in the rest nine subsets to calculate the gradients of G_1's parameters and update these parameters accordingly. To accelerate the training process, I use the average gradients calculated with 20 training samples to update G_1's parameters. At the end of each epoch, I calculate the averaged loss value with the training samples in the validation set and stop the training if the decrease of the loss value after this epoch is not significant, e.g., when the loss value decreases by less than five percent.

4.6 Key Generation and Reconciliation

In the key-generation stage, $\mathcal{H}$ and $\mathcal{P}_1$ first manage to obtain two related CSI streams from the same broadcast signals of $\mathcal{D}$ and then obtain the group key K through key generation and reconciliation. I denote $\mathcal{H}$'s and $\mathcal{P}_1$'s processed CSI streams by C_H^G and C_1^G, respectively. C_H^G and C_1^G are synchronized with sample indexes. Assume that C_H^G and C_1^G are extracted within one second such that they each contains 140 CSI samples. I demonstrate how to generate K from C_H^G and C_1^G in this section. To generate a group key from CSI streams longer than one second, $\mathcal{H}$ and $\mathcal{P}_1$ can obtain the key by segmenting their CSI streams, generating a key fragment from each CSI segment, and finally piecing all the key fragments together in the order of time.

$\mathcal{H}$ can directly generate K from C_H^G, and $\mathcal{P}_1$ acquires K with the aid of $\mathcal{H}$. In particular, $\mathcal{H}$ first generates a group key-source vector Θ_H from C_H^G. Then $\mathcal{H}$ acquires K by quantifying Θ_H's elements. Similarly, $\mathcal{P}_1$ also first generates its group key-source vector Θ_1 and then acquires its primitive group key K_1 by quantifying Θ_1's elements. Finally, $\mathcal{P}_1$ adjusts K_1 according to the ECC broadcast by $\mathcal{H}$ and acquires K after key reconciliation. In what follows, I first illustrate the generation of key-source vectors, then how to quantify them, and finally the key reconciliation process.

4.6.1 Key-source Vector Generation

$\mathcal{P}_1$'s group key-source vector Θ_1 must be similar to that of $\mathcal{H}$'s (i.e., Θ_H) so that the difference between the extracted K_1 and K is subtle and can be mitigated by ECC. To fulfill this requirement, I use $\mathcal{Z}(C_H^G, O)$ as Θ_H and use $G_1(C_1^G)$ as Θ_1. Here, $\mathcal{Z}$ and O are the label-generation process and obfuscation matrix explained in

Section 4.5.1, respectively. $G_1(C_1^G)$ is the output of G_1 with C_1^G as the input. As introduced in Section 4.5.2, $G_1(C_1^G)$ is the estimation of $\mathcal{Z}(C_H^G, O)$, so they are very close. In addition, Θ_1 and Θ_H are both 1×28 vectors.

4.6.2 Key-source Vector Quantification

$\mathcal{H}$ obtains K by quantifying the elements of Θ_H, and $\mathcal{P}_1$ obtains K_1 by quantifying Θ_1's elements. I use Θ_H's first element, denoted by $\theta_{(H,1)}$, as an example to illustrate the element quantification. I denote the the minimum and maximum values of $\theta_{(H,1)}$ by Min_{θ_1} and Max_{θ_1}, respectively, and divide the range $[\text{Min}_{\theta_1}, \text{Max}_{\theta_1}]$ to multiple bins. These bins are represented with a vector $B_1 = [b_{(1,0)}, b_{(1,1)}, b_{(1,2)}, ..., b_{(1,M_1)}]$, where $b_{(1,m-1)}$ and $b_{(1,m)}$ are the upper and lower bounds of the mth bin, respectively. $b_{(1,0)}$ equals Min_{θ_1}, and $b_{(1,M)}$ equals Max_{θ_1}. M_1 is the number of quantification bins for $\theta_{(H,1)}$. In practice, Min_{θ_1} and Max_{θ_1} are estimated, and $\theta_{(H,1)}$ may be outside the range $[\text{Min}_{\theta_1}, \text{Max}_{\theta_1}]$. Therefore, I quantify $\theta_{(H,1)}$ as

$$Q(\theta_{(H,1)}) = \begin{cases} 0, & \text{if } \theta_{(H,1)} < \text{Min}_{\theta_1}, \\ m, & \text{if } b_{(1,m)} < \theta_{(H,1)} \le b_{(1,m+1)}, \\ M-1, & \text{if } \theta_{(H,1)} > \text{Max}_{\theta_1}. \end{cases} \tag{4.6}$$

$\mathcal{H}$ can extract $\log_2(M_1)$ key bits from $Q(\theta_{(H,1)})$ with the gray coding technique (Ye, Reznik, and Shah 2006). Since Θ_1's first element $\theta_{(1,1)}$ is the estimation of $\theta_{(H,1)}$, I also quantify $\theta_{(1,1)}$ with B_1.

Now I demonstrate how to obtain B_1. I first look into the distribution of $\theta_{(H,1)}$. Since Θ_H and the label vectors in S_1 are all generated from $\mathcal{H}$'s CSI streams with the same process, I use the distribution of the label vector's first element v_1^l in S_1 as the estimated distribution of $\theta_{(H,1)}$. Let $F_{\theta_{(H,1)}}(x) = P(\theta_{(H,1)} \le x)$ denote $\theta_{(H,1)}$'s

Cumulative Distribution Function (CDF). I use the minimum and maximum values of v_1^l as Min_{θ_1} and Max_{θ_1}, respectively. So $b_{(1,0)}$ and $b_{(1,M)}$ are obtained. Given the number M_1 of the quantification bins, the rest elements of B_1 are determined as $F_{\theta_1}(b_{(1,m)}) = m/M_1$. As demonstrated in Section 4.5.1, the first 14 elements of Θ_H are related to the mag values of C_H^G with the same ranges. So I adopt the same number of quantification bins for the first 14 elements of Θ_H, which is denoted by M_A. For the same reason, the number of quantification bins for the last 14 elements of Θ_H is the same, which is denoted by M_Φ. Since $\mathcal{H}$ uses M_A and M_Φ to quantify its key-source vector, the peer devices in the network all use M_A and M_Φ for key-source vector quantification.

M_A and M_Φ significantly affect the group key-generation rate. For example, I can extract $\log_2(M_A)$ valid key bits from $\theta_{(H,1)}$, and a large M_A results in a high key-generation rate. However, with the increase of M_A, the key-mismatch rate between $\mathcal{H}$ and $\mathcal{P}_1$ also increases. Specifically, $\theta_{(1,1)}$ is the estimation of $\theta_{(H,1)}$ with a deviation $\delta_{(1,1)}$. With the increase of M_A, the sizes of quantification bins decrease, the possibility that $\theta_{(1,1)}$ and $\theta_{(H,1)}$ fall to different bins increases, and the number of mismatched key bits increases. If the mismatched key bits cannot be corrected by the ECC, the group-key generation instance fails.

I determine the optimal values of M_A and M_Φ for $\mathcal{P}_1$ based on the success rate of $\mathcal{P}_1$'s group-key generation and obtain the specific values with experiments. For convenience of presentation, I term the key bits extracted from the first 14 elements of Θ_H and Θ_1 as mag bits and term the key bits extracted from the rest elements of Θ_H and Θ_1 as phase bits. A group key-generation instance is successful for $\mathcal{P}_1$ only if the mismatched key bits between K_1 and K can be corrected by the ECC, requiring that mismatched mag and phase bits be corrected. I determine M_A based on the mag-part success rate, i.e., the possibility that the mismatched mag bits are correctable. Specifically, I test multiple values for M_A and estimate the mag-part success rate for each tested value with S_1. Among the tested values whose mag-part success rates are above a predefined threshold τ_s, I select the maximum one as M_A. The value of M_Φ can be obtained with a similar experiment using the same threshold value τ_s. The determined values of M_A and M_Φ guarantee that $\mathcal{P}_1$ can succeed with a probability above τ_s^2 in a group key-generation instance. The details of the experiments and the numerical results in different scenarios are given in Section 4.8.

The values of M_A and M_Φ determined with different peer devices' CSI streams may be different. After all the peer devices being initialized, DroneKey adopts the minimum one among the many obtained values of M_A as the final M_A value and determines the final M_Φ value similarly.

4.6.3 Reconciliation

I use the final reconciliation step to mitigate the deviations between $\mathcal{H}$'s and $\mathcal{P}$'s secret keys. Specifically, $\mathcal{H}$ calculates K's ECC, denoted by $E(K)$, and broadcast it. Then $\mathcal{P}_1$ adjusts K_1 according to the received ECC. DroneKey adopts the BCH(31,15) code (Bose and Ray-Chaudhuri 1960) in the evaluation, which allows three mismatched key bits for every 16 key bits.

4.7 Security Analysis

4.7.1 Security against Malicious-drone Attack

In DroneKey, $\mathcal{D}$ merely keeps broadcasting wireless signals to serve as a source for correlated channel randomness and has no other interaction with either the group head or peer devices. This means that the attacker $\mathcal{A}$ gets no information about the group key K from the compromised $\mathcal{D}$.

$\mathcal{A}$ may also manipulate the drone's trajectory by either compromising $\mathcal{D}$ or mimicking $\mathcal{D}$ with a malicious drone, hoping to manipulate K. As long as the predefined signal is broadcast, the correlation among devices' CSI streams exists, and a group key can be obtained by the head and all peer devices. Although the generated group key is manipulated, $\mathcal{A}$ cannot infer the group key without knowing the key-generation DNN and the obfuscation matrix.

In addition, $\mathcal{A}$ may manipulate the broadcast signal. In this case, the correlation among devices' CSI streams is broken. The number of unmatched bits between $\mathcal{H}$ and $\mathcal{P}_1$ increases and cannot be corrected with the ECC. The GKG instance fails, and all the devices stay with the old group key until a legitimate drone starts a new GKG instance. The old group key is unknown to $\mathcal{A}$, so DroneKey is not compromised either.

4.7.2 Security against Eavesdropping and Reproduction Attacks

Before analyzing DroneKey's security against eavesdropping and reproduction attacks, I first discuss how to launch these two attacks effectively. I assume that $\mathcal{A}$

is aware of DroneKey's workflow and the locations of $\mathcal{H}$ and $\mathcal{P}_1$, so $\mathcal{A}$ can obtain similar copies of $\mathcal{H}$'s and $\mathcal{P}_1$'s CSI streams through eavesdropping or reproduction attacks. Moreover, I assume that $\mathcal{A}$ has inferred M_A and M_Φ, which are the numbers of quantification bins demonstrated in Section 4.6.2, through the reproduction attack. Since the group key K is generated from $\mathcal{H}$'s CSI stream C_H^G, $\mathcal{A}$ tries to infer K from the obtained copy of C_H^G. In addition, I assume that $\mathcal{A}$ is lucky enough to obtain an identical copy of C_H^G. However, $\mathcal{A}$ has no information about the obfuscation matrix and can only try random matrices.

I aim to give a lower bound for the number of attempts after which $\mathcal{A}$ can certainly find a matrix, denoted by $\hat{O}$, that is close enough to O and can thus be used as the replacement of O to generate group keys. With the same format as O, $\hat{O}$ can be represented as $\hat{O} = \begin{bmatrix} \hat{O}_A & 0 \\ 0 & \hat{O}_\Phi \end{bmatrix}$. Since each element of the key-source vector is the dot product of V_d^G and the corresponding column of O, $\mathcal{A}$ can find $\hat{O}$ by searching for all the columns, i.e., the columns of $\hat{O}_A$ and $\hat{O}_\Phi$. Here, V_d^G is the vector obtained by down-sampling C_H^G and is represented as $[a^G_{(H,5)}, a^G_{(H,15)}, ..., a^G_{(H,135)}, \phi^G_{(H,5)}, \phi^G_{(H,15)}, ..., \phi^G_{(H,135)}]$, where $a^G_{H,n}$ and $\phi^G_{H,n}$ are the mag and phase values of the nth CSI sample in C_H^G, respectively.

I first investigate how many attempts are needed for the attacker to find $\hat{O}_A$'s first column, which can be denoted by $\hat{O}_{(A,1)} = [o_1, o_2, ..., o_{14}]'$. Also, I represent the mag values of V_d^G as $V_A = [a^G_{(H,5)}, a^G_{(H,15)}, ..., a^G_{(H,135)}]$. $\mathcal{A}$ uses $Q(V_A\hat{O}_{(A,1)})$ to estimate $Q(\theta_{(H,1)})$ and generates key bits from $Q(V_A\hat{O}_{(A,1)})$ with gray coding. Here, Q denote the quantification function, and $\theta_{(H,1)}$ is the first element of the group key-source vector Θ_H. I can know $\theta_{(H,1)} = V_A O_{(A,1)}$, where $O_{(A,1)}$ is the first column of O_A. I denote the key bits generated from $\theta_{(H,1)}$ and $V_d^G O_{(A,1)}$ as κ and $\hat{\kappa}$, respectively. The number of mismatched bits between κ and $\hat{\kappa}$ must be less than $\dfrac{3 * \text{len}(\kappa)}{16}$ so that $\hat{\kappa}$ can be corrected to κ according to its ECC. Here, $\text{len}(\kappa)$ is the number of key bits in κ. Experiments show that $\text{len}(\kappa)$ is always less than 32, so κ and $\hat{\kappa}$ differ by at most one bit. In gray coding, two integers' codes differ in one bit only if they are adjacent, so I can get $Q(\theta_{(H,1)}) - 1 \leq Q(V_A\hat{O}_{(A,1)}) \leq Q(\theta_{(H,1)}) + 1$. For simplicity, I assume that the quantification bins for $Q(\theta_{(H,1)})$ are of the same size. According to Eq. (4.6), $\hat{O}_{(A,1)}$ must satisfy the following requirement

$$V_A\hat{O}_{(A,1)} - V_A O_{(A,1)} \leq \frac{\text{Max}_A - \text{Min}_A}{M_A}, \tag{4.7}$$

where Min_A and Max_A denote the minimum and maximum mag values of $\mathcal{H}$'s CSI samples, respectively. I denote $\hat{O}_{(A,1)} - O_{(A,1)}$ as μ, then I can know

$$\begin{aligned} V_A\mu &= V_A \cdot \mu' \\ &= |V_A||\mu'| \cos\omega \\ &\leq \frac{\text{Max}_A - \text{Min}_A}{M_A}, \end{aligned} \tag{4.8}$$

where μ' is the transpose of μ and ω is the angle between vectors V_A and μ'. Since the drone $\mathcal{D}$'s movement is quite random, the V_A's direction and the angle ω are random. V_A's elements are non-negative, and experimental results show that M_A is always more than eight. So I reformulate the requirement as

$$\begin{aligned} |(\hat{O}_{(A,1)} - O_{(A,1)})'| &\leq \frac{\text{Max}_A - \text{Min}_A}{M_A * |V_A|} \\ &\leq \frac{\text{Max}_A - \text{Min}_A}{M_A * \min(|V_A|)} \\ &= \frac{1}{\sqrt{14}} * \frac{\text{Max}_A - \text{Min}_A}{M_A * \text{Min}_A} \\ &= \frac{1}{8\sqrt{14}} * \frac{\text{Max}_A - \text{Min}_A}{\text{Min}_A}. \end{aligned} \tag{4.9}$$

Now I talk about $\mathcal{A}$'s strategy searching for $\hat{O}_{(A,1)}$. Without any information about $O_{(A,1)}$, $\mathcal{A}$'s best strategy is brute-force searching with a grain of $\dfrac{1}{\gamma}$, where γ is a positive integer. Specifically, knowing that the sum of $\hat{O}_{(A,1)}$'s elements equals one, $\mathcal{A}$ tries all the vectors like $[\frac{n_1}{\gamma}, \frac{n_2}{\gamma}, ..., \frac{n_{14}}{\gamma}]'$, where n_i is called the grain-amount of ith element. All the grain-amounts are non-negative integers, and $\sum_{i=1}^{14} n_i = \gamma$. Given a grain $\frac{1}{\gamma}$, the number of vectors in the searching space can be calculated as

$$C(\gamma + 13, 13) = \frac{(\gamma + 13)!}{\gamma! * 13!}, \tag{4.10}$$

where γ must be sufficiently small so that at least one of the vectors in the searching space is close enough to $O_{(A,1)}$ and meets the requirement in Eq. (4.9). Among all the vectors in the searching space, I denote the one that is closest to $O_{(A,1)}$ as V_c. The largest possible value of $|(V_c - O_{(A,1)})'|$ is $\frac{\sqrt{14}}{2\gamma}$. Therefore, I formulate the requirement for γ as

$$\frac{\sqrt{14}}{2\gamma} \leq \frac{1}{8\sqrt{14}} * \frac{\text{Max}_A - \text{Min}_A}{\text{Min}_A}, \tag{4.11}$$

i.e.,

$$\gamma \geq \frac{56 * \text{Min}_A}{\text{Max}_A - \text{Min}_A}. \tag{4.12}$$

In my experiments, the value of $\frac{\text{Min}_A}{\text{Max}_A - \text{Min}_A}$ is always greater than 0.3, so γ must be greater than 17. According to Eq. (4.10), the searching space contains more than 1×10^8 vectors. Therefore, $\mathcal{A}$ needs to try at least 14×10^8 vectors to obtain a replacement matrix for O_A and try even more vectors to obtain a replacement matrix for O, which is extremely hard. In practice, there is a nontrivial deviation between C_H^G and $\mathcal{A}$'s copy, which makes it even harder to obtain a replacement obfuscation matrix. Therefore, DroneKey is robust to the eavesdropping and reproduction attacks.

4.8 Evaluation

I evaluate DroneKey in this section. In what follows, I first introduce the prototype implementation and then measure DroneKey's key-generation rates in different scenarios. I also evaluate the randomness and entropy of the generated keys with the standard NIST runs test and the system overhead of DroneKey. Finally, I give a theoretical estimation of the time consumed to generate and update the group keys for a large-scale network based on experimental results.

4.8.1 Implementation

I implement the system in Fig. 23 with three USRPs and a DJI Matrice 100 drone. Specifically, I attach one N210 USRP to the drone and use them together as $\mathcal{D}$. I use one B210 USRP as the group head $\mathcal{H}$ and another B210 USRP as the peer device $\mathcal{P}_1$. The N210 USRP is connected to a laptop with an Ethernet cable, and the two B210 USRPs are connected to a desktop which has two Intel 4.2 GHz i5 processors where all the computations are executed. I implement the CSI-estimation tool on GNU Radio by modifying the open source code of the Wime project (Bloessl et al. 2018) and implement the group key-generation DNN with PyTorch. The prototype is for the purpose of evaluation. In practice, $\mathcal{D}$ can use its embedded WiFi transceiver or an attached lightweight battery-powered WiFi router, instead of the USRP, for signal broadcasting.

4.8.2 Performance Metrics

I use three performance metrics, including the key-generation rate, the group-success rate, and the randomness of the generated keys. I define the key-generation rate as the number of key bits generated from the CSI samples collected within one sec. This definition is also adopted in previous studies (Arazi and Qi 2005; Liu et al. 2014; Thai, Lee, and Quek 2015; Tang, Jiang, and Zou 2017). I define the group-success rate as the possibility that all the peer devices in the group can obtain a common key identical to the key generated by $\mathcal{H}$. Correspondingly, I term the possibility that a specific peer device can obtain the key generated by $\mathcal{H}$ as the peer device's individual-success rate. Without loss of generality, I assume that $\mathcal{P}_1$ is the peer device whose individual-success rate is the lowest. As recommend by NIST (Rukhin et al. 2001), I use the runs test for randomness checking and also measure the entropy of generated key bits.

I conduct experiments with different configurations to evaluate DroneKey's performance in various scenarios. These experiments have the basic procedure and only differ in specific settings. To avoid redundancy, I use a basic experiment to demonstrate the common experimental procedure and then present the results for specific additional settings.

4.8.2.1 Basic Experiment

I conduct the basic experiment in a regular one-story residential house. $\mathcal{P}_1$ is placed 3 m away from $\mathcal{H}$. Since $\mathcal{D}$'s movement is limited by the cable connecting USRP to the laptop, I fly the drone within a 2 m$\times$2 m$\times$2 m cubic area whose center is around 4 m away from both $\mathcal{P}_1$ and $\mathcal{H}$. Hereafter, I refer to the center of $\mathcal{D}$'s moving area as $\mathcal{D}$'s location.

I first extract $\mathcal{H}$'s and $\mathcal{P}_1$'s CSI streams for 5 min and obtain around 40,000 CSI samples for each of them. Then I generate 10,000 training samples for the training dataset S_1 and train the group key-generation DNN G_1.

I also determine the quantification bins' numbers, i.e., M_A and M_Φ, from S_1 through massive virtual key-generation instances. For each virtual instance, I randomly select two synchronized CSI segments, termed as a CSI-segment pair, with each containing 140 CSI samples from $\mathcal{H}$'s and $\mathcal{P}_1$'s CSI streams. By repeating this process, I obtain 2,000 CSI-segment pairs. Then the 20 integers between 1 and 20 are used as M_A in turn to generate group keys from the 2,000 CSI-segment pairs. I consider a virtual instance mag-part successful if the mag bits generated from $\mathcal{P}_1$'s segment can be corrected to those generated from $\mathcal{H}$'s segment with BCH(31,15). Finally, I choose M_A as the highest value whose mag-part success rate is above a threshold τ_s. Similarly, I use the same threshold to determine M_Φ, and $\mathcal{P}_1$'s individual-success rate is thus τ_s^2. Since group-success rate decreases exponentially with the decrease of the individual-success rate, I adopt 99.5% as τ_s so that DroneKey can achieve an individual-success rate above 99% for $\mathcal{P}_1$. M_A and M_Φ are thus set to 12 and 9, respectively, and the corresponding key-generation rate is around 95 bit/sec.

Finally, I conduct real key-generation instances to verify that DroneKey can indeed achieve the expected success rate and also check the randomness of the generated keys. I extract $\mathcal{H}$'s and $\mathcal{P}_1$'s CSI streams for 500 sec, which can be considered 500 continuous key-generation instances. For each instance, I extract and compare $\mathcal{H}$'s and $\mathcal{P}_1$'s keys. Only three of the 500 instances fail, and $\mathcal{P}_1$'s individual-success rate is 99.4%, which is above the expectation. Then I piece together the 497 keys from the successful instances in the order of time and obtain a binary sequence containing around 47,000 bits. This binary sequence passes the runs test with a p-value of 0.69, which is much larger than the threshold 0.1, and the entropy of each key bits was 0.99. Therefore, DroneKey can generate around 95 random key bits per sec for a peer device while guaranteeing the individual-success rate of the device is above 99%.

I also measure the DNN training time in the basic experiment. A desktop which has two Intel 4.1 GHz i5 processors can finish the training within 50 sec. Considering that each peer device requires a unique DNN, the whole DNN training task for a large group can cost several hours. This issue can be addressed by offloading the training task to a remote server. A Dell 7920 Tower server with a Quadro RTX 5000 GPU can finish the DNN training within 5 sec. For a group containing 100 devices, all the peer devices' DNNs can be trained within 8 min, which is acceptable for the one-time group initialization.

I repeat the basic experiment in the outdoor environment, which is a backyard. With the same requirement for individual-success rate, DroneKey achieves a key-generation rate of 99 bit/sec, slightly better than in the indoor environment. This is because the impact of multipath signals is less significant in the outdoor environment, and the correlation between $\mathcal{H}$'s and $\mathcal{P}_1$'s CSI streams is thus more significant. The generated keys also pass the runs test with p-value of 0.73, and the entropy of each key bits is 0.99.

4.8.2.2 Impact of Network Scale.

DroneKey's performance also relates to the group size and the group-coverage area.

Impact of Group Size. Since the key generations at different peer devices are independent, the group size does not impact the individual-success rate of a specific device. However, when the group size increases, the group-success rate may decrease significantly. Therefore, I use the time consumed to generate 1,000 key bits as the metric to evaluate the impact of the group size and denote it by T_t. Although a large-scale IoT network may contain thousands of devices, the number of devices that are within the communication range of a drone is much smaller. In this evaluation, I assume that the group size is at most 100, which is much larger than the maximum group size of 10 reported in previous studies. I also consider the outdoor environment which is common for large-scale IoT network. I adopt the individual-success rate obtained in the aforementioned experiment as the minimum individual-success rate of any peer devices and can thus estimate the time consumption as $T_t = 1000/(99 * 0.99^N)$ sec, where N is the group size. The corresponding values of T_t for N equaling 10, 50, and 100 are 11.17 sec, 20.01 sec, and 27.6 sec, respectively. With the group size increasing by 400% and 900%, the time consumption only increases by 79.05% and 147.09%, respectively. For a dense network containing 100 devices, DroneKey can still generate 1,000 key bits within 30 sec. Therefore, DroneKey is scalable with the group size.

Impact of Group-coverage Area. To cover as many devices as possible, $\mathcal{H}$ should be close to the center of the group, and so should $\mathcal{D}$. Therefore, I measure the group-coverage area with the distance between $\mathcal{P}_1$ and $\mathcal{D}$. I first evaluate the group-coverage's impact in the indoor environment. In particular, I fix the distance between $\mathcal{H}$ and $\mathcal{D}$ as 3 m and increase the distance between $\mathcal{P}_1$ and $\mathcal{D}$ from 3 m to 7 m with a step of 0.5 m. Due to the constraints in the environment, I are unable to evaluate longer distance settings. For each location of $\mathcal{P}_1$, I measure the corresponding key-generation rate and show the results in Fig. 27. Then I repeat the experiment

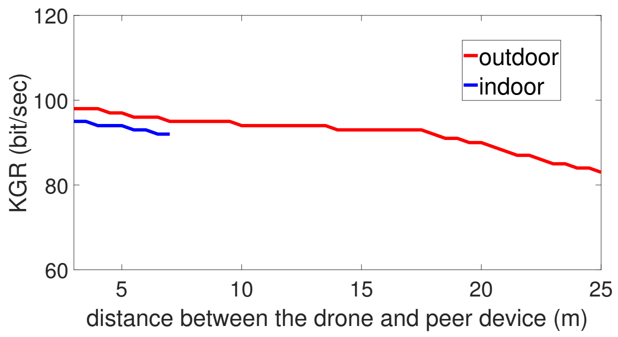


Figure 27. The impact of the group-coverage area.

in the outdoor environment and test 45 distance settings ranging from 3 m to 25 m with a step of 0.5 m. The results are also shown in Fig. 27. All the generated keys have passed the randomness check with p-values above 0.65. The results show that the impact of the group-coverage area is not significant. For a peer device 25 m away from $\mathcal{D}$, DroneKey can still achieve a key-generation rate above 80 bit/sec. Therefore, DroneKey is scalable in terms of the group-coverage area.

4.8.2.3 Impact of Environmental Factors

Since CSI is sensitive to environment changes, I evaluate the impact of environmental factors with experiments. Environment changes are usually caused by the relocation and movements of objects. I consider two common objects including persons and furniture for the indoor environment, as well as two common objects including persons and vehicles for the outdoor environment.

Impact of Indoor Environment Changes. I first measure the key-generation rate of DroneKey in a static indoor environment and use the result as the reference for comparison. In the experiments, $\mathcal{H}$ are placed 3 m away from $\mathcal{P}_1$, and $\mathcal{D}$ is 4 m away from both $\mathcal{H}$ and $\mathcal{P}_1$. One person stands still during the experiment, and there is a 2.5m (H)×2m(W)×1m(D) cabinet in the room. The person and the cabinet do not block the Line On Sight (LOS) channel between $\mathcal{H}$ and $\mathcal{D}$, denoted as channel H2D, or the LOS channel between $\mathcal{P}_1$ and $\mathcal{D}$, denoted as channel P2D. Then I let the person move to three new locations, denoted as locations A, B, and C. Location B blocks channel H2D; location C blocks the channel P2D, and location A does not block either channel. I measure DroneKey's key-generation rate with the person standing still in each location. Specifically, I conduct 200 GKG instances for each location and still uses the key-generation DNN obtained in the reference experiment to extract the group key. Due to the person's relocation, unmatched bits between $\mathcal{H}$ and $\mathcal{P}_1$ increase, and many GKG instances fail with the key-generation rate equal to 0 bit/sec. I calculate the average key-generation rate for each location and show the result in Table 9. After that, the person returns to the original location in the reference experiment, and I measure DroneKey's key-generation rates with the cabinet moved to locations A, B, and C. Finally, I let the person move in the area where DroneKey is deployed along two trajectories and measure DroneKey's key-generation rates accordingly. The person does not block the two LOS channels along trajectory 1 but frequently cuts off the two channels along trajectory 2. Table 9 shows the results measured in different settings. In the indoor environment, DroneKey's key-generation rate slightly decreases due to environment changes. Human movement has the most significant impact, but DroneKey can still achieve a key-generation rate of 91 bit/sec. Therefore, DroneKey is robust to indoor environment changes.

Impact of Outdoor Environment Changes. I first measure DroneKey's key-generation rate in a reference outdoor setting. Then I change some environmental factors and measure the corresponding key-generation rates. The locations of $\mathcal{H}$, $\mathcal{D}$, and $\mathcal{P}_1$ are fixed in all the experiments. $\mathcal{H}$ is placed 3 m away from $\mathcal{P}_1$, and $\mathcal{D}$ is 4 m away from both $\mathcal{H}$ and $\mathcal{P}_1$. There are one person standing still and a static vehicle in the reference setting, and neither blocks channel H2D or P2D. In the following experiments, I first let the person stand still in three different locations, denoted as A, B, and C, and measure the corresponding key-generation rates. Location B blocks channel H2D; location C blocks the channel P2D; and location A does not block either channel. Then I let the person return to the original location in the reference setting, move the vehicle to locations A, B, and C, and measure the corresponding key-generation rates. Finally, I measure DroneKey's key-generation rates with a person or a vehicle moving around. I evaluate trajectories 1 and 2 for the person and 3 and 4 for the vehicle. Trajectories 1 and 3 do not block the two LOS channels, while trajectories 2 and 4 do. Table 9 shows the key-generation rates measured in different settings. The movements and relocation of small objects such as humans have no obvious impact on DroneKey in the outdoor environment. In contrast, the movement and relocation of large objects, especially metal objects with flat surfaces such as vehicles, can slightly decrease DroneKey's key-generation rate. However, DroneKey still achieves a key-generation rate of 90 bit/sec in the worst case, which significantly outperforms the existing GKG schemes.

4.8.3 Overhead of DroneKey

This section evaluates DroneKey's overhead, including the memory cost, the energy consumption, and the computation and communication overhead.

Memory Cost. $\mathcal{D}$ stores the broadcast signal, which is less than 1 KB. $\mathcal{H}$ stores the obfuscation matrix O, M_A, and M_Φ. M_A and M_Φ are two float-type numbers, and O has 392 float-type none-zero elements. So the memory cost for $\mathcal{H}$ is less than 2 KB. A

Indoor factors	**Key-generation rate** (bit/sec)
Indoor reference	95
Human locations A/B/C	95/95/94
Cabinet locations A/B/C	95/92/93
Human trajectories 1/2	93/91
Outdoor factors	**Key-generation rate** (bit/sec)
Outdoor reference	99
Human locations A/B/C	99/99/97
Vehicle locations A/B/C	97/93/92
Human trajectories 1/2	99/96
Vehicle trajectories 3/4	95/90

Table 9. The impact of environment changes.

peer device stores a key-generation DNN, M_A, and M_Φ. The DNN contains around 86,000 parameters, each of which is a float-type number. So the memory cost for a peer device is less than 0.5 MB. The memory cost is trivial for any of the devices involved in DroneKey.

Energy Consumption. In DroneKey, $\mathcal{D}$ can return to a support station for battery charging or get battery changes after each key-generation instance. So the energy consumption is not a constraint for $\mathcal{D}$. I only evaluate the energy consumption of the head and peer devices. The energy consumption mainly results from the computation and communication, so I evaluate the computation and communication overhead of the head and peer devices instead.

Computation and Communication Overhead. Since the initialization is conducted once, I only consider the key-generation stage. In one key-generation instance, $\mathcal{H}$ broadcasts the ECC for one time, and no transmission is needed at the peer device. Therefore, the communication overhead for the whole group is one transmission per instance. Existing PHY-based GKG schemes all require each device to transmit at least one probe packet (Liu et al. 2014; Thai, Lee, and Quek 2015; Wei, Zhu, and Ni

2012; P. Xu et al. 2016). The communication overhead of these schemes is at least n transmissions per GKG instance, where n is the number of devices in the group. So DroneKey has much lower communication overhead than existing GKG schemes.

For each key-generation instance, $\mathcal{H}$ needs to perform a matrix multiplication, and a peer device needs to calculate the output of the key-generation DNN, which involves three matrix-multiplication operations. The computation overhead of DroneKey is slightly higher than that of existing PHY-based GKG schemes, which only involve quantification operations. However, the sizes of the involved matrices are not large, and the computation task can be easily handled by the processor of most IoT devices. Therefore, the computation overhead does not impact the deployment of DroneKey.

4.8.4 Whole-network Group-key Generation

A large-scale IoT network consists of many distribute device groups, each of which needs to refresh its group key frequently. I can estimate the total time it takes to generate or update the group keys for the entire network based on previous results for a single group. Assume that the network covers a 1 km$\times$1 km square region in which 20,000 IoT devices are deployed. The goal is to generate a group key of 256 bits for each device group, which is a recommended key size of the Advanced Encryption Standard (AES). Assume that a drone can cover a circle region with a diameter of 100 m (i.e., the typical WiFi transmission range at 2.4 GHz), each corresponding to the coverage area of a group. The whole network can then be divided into 196 device groups, each containing about 100 IoT devices. I first discuss the scenario in which only one drone is available. In this case, the drone flies to each device group one by one to help generate a group key. The distance between the centers of two adjacent device groups is around 71 m, and the speed of a COTS drone can be above 15 m/sec. So the drone movement between two adjacent groups can be finished within 5 sec. According to previous experimental results, the expected time of the drone's random movement for generating 256 key bits is 7.07 sec. Since the key-generation and reconciliation steps do not involve the drone, the drone only needs to stay with each group for no more than 8 sec. The total time consumption for generating the group keys of all the groups can thus be estimated as $196 \times (5 \text{ sec} + 8 \text{ sec}) = 2,548$ sec, which is around 42 minutes. The communication range of the drone can be increased by using a more powerful transmitter. In addition, multiple drones can be dispatched to assist different groups in parallel. For example, if five drones are used, DroneKey can update the group keys of the entire network within 10 minutes, which is short enough for most application scenarios.

4.9 Related Work

Recently, researchers have proposed many GKG schemes for IoT networks. In addition, a number of earlier GKG schemes proposed for wireless sensor networks can also be applied to the IoT network. Those schemes mainly fall into two categories based on cryptography and PHY information, respectively.

The schemes in the first category rely on cryptographic methods to secure the group-key distribution or agreement. Tubaishat *et al.* propose a scheme based on the multi-party Diffie–Hellman protocol (Tubaishat et al. 2004). In this scheme, a head device generates the group key after accumulating the rest devices' partial keys. The transmissions of both partial keys and the final group key to each device rely on pairwise key-based encryption. Public key cryptography is used to distribute the group key in (Seo et al. 2014) and (Bao et al. 2014). Zhu *et al.* propose a scheme that first establishes pairwise keys between the head device and each other device, and then the group key can be distributed (Zhu, Setia, and Jajodia 2006). In the context of large-scale IoT networks, a large amount of pairwise keys and public-private key pairs involved in (Tubaishat et al. 2004; Seo et al. 2014; Bao et al. 2014; Zhu, Setia, and Jajodia 2006) must be updated for each group-key update instance, making those schemes impractical in the considered context. In contrast, DroneKey also uses pairwise keys to secure the very short initialization stage and has no need for pairwise-key updates, so it is a more practical GKG solution for large-scale IoT networks.

Researchers have also proposed many cryptographic schemes not involving pairwise keys. Wen *et al.* propose a Bloom's matrix-based GKG scheme, in which a matrix is pre-shared among a group of devices and can be used for subsequent group-key generation (Wen et al. 2009). Teo *et al.* explore the Burmester-Desmedt group-key agreement method to generate a common key for devices forming a circular hierarchical group (Teo and Tan 2005). Those schemes require multiple rounds of communications involving all the devices and also incur heavy computation load when the group size is large. By comparison, DroneKey is efficient in both communication and computation. In terms of the communication overhead, DroneKey only requires the group head to broadcast the ECC in the group-key generation stage. As for the computation load, the group head performs a matrix multiplication and a simple quantification operation to obtain the group key, and each peer device conducts the DNN forward computation for one time and a similar quantification operation to obtain the group key. All the involved calculations are lightweight and suitable for resource-constrained IoT devices.

The PHY-based GKG schemes adopt the channel variations of one or multiple channels in the network as the randomness factor, from which the group key is generated. The PHY information is first explored to establish pairwise keys between two devices, and many secure and efficient schemes have been proposed (Hershey, Hassan, and Yarlagadda 1995; Sayeed and Perrig 2008; Liu et al. 2013; J. Zhang et al. 2016). There is also effort to achieve PHY-based GKG. The most intuitive solution is generating enough pairwise keys which can be used to distribute a group key from device to device. The scheme in (Li, Hu, and Hu 2019) follows this idea. The authors propose to first establish pairwise keys between a virtual center node and each of the rest nodes in a star network or between each node and its two neighbors in a chain network. Then the pairwise keys are used to securely transmit a random group key from device to device. Their scheme is not practical for large-scale IoT networks because a huge number of pairwise keys must be established in each group-key generation instance, which can consume significant time. Researchers have also proposed to spread the measurements of selected channels to the entire group of devices. Specifically, each device broadcasts a signal which can be obtained by fusing the measurements of multiple channels (Liu et al. 2013; Thai, Lee, and Quek 2015) or splitting the measurement of one channel (P. Xu et al. 2016). A legitimate device can infer the measurements of selected channels from the broadcast signals, while an attacker cannot. However, these methods require that each device transmit at least one probe packet in a group-key generation instance, and all the packets must be transmitted within the short channel coherent time without interfering with each other. So these schemes are not scalable to a large group in a dense IoT network either.

4.10 Conclusion

In this chapter, I propose DroneKey, a drone-aided PHY-based GKG scheme for large-scale mission-critical IoT networks. I use a randomly moving drone to introduce a randomness factor, which can be acquired by the entire group of devices and thus be used as the common randomness source for generating group keys. In particular, the CSI streams extracted from the same broadcast signals but by different devices are all correlated to the drone's movement and inherently correlated with each other. I adopt the deep-learning technique to capture the correlations among the CSI streams of different devices in a group, which guarantee the consistency of their individually generated keys. DroneKey involves a single broadcast message by the group head and no other message exchange within the group, so it is highly scalable with the group size. In case that a powerful attacker may obtain the drone's trajectory and infer the group key, I adopt an obfuscation function to enhance DroneKey's security and theoretically prove that DroneKey is robust against both eavesdropping and trajectory-reproduction attacks.

I build a prototype and evaluate DroneKey's performance in multiple scenarios. DroneKey can achieve a group-key generation rate over 85 bit/sec in most evaluated scenarios, significantly outperforming the state-of-the-art prior work. According to the experimental results, DroneKey is scalable in terms of the group size and network scale. Moreover, I estimate the time consumption of generating and updating group keys for an extremely large-scale IoT network covering a region of 1 km^2 and containing 20,000 devices. According to the estimation, the group-key update can be finished within 43 minutes, and the time consumption can be further reduced to 10 minutes by involving multiple drones that are equipped with signal amplifiers. In summary, DroneKey is a scalable, fast, and efficient GKG solution for large-scale IoT networks.

Chapter 5

CONCLUSION AND SUMMARY

This dissertation is an attempt to improve the security of IoT systems with AI-powered schemes and also sounds the alarm about the AI-posed threats to IoT security. On the one hand, Chapters 2 and 4 are two examples of AI-powered IoT security designs. In particular, data mining techniques are explored in Chapter 2 to improve the device identification accuracy in acoustic mobile authentication. Deep neural networks are used in Chapter 4 to extract a common key from indirectly related wireless signals, which enables a scalable group key generation scheme. On the other hand, DeepJam proposed in Chapter 3 shows that the advancement in AI also poses new threats to IoT systems. This DL-guided jamming attack is more stealthy and thus more threatening than traditional jamming attacks.

The work presented in this dissertation is far from perfect, and several challenges remain to be addressed in the future research. First, integrating AI in IoT systems may expose new attack surfaces. In particular, AI itself is vulnerable to many well-known attacks such as adversarial samples, the backdoor attack, and membership inference, so an AI-based system may also suffer from the same vulnerabilities. AI security in the context of IoT will be one focus of my future research. Second, the adoption of AI techniques such as deep learning places additional burdens on IoT systems. For example, the DNN model training consumes non-trivial power and time, and an IoT device needs extra memory to store the DNN model. A resource-constrained IoT system may not be able to afford these costs. In light of this, efficient and sustainable AI techniques for IoT will be another focus of my future research.

REFERENCES

Abdi, H. 2007. “The Kendall Rank Correlation Coefficient.” *Encyclopedia of Measurement and Statistics,* 508–510.

Arazi, O., and H. Qi. 2005. “Self-certified group key generation for ad hoc clusters in wireless sensor networks.” In *IEEE INFOCOM.* Miami, FL, March.

Aurelle, N., D. Guyomar, C. Richard, P. Gonnard, and L. Eyraud. 1996. “Nonlinear behavior of an ultrasonic transducer.” *Ultrasonics* 34 (2-5): 187–191.

Ba, Z., S. Piao, X. Fu, D. Koutsonikolas, A. Mohaisen, and K. Ren. 2018. “ABC: Enabling Smartphone Authentication with Built-in Camera.” In *NDSS.* San Diego, CA, February.

Bao, X., J. Liu, L. She, and S. Zhang. 2014. “A key management scheme based on grouping within cluster,” 3455–3460.

Beranek, L., and T. Mellow. 2012. *Acoustics: sound fields and transducers.* Academic Press.

Bloessl, B., M. Segata, C. Sommer, and F. Dressler. 2018. “Performance assessment of IEEE 802.11p with an open source SDR-based prototype.” *IEEE Transactions on Mobile Computing* 17 (5): 1162–1175.

Bojinov, H., Y. Michalevsky, G. Nakibly, and D. Boneh. 2014. “Mobile device identification via sensor fingerprinting.” *arXiv preprint arXiv:1408.1416.*

Bose, C., and D. Ray-Chaudhuri. 1960. “On a class of error correcting binary group codes.” *Information and control* 3 (1): 68–79.

Brik, V., S. Banerjee, M. Gruteser, and S. Oh. 2008. “Wireless device identification with radiometric signatures.” In *ACM MobiCom,* 116–127. San Francisco, CA, September.

Chen, D., N. Zhang, Z. Qin, X. Mao, Z. Qin, X. Shen, and X. Li. 2017. “S2M: A lightweight acoustic fingerprints-based wireless device authentication protocol.” *IEEE Internet of Things Journal* 4 (1): 88–100.

Chen, S., K. Zeng, and P. Mohapatra. 2010. “Jamming-resistant communication: channel surfing without negotiation.” In *2010 IEEE International Conference on Communications,* 1–6. IEEE.

Chen, W., and Q. Wu. 2010. "A proof of MITM vulnerability in public WLANs guarded by captive portal." *Asia-Pacific Advanced Network* 30:66–70.

Chen, Y., W. Xu, W. Trappe, and Y. Zhang. 2009. *A Brief Survey of Jamming and Defense Strategies,* 1–26. Springer.

Coleri, S., M. Ergen, A. Puri, and A. Bahai. 2002. "Channel estimation techniques based on pilot arrangement in OFDM systems." *IEEE Transactions on Broadcasting* 48, no. 3 (September): 223–229.

Czeski, A., M. Dietz, T. Kohno, D. Wallach, and D. Balfanz. 2012. "Strengthening User Authentication through Opportunistic Cryptographic Identity Assertions." In *ACM CCS.* Raleigh, NC, October.

Das, A., and N. Borisov. 2016. "Tracking mobile web users through motion sensors: Attacks and defenses." In *NDSS.* February, San Diego, CA.

Das, A., N. Borisov, and M. Caesar. 2014. "Do You Hear what I Hear?: Fingerprinting Smart Devices Through Embedded Acoustic Components." In *ACM CCS.* Scottsdale, AZ, November.

Derham, T., S. Doughty, K. Woodbridge, and C. Baker. 2007. "Design and evaluation of a low-cost multistatic netted radar system." *IET Radar, Sonar & Navigation* 1, no. 5 (October).

Dey, S., N. Roy, W. Xu, R. Choudhury, and S. Nelakuditi. 2014. "AccelPrint: Imperfections of Accelerometners Maks Smartphones Trackable." In *NDSS.* San Diego, CA, February.

Diffie, W., and M. Hellman. 1976. "New directions in cryptography." *Transactions on Information Theory* 22 (6): 644–654.

Du, W., J. Deng, Y. Han, P. Varshney, J. Katz, and A. Khalili. 2005. "A pairwise key predistribution scheme for wireless sensor networks." *Transactions on Information and System Security (TISSEC)* 8 (2): 228–258.

Eschenauer, L., and V. Gligor. 2002. "A key-management scheme for distributed sensor networks." In *ACM CCS,* 41–47. Washington, DC, November.

Garroppo, R., L. Gazzarrini, S. Giordano, and L. Tavanti. 2011. "Experimental assessment of the coexistence of Wi-Fi, ZigBee, and Bluetooth devices." In *International Symposium on a World of Wireless, Mobile and Multimedia Networks,* 1–9. IEEE.

Goodfellow, I., Y. Bengio, and A. Courville, eds. 2016. *Deep learning.* MIT press.

Goodfellow, I., J. Pouget, M. Mirza, B. Xu, D. Wardeand S. Ozair, A. Courville, and Y. Bengio. 2014. “Generative adversarial nets.” In *Advances in neural information processing systems,* 2672–2680.

Gunson, N., D. Marshall, H. Morton, and M. Jack. 2011. “User perceptions of security and usability of single-factor and two-factor authentication in automated telephone banking.” *Computers & Security* 30, no. 4 (June): 208–220.

Han, D., Y. Chen, T. Li, R. Zhang, Y. Zhang, and T. Hedgpeth. 2018. “Proximity-Proof: Secure and Usable Mobile Two-Factor Authentication.” In *ACM MobiCom.* New Delhi, India, October.

Han, D., A. Li, L. Zhang, Y. Zhang, J. Li, T. Li, R. Zhang, and Y. Zhang. 2021. “(In)secure Acoustic Mobile Authentication.” *IEEE Transactions on Mobile Computing.*

Hershey, J., A. Hassan, and R. Yarlagadda. 1995. “Unconventional cryptographic keying variable management.” *IEEE Transactions on Communications* 43 (1): 3–6.

Hochreiter, S., and J. Schmidhuber. 1997. “Long short-term memory.” *Neural computation* 9 (8): 1735–1780.

Hristo, B., M. Yan, N. Gabi, and B. Dan. 2014. “Mobile device identification via sensor fingerprinting.” *arXiv preprint arXiv:1408.1416.*

Jagadeesh, H., R. Joshi, and M. Rao. 2021. “Group secret-key generation using algebraic rings in wireless networks.” *IEEE Transactions on Vehicular Technology* 70 (2): 1538–1553.

Jana, S., S. Premnath, M. Clark, S. Kasera, N. Patwari, and S. Krishnamurthy. 2009. “On the effectiveness of secret key extraction from wireless signal strength in real environments.” In *ACM MobiCom.* Beijing, China, September.

Kang, S., and T. Nguyen. 2012. “Distance based thresholds for cluster head selection in wireless sensor networks.” *IEEE Communications Letters* 16 (9): 1396–1399.

Karapanos, N., C. Marforio, C. Soriente, and S. Capkun. 2015. “Sound-Proof: Usable Two-Factor Authentication Based on Ambient Sound.” In *USENIX Security.* Washington, DC, August.

Kulpa, K. 2006. “Continuous wave radars-monostatic, multistatic and network.” *Advances in Sensing with Security Applications,* 215–242.

Li, G., L. Hu, and A. Hu. 2019. "Lightweight group secret key generation leveraging non-reconciled received signal strength in mobile wireless networks." In *IEEE ICC,* 1–6. Shanghai, China, May.

Lin, Q., Z. An, and L. Yang. 2019. "Rebooting Ultrasonic Positioning Systems for Ultrasound-incapable Smart Devices." In *ACM MobiCom.* Los Cabos, Mexico, October.

Liu, H., Y. Wang, J. Yang, and Y. Chen. 2013. "Fast and practical secret key extraction by exploiting channel response." In *IEEE INFOCOM.* Turin, Italy, April.

Liu, H., J. Yang, Y. Wang, Y. Chen, and C. Koksal. 2014. "Group secret key generation via received signal strength: Protocols, achievable rates, and implementation." *IEEE Transactions on Mobile Computing* 13 (12): 2820–2835.

Machuzak, S., and S. Jayaweera. 2016. "Reinforcement learning based anti-jamming with wideband autonomous cognitive radios." In *2016 IEEE/CIC International Conference on Communications in China (ICCC),* 1–5. IEEE.

Mao, W., J. He, and L. Qiu. 2016. "CAT: High-Precision Acoustic Motion Tracking." In *ACM MobiCom.* New York, NY, USA, October.

Mathur, S., W. Trappe, N. Mandayam, C. Ye, and A. Reznik. 2008. "Radio-telepathy: Extracting a secret key from an unauthenticated wireless channel." In *ACM MobiCom.* San Francisco, California, September.

Mnih, V., K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, and G. Ostrovski. 2015. "Human-level control through deep reinforcement learning." *Nature* 518 (7540): 529–533.

Morelli, M., and U. Mengali. 2001. "A comparison of pilot-aided channel estimation methods for OFDM systems." *IEEE Transactions on Signal Processing* 49, no. 12 (December): 3065–3073.

Musaloiu-E, R., and A. Terzis. 2008. "Minimising the effect of WiFi interference in 802.15. 4 wireless sensor networks." *International Journal of Sensor Networks* 3 (1): 43–54.

Nandakumar, R., V. Iyer, D. Tan, and S. Gollakota. 2016. "FingerIO: Using Active Sonar for Fine-Grained Finger Tracking." In *ACM CHI.* San Jose, CA, May.

Pelechrinis, K., M. Iliofotou, and S. Krishnamurthy. 2010. "Denial of service attacks in wireless networks: The case of jammers." *IEEE Communications surveys & tutorials* 13 (2): 245–257.

Peng, C., G. Shen, Y. Zhang, Y. Li, and K. Tan. 2007. "BeepBeep: A High Accuracy Acoustic Ranging System using COTS Mobile Devices." In *ACM Sensys.* Sydney, Australia., November.

Polak, A., S. Dolatshahi, and D. Goeckel. 2011. "Identifying wireless users via transmitter imperfections." *IEEE Journal on selected areas in communications* 29 (7): 1469–1479.

PyTorch. 2004. Https://pytorch.org/.

Remley, K., C. Grosvenor, R. Johnk, D. Novotny, P. Hale, M. McKinley, A. Karygiannis, and E. Antonakakis. 2005. "Electromagnetic signatures of WLAN cards and network security." *International Symposium on Signal Processing and Information Technology, IEEE,* 484–488.

Roy, N., H. Hassanieh, and R. Roy Choudhury. 2017. "Backdoor: Making microphones hear inaudible sounds." In *ACM MobiSys.* Niagara Falls, NY, June.

Roy, N., S. Shen, H. Hassanieh, and R. Choudhury. 2018. "Inaudible voice commands: the long-range attack and defense." In *USENIX.* Baltimore, MD, August.

Rukhin, A., J. Soto, J. Nechvatal, M. Smid, and E. Barker. 2001. *A statistical test suite for random and pseudorandom number generators for cryptographic applications.* Https://csrc.nist.gov/publications/detail/sp/800-22/rev-1a/final. Booz-allen and hamilton inc mclean va.

Sayeed, A., and A. Perrig. 2008. "Secure wireless communications: Secret keys through multipath." In *ICASSP.* Las Vegas, NV, March.

Seo, S., J. Won, S. Sultana, and E. Bertino. 2014. "Effective key management in dynamic wireless sensor networks." *Transactions on Information Forensics and Security* 10 (2): 371–383.

Shannon, C. 2001. "A mathematical theory of communication." *ACM SIGMOBILE mobile computing and communications review* 5 (1): 3–55.

Shi, Y., K. Davaslioglu, and Y. Sagduyu. 2019. "Generative adversarial network for wireless signal spoofing." In *ACM Workshop on Wireless Security and Machine Learning,* 55–60.

Shi, Y., T. Erpek, Y. Sagduyu, and J. Li. 2018. "Spectrum data poisoning with adversarial deep learning." In *Military Communications Conference,* 407–412. IEEE.

Shi, Y., Y. Sagduyu, T. Erpek, K. Davaslioglu, Z. Lu, and J. Li. 2018. "Adversarial deep learning for cognitive radio security: Jamming attack and defense strategies." In *International Conference on Communications Workshops,* 1–6. IEEE.

Shirvanian, M., S. Jarecki, N. Saxena, and N. Nathan. 2014. "Two-Factor Authentication Resilient to Server Compromise Using Mix-Bandwidth Devices." In *NDSS.* San Diego, CA, February.

Slimeni, F., B. Scheers, Z. Chtourou, and V. Le. 2015. "Jamming mitigation in cognitive radio networks using a modified Q-learning algorithm." In *2015 International Conference on Military Communications and Information Systems (ICMCIS),* 1–7. IEEE.

Sutton, R., and A. Barto. 2018. *Reinforcement learning: An introduction.* MIT press.

Szabo, T. 1994. "Time domain wave equations for lossy media obeying a frequency power law." *The Journal of the Acoustical Society of America* 96 (1): 492–500.

Tang, T., T. Jiang, and Wei. Zou. 2017. "Group secret key generation in physical layer, protocols and achievable rates." In *IEEE ISCIT.* Cairns, Australia, September.

Teo, J., and C. Tan. 2005. "Energy-efficient and scalable group key agreement for large ad hoc networks." In *Proceedings of the 2nd ACM international workshop on performance evaluation of wireless ad hoc, sensor, and ubiquitous networks,* 114–121.

Thai, C., J. Lee, and T. Quek. 2015. "Secret group key generation in physical layer for mesh topology." In *IEEE GLOBECOM.* San Diego, CA, December.

Thein, M., and T. Thein. 2010. "An energy efficient cluster-head selection for wireless sensor networks." In *IEEE ISMS,* 287–291. Liverpool, UK, January.

Tope, M., and J. McEachen. 2001. "Unconditionally secure communications over fading channels." In *IEEE MILCOM.* McLean, VA, October.

Tubaishat, M., J. Yin, B. Panja, and S. Madria. 2004. "A secure hierarchical model for sensor network." *Sigmod Record* 33 (1): 7–13.

Wang, Q., K. Ren, M. Zhou, T. Lei, D. Koutsonikolas, and L. Su. 2016. “Messages Behind the Sound: Real-Time Hidden Acoustic Signal Capture with Smartphones.” In *ACM MobiCom.* New York City, NY, October.

Wang, W., and H. Shao. 2013. “Performance prediction of a synchronization link for distributed aerospace wireless systems.” *The Scientific World Journal* 2013 (July).

Watkins, C. 1989. *Learning from delayed rewards.* King’s College, Cambridge.

Wei, T., and X. Zhang. 2015. “mtrack: High-precision passive tracking using millimeter wave radios.” In *ACM MobiCom.* Paris, France, September.

Wei, Y., C. Zhu, and J. Ni. 2012. “Group secret key generation algorithm from wireless signal strength.” In *IEEE ICICSE,* 239–245. Zhengzhou, China, April.

Weir, C., G. Douglas, T. Richardson, and M. Jack. 2009. “Usable security: User preferences for authentication methods in eBanking and the effects of experience.” *Interacting with Computers* 22, no. 3 (October): 153–164.

Wen, M., Y. Zheng, W. Ye, K. Chen, and W. Qiu. 2009. “A key management protocol with robust continuity for sensor networks.” *Computer Standards & Interfaces* 31 (4): 642–647.

Wilhelm, M., I. Martinovic, J. Schmitt, and V. Lenders. 2011. “Short paper: reactive jamming in wireless networks: how realistic is the threat?” In *WiSec.* Hamburg, Germany, June.

Wu, Y., Q. Lin, H. Jia, M. Hassan, and W. Hu. 2020. “Auto-Key: Using autoencoder to speed up gait-based key generation in body area networks.” *ACM, Interactive, Mobile, Wearable and Ubiquitous Technologies* 4 (1): 1–23.

Xu, P., K. Cumanan, Z. Ding, X. Dai, and K. Leung. 2016. “Group secret key generation in wireless networks: algorithms and rate optimization.” *IEEE Transactions on Information Forensics and Security* 11 (8): 1831–1846.

Xu, W., W. Trappe, Y. Zhang, and T. Wood. 2005. “The feasibility of launching and detecting jamming attacks in wireless networks.” In *ACM MobiHoc.* Urbana-Champaign, IL, May.

Yan, Q., H. Zeng, T. Jiang, M. Li, W. Lou, and T. Hou. 2016. “Jamming resilient communication using MIMO interference cancellation.” *IEEE Transactions on Information Forensics and Security* 11 (7): 1486–1499.

Ye, C., A. Reznik, and Y. Shah. 2006. "Extracting secrecy from jointly Gaussian random variables." In *IEEE ISIT.* Seattle, WA, July.

Yu, Y., T. Wang, and S. Liew. 2019. "Deep-reinforcement learning multiple access for heterogeneous wireless networks." *Journal on Selected Areas in Communications* 37 (6): 1277–1290.

Zar, J. 2005. "Spearman rank correlation." *Encyclopedia of Biostatistics* 7.

Zhang, G., C.Yan, X. Ji, T. Zhang, T. Zhang, and W. Xu. 2017. "DolphinAttack: Inaudible voice commands." In *ACM CCS.* Dallas,TX, October.

Zhang, J., A. Marshall, R. Woods, and T. Duong. 2016. "Efficient key generation by exploiting randomness from channel responses of individual OFDM subcarriers." *IEEE Transactions on Communications* 64 (6): 2578–2588.

Zhang, X., and K. Shin. 2011. "Enabling coexistence of heterogeneous wireless systems: Case for ZigBee and WiFi." In *Proceedings of the Twelfth ACM International Symposium on Mobile Ad Hoc Networking and Computing,* 1–11.

Zhao, S., Z. Lu, Z. Luo, and Y. Liu. 2019. "Orthogonality-Sabotaging Attacks against OFDMA-based Wireless Networks." In *INFOCOM.* Paris, France: IEEE, April.

Zhou, X., X. Ji, C. Yan, J. Deng, and W. Xu. 2019. "NAuth: Secure Face-to-Face Device Authentication via Nonlinearity." In *INFOCOM.* Paris, France, April.

Zhou, Z., W. Diao, X. Liu, and K. Zhang. 2014. "Acoustic Fingerprinting Revisited: Generate Stable Device ID Stealthy with Inaudible Sound." In *ACM CCS.* Scottsdale, AZ, November.

Zhu, S., S. Setia, and S. Jajodia. 2006. "LEAP+: Efficient security mechanisms for large-scale distributed sensor networks." *Transactions on Sensor Networks* 2 (4): 500–528.