LEVERAGING MULTIMODAL SENSING FOR ENHANCING THE SECURITY AND PRIVACY OF MOBILE SYSTEMS

by

Habiba Farrukh

A Dissertation

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Doctor of Philosophy

Department of Computer Science

West Lafayette, Indiana

August 2023

To friends and family, both given and chosen.

ACKNOWLEDGMENTS

I started my graduate school in 2017 when I traveled outside my home country for the first time to pursue a Ph.D. in Computer Science at Purdue University. As I write this thesis, I am overwhelmed with a profound sense of gratitude and humility. I learned a lot of lessons during my Ph.D. that shaped me into who I am today, and I have several individuals and entities to thank for supporting me through this challenging but rewarding journey.

First and foremost, my deepest gratitude goes to my advisor, Prof. Z. Berkay Celik. His unwavering support during my Ph.D. was instrumental in my growth as a researcher. Berkay not only taught me the art of effective research and communication but also inspired me to become an empathetic and thoughtful academic. Working alongside him, I gained numerous opportunities to collaborate with fellow researchers, honing my mentoring and leadership skills. The values of ethics and professionalism that he instilled in me have transformed my perspective on research and life, helping me become a better scholar. I am truly honored to have been advised by him, and I am committed to passing on these invaluable lessons to future generations of academics.

I am also extremely thankful to Prof. Antonio Bianchi for his guidance, invaluable comments, and positive attitude toward my research work. He has taught me a great deal about mobile security and provided me with insightful feedback for my projects. I also certainly appreciated his knowledge of US immigration law when navigating through visa issues. I extend my heartfelt thanks to Prof. Dongyan Xu and Prof. Sonia Fahmy for serving on my thesis committee, sharing their invaluable feedback, and supporting me in my academic job search. I would also like to thank Prof. Chunyi Peng for serving on my examining committee. Additionally, I am indebted to Dr. Roopsha Samanta for being my initial advisor. When I joined Purdue CS, Dr. Samanta was extremely kind in helping me settle at Purdue and navigate through graduate school.

This thesis would not have been possible without the help and support of Dr. He Wang. He introduced me to the field of mobile sensing and taught me several fundamental concepts in computer vision and signal processing that helped me immensely in my research. His feedback on my initial projects greatly improved my critical thinking and helped me learn from my mistakes. I also learned to tackle research problems with perseverance and patience from him. I am forever grateful to him.

I am deeply grateful to Purdue University and the Department of Computer Science for supporting me during my Ph.D. I made several memories in the Lawson building over the last six years that I will cherish forever. I am especially thankful to the staff members at Purdue CS who continue to help grad students through all academic/non-academic problems and try their best to provide a welcoming environment to the students during stressful times. Some of my fondest memories from my Ph.D. journey are associated with the weekly graduate social hours, and I am forever grateful to the Purdue CS graduate office for organizing them. I am also thankful for the friendship and encouragement that I received from several of my friends at Purdue, including Adil Ahmad, Haleema Sadia, Meher Alam, Anna Mirza, Dana El Rushaidat, Pavani Guttula, Gowtham Kaki, Vikram Ravindra, Negin Karisani, Iqra Nazir, Mah Gull Syed, Neha Zaidi, Roshan Salim, Bilal Ahmed, Muddassar Hussain, Kamran Khalil, Ans Fida, Muhammad Haseeb Malik, Qudsia Rizvi, Nouraldin Jaber, Joe Eappen, Gregory Essertel, Yi Sun, Lu Yan, Pedro da Costa Abreu, Kevin and Amy Smith, and Samantha and Andrew Hansford.

At Purdue, I have been fortunate to be a part of a vibrant research community in the PurSec Lab. I express my gratitude to all the lab members for their support, encouragement, and camaraderie. I will fondly remember our weekly group meetings (with and without pizza :)) with all the presentations and discussions. The material in this dissertation is spawned from several research projects, none of which would be possible if not for the help I received from my exceptionally talented labmates and collaborators. These collaborators include Siyuan Cao, Reham Aburas, Ozgur Ozmen, Arjun Arunaslam, Doguhan Yeke, Abdullah Imran, Muhammad Ibrahim, Faik Kerem Ors, Chandrika Mukherjee, Ruoyu Song, Raymond Muller, Prashast Srivastava, Jianliang Wu, Khaled Serag, Kyungtae Kim, Hyungsub Kim, Arslan Khan, Siddharth Muralee, Ashwin Nambiar, Zeyu Lei, and Abdulellah Alsaheel. They not only taught me new perspectives in problem-solving but also created a friendly lab environment that made research an enjoyable and enriching experience. I particularly want to thank Siyuan since she was the first to help me navigate research, and her examples significantly improved my writing and development skills. I am grateful for her mentorship, friendship, and kindness during my initial years. I am also indebted to Reham for being a constant support throughout this challenging Ph.D. journey. We sat next to each other in the lab for five years and experienced the highs and lows of life together. I will always cherish her friendship, support, and encouragement. A special shoutout also goes to Arjun for being the kindest and caring labmate, collaborator, and friend. I will always remember Arjun's kindness and support during my academic job search and beyond.

Before joining Purdue, I was an undergraduate student at the Lahore University of Management Sciences (LUMS), where I had the privilege of being mentored by world-class professors, without whose encouragement perhaps I would not have dreamt of pursuing a doctorate. I am deeply indebted to Prof. Junaid Haroon Siddiqui, Prof. Ihsan Ayyub Qazi, and Prof. Fareed Zaffar. They introduced me to the concept of conducting high-quality research, and their contagious passion for teaching and mentoring significantly shaped my career. I hope, someday, I can change a student's life the way they changed mine.

I am incredibly lucky to have made lasting friendships during my undergraduate and high school. Over the years, it is to these friends that I have often turned to during both good and tough times. To name a few, I am forever grateful to Shalan Naqvi, Mahir Latif, Shabhaz Gardezi, Ubaid Ullah Hafeez, Izza Tariq, Murayyiam Parvez, Aynoor Saleem, Rahma Nawab, Asher Anjum, Tayyab Hussain, Aqib Nisar, Aiman Awan, and Zammad Idrees.

I am grateful to the Greater Lafayette community with whom I shared six years of my life. I am happy to call this beautiful and peaceful town my home, and I will always cherish my memories of it.

Lastly, my deepest gratitude goes to my parents and brother for their unconditional love and unwavering support. They have always put my needs before theirs, supporting me through the highs and lows of life. I am forever indebted to them and strive to make them proud and content. Above all, I owe an immeasurable debt of gratitude to my partner, whose unconditional love, tireless support, and constant encouragement made all the work described in this thesis possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Mobile systems, such as smartphones, wearables (e.g., smartwatches, AR/VR headsets), and IoT devices, have come a long way from being just a method of communication to sophisticated sensing devices that monitor and control several aspects of our lives. These devices have enabled several useful applications in a wide range of domains ranging from healthcare and finance to energy and agriculture industries. While such advancement has enabled applications in several aspects of human life, it has also made these devices an interesting target for adversaries.

In this dissertation, I specifically focus on how the various sensors on mobile devices can be exploited by adversaries to violate users' privacy and present methods to use sensors to improve the security of these devices. My thesis posits that multi-modal sensing can be leveraged to enhance the security and privacy of mobile systems.

In this, first, I describe my work demonstrating that human interaction with mobile devices and their accessories (e.g., stylus pencils) generates identifiable patterns in permissionless mobile sensors' data, revealing sensitive information about users. Specifically, I developed `S3` to show how embedded magnets in stylus pencils impact the mobile magnetometer sensor and can be exploited to infer a users incredibly private handwriting. Then, I designed `LocIn` to infer a users indoor semantic location from 3D spatial data collected by mixed reality devices through LiDAR and depth sensors. These works highlight new privacy issues due to advanced sensors on emerging commodity devices.

Second, I present my work that characterizes the threats against smartphone authentication and IoT device pairing and proposes usable and secure methods to protect against these threats. I developed two systems, `FaceRevelio` and `IoTCupid`, to enable reliable and secure user and device authentication, respectively, to protect users' private information (e.g., contacts, messages, credit card details) on commodity mobile and allow secure communication between IoT devices. These works enable usable authentication on diverse mobile and IoT devices and eliminate the dependency on sophisticated hardware for user-friendly authentication.

1. INTRODUCTION

Mobile systems, including smartphones, wearables, AR/VR headsets, and Internet of Things (IoT) devices, have evolved significantly over the years. They have transformed from being a simple source of communication into advanced sensing devices that play a crucial role in monitoring and managing various aspects of our daily lives. With the ever-increasing storage capacities of mobile devices, more and more sensitive information in the form of messages, photos, bank accounts and more find its place on these devices. The continuous improvements in cellular communication and wireless networking have pushed mobile devices from being a simple two-way communication channel to being GPS navigators, browsers, personal assistants, and even handheld gaming consoles. A lot of these additional functionalities can be attributed to the availability of several mobile sensors incorporated in these devices. For example, today's smartphones come embedded with 14 sensors including but not limited to accelerometer, gyroscope, magnetometer, microphone, and ambient light sensor. The data from these sensors has enabled a vast variety of applications in mobile health, context awareness, activity tracking, gaming, etc. Although these applications have increased the overall usability of these devices, they also leak information about aspects of users' activities that might be considered private.

The security and privacy implications of unauthorized access to users' personal information, habits, behaviors, and preferences through sensor data have long been a significant concern [1, 2]. Mobile operating systems like Android and iOS have implemented permission systems that require users to explicitly grant access to sensitive data, such as GPS locations, microphones, and camera recordings, to applications. However, the regulation of data from more generic sensors like the magnetometer, accelerometer, and gyroscope is less stringent, allowing applications to access this data without users' permission. This unmonitored access to sensor data has opened doors to a variety of side-channel attacks on mobile devices.

Previous research has highlighted the security and privacy risks associated with applications accessing various permission-less sensors e.g., motion sensors. These studies have proposed attacks aimed at inferring users' privacy-sensitive information, including touch actions and keystrokes [3–5], passwords and PIN codes [4, 6], and application and webpage fingerprinting [7, 8]. Unfortunately, side-channel attacks remain a significant concern for both users and developers as software updates and new devices are continually introduced to the market. For instance, in iOS, despite restrictions on apps' activity in the background since iOS 5, an application can stay active in the background and continuously access motion sensors' data if it performs specific tasks such as playing audio, receiving updates from a server, and using location services. Given this, an adversary can easily mimic a benign legitimate app such as a fitness and activity tracker to stay active in the background to collect motion data and track a user's activity and behavior. Similarly, modern mixed reality headsets (e.g., HoloLens and Oculus) are equipped with specialized depth sensors that continuously build high-quality maps of users' environment, revealing sensitive information about users and their surroundings. Thus, this ongoing threat underscores the need for careful investigation of the impact of user interaction with mobile devices and their accessories on mobile sensors and the methods that allow adversaries to exploit sensor data for inferring private information.

In tandem, the existence of vast amounts of privacy-sensitive information on mobile and IoT devices and the potential privacy leakages through sensors demand an investigation of effective measures to mitigate the risks posed by unauthorized access to sensor data. Securing this data involves two main challenges. First, the identity of the user operating and interacting with the device must be authenticated. For this, access to modern devices is secured by enabling different authentication solutions, such as PINs/passwords, face recognition, and fingerprint. In recent years, face authentication on mobile devices has become a popular alternative to traditional password-based protection mechanisms due to the ease with which face can be captured with the front camera on these devices and highly accurate face recognition systems. However, most of the existing systems either rely on 2D face recognition systems that are vulnerable to spoofing attacks or employ specialized hardware (e.g., to verify the *liveness* of the subjects [9]. Although such specialized hardware increases the security guarantee provided by these systems, deployment of such specialized hardware components, adding a notch on the screen, is against the current bezel-less trend in the smartphone market and limits its adoption to several existing devices. Therefore, an ideal liveness detection should not rely on any extra hardware components or user involvement and still provide a high-security guarantee.

Second, secure communication channels need to be established between mobile and IoT devices to ensure the confidentiality and security of data exchanges between devices [10]. Existing approaches for establishing these secure communication channels i.e., *pairing methods*, can be mainly categorized into two approaches (1) human-in-the-loop based approaches and (2) context-based solutions. Human-in-the-loop-based pairing approaches require user involvement to perform physical contact between devices or specific actions such as entering passwords, scanning QR codes, or manual interactions [11, 12]. However, these approaches have usability and scalability limitations, especially with an increasing number of devices. To overcome these limitations, context-based pairing schemes have gained interest. These schemes use co-located sensors to establish shared keys based on observed events [13, 14]. However, they are limited to devices with homogeneous sensors of the same modality [10, 15]. To address these limitations, there is a need for pairing schemes that support devices with heterogeneous sensors. These schemes would enable secure communication based on shared context without extensive user involvement or identical sensor capabilities across devices.

Given the imminent need for protecting users' privacy and the "double-edged sword" nature of the sensor data, this dissertation focuses on two important questions that naturally arise: (1) What private information can possibly be inferred from mobile sensor data? and (2) How can we protect mobile systems from unauthorized access to users' data? To answer these two questions, my thesis is:

> *By combining signal processing, computer vision, and machine learning methods, multi-modal sensing can be leveraged to investigate privacy threats and design usable and secure methods for modern mobile and IoT device authentication.*

The first half of this dissertation unveils privacy leakages through mobile sensors using a combination of signal processing, computer vision, and machine learning methods. This dissertation demonstrates two novel permissionless sensor-based side channels on mobile devices and show that leakages through these channels seriously threaten users' privacy. We develop $\mathtt{S}^3$ to show how embedded magnets in stylus pencils impact the mobile magnetometer sensor and can be exploited to infer a users incredibly private handwriting. We then design `LocIn` to infer a users indoor semantic location from 3D spatial data collected by mixed reality devices through LiDAR and depth sensors. These works highlight new privacy issues due to advanced sensors on emerging commodity devices.

The second half of this dissertation explores secure and usable user and device authentication for smartphones and IoT devices. This dissertation develops two systems, `FaceRevelio` and `IoTCupid` to enable reliable and secure user and device authentication to protect users' private information (e.g., contacts, messages, credit card details) on commodity mobile and allow secure communication between IoT devices. First, we design `FaceRevelio`, a novel liveness detection system to protect facial authentication mechanisms on commodity smartphones from spoofing attacks, without requiring effort from the users or any external hardware. Then, we introduce `IoTCupid`, a secure and usable group pairing system for heterogeneous sensing devices in IoT environments. These works enable usable authentication on diverse mobile and IoT devices and eliminate the dependency on sophisticated hardware for a user-friendly authentication mechanism.

1.1 Thesis Contributions

This section presents an overview of the research problems investigated in this thesis. Following the thesis statement above, I make the following contributions:

- S^3**.** With smart devices being an essential part of our everyday lives, unsupervised access to the mobile sensors' data can result in a multitude of side-channel attacks. In chapter 2, we study potential data leaks from Apple Pencil (2$^{\text{nd}}$ generation) supported by the Apple iPad Pro, the latest stylus pen which attaches to the iPad body magnetically for charging. We observe that the Pencil's body affects the magnetic readings sensed by the iPad's magnetometer when a user is using the Pencil. Therefore, we ask: *Can we infer what a user is writing on the iPad screen with the Apple Pencil, given access to only the iPad's motion sensors' data?* To answer this question, we present **S**ide-channel attack on **S**tylus pencil through **S**ensors ($\mathtt{S}^3$), a system that identifies what a user is writing from motion sensor readings. We first use the sharp fluctuations in the motion sensors' data to determine when a user is writing on the iPad. We then introduce a high-dimensional particle filter to track the location and orientation of the Pencil during usage. Lastly, to guide particles, we build the Pencil's magnetic map serving as a bridge between the measured magnetic data and the Pencil's location and orientation. We evaluate $\texttt{S}^3$ with 10 subjects and demonstrate that we correctly identify 93.9%, 96%, 97.9%, and 93.33% of the letters, numbers, shapes, and words by only using motion sensors' data. This chapter originally appeared in the *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (UbiComp 2021)* [16].

- `LocIn`. Mixed reality (MR) devices capture 3D spatial maps of users' surroundings to integrate virtual content into their physical environment. Existing permission models implemented in popular MR platforms allow all MR apps to access these 3D spatial maps without explicit permission. Unmonitored access of MR apps to these 3D spatial maps poses serious privacy threats to users as these maps capture detailed geometric and semantic characteristics of users' environments. In chapter 3, we present `LocIn`, a new location inference attack that exploits these detailed characteristics embedded in 3D spatial maps to infer a user's indoor location type. `LocIn` develops a multi-task approach to train an end-to-end encoder-decoder network that extracts a spatial feature representation for capturing contextual patterns of the user's environment. `LocIn` leverages this representation to detect 3D objects and surfaces and integrates them into a classification network with a novel unified optimization function to predict the user's indoor location. We demonstrate `LocIn` attack on spatial maps collected from three popular MR devices and show that it infers a user's location type with an average 84.1% accuracy. This chapter originally appeared in the *USENIX Security Symposium 2023* [17].

- `FaceRevelio`. Facial authentication mechanisms are gaining traction on smartphones because of their convenience and increasingly good performance of face recognition systems. However, mainstream systems use traditional 2D face recognition technologies, which are vulnerable to various spoofing attacks. Existing systems perform *liveness detection* via specialized hardware, such as infrared dot projectors and dedicated cameras. Although effective, such methods do not align well with the smartphone industry's desire to maximize screen space.

In chapter 4, we present a new liveness detection system, `FaceRevelio`, for commodity smartphones with a single front camera. It utilizes the smartphone screen to illuminate a user's face from multiple directions. The facial images captured under varying illumination enable the recovery of the face surface normals via photometric stereo, which can then be integrated into a 3D shape. We leverage the facial depth features of this 3D surface to distinguish a human face from its 2D counterpart. On top of this, we change the screen via a *light passcode* consisting of a combination of random light patterns to provide security against replay attacks. We evaluate `FaceRevelio` with 30 users trying to authenticate under various lighting conditions and with a series of 2D spoofing attacks. The results show that using a passcode of $1s$, `FaceRevelio` achieves a mean EER of 1.4% and 0.15% against photo and video attacks, respectively. In the worst case, the EER is still low at 1.4% and 0.3% for photo and video attacks, respectively. This chapter originally appeared at the *International Conference on Mobile Computing and Networking (MobiCom 2020)* [18].

- `IoTCupid`. Pairing schemes establish cryptographic keys to secure communication among IoT devices. Existing pairing approaches that rely on trusted central entities, human interaction, or shared homogeneous context are prone to a single point of failure, have limited usability, and require additional sensors. Recent work has explored event timings observed by devices with heterogeneous sensing modalities as proof of co-presence for decentralized pairing. Yet, this approach incurs high pairing time, cannot pair sensors that sense continuous physical quantities and does not support group pairing, making it infeasible for many IoT deployments. In chapter 5, we design and develop `IoTCupid`, a secure group pairing system for IoT devices with heterogeneous sensing modalities, without requiring active user involvement. `IoTCupid` operates in three phases: (a) detecting events sensed by both instant and continuous sensors with a novel window-based derivation technique, (b) grouping the events through a fuzzy clustering algorithm to extract inter-event timings, and (c) establishing group keys among devices with identical inter-event timings through a partitioned group password-authenticated key exchange scheme. We evaluate `IoTCupid` in smart home and office environments with

11 heterogeneous devices and show that it effectively pairs all devices with only 2 group keys with a minimal pairing overhead. This chapter originally appeared in the *IEEE Symposium on Security and Privacy (SP 2023)* [19].

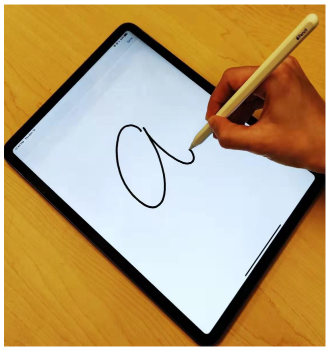
The following chapters describe the design and evaluation details of each of these four systems outlined above. Lastly, chapter 6 presents concluding remarks and discusses the future research directions in mobile security and privacy.

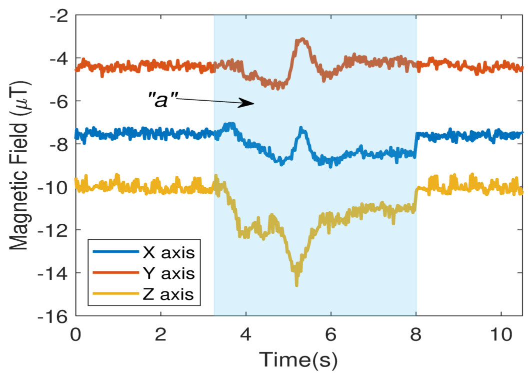
2. S^3:SIDE-CHANNEL ATTACK ON STYLUS PENCIL THROUGH SENSORS

2.1 Introduction

Modern-day smart devices come embedded with various sensors, enabling a vast range of applications in activity recognition, context awareness, mobile health, and productivity. Unfortunately, these sensors are also gateways for unintended information leakage about users' activities. Unauthorized access to users' personal information, habits, behaviors, and preferences through sensor data has long been a major security and privacy concern. Mobile operating systems such as Android and iOS require users' explicit permission to grant access to sensitive data (e.g., GPS locations, microphone, and camera recordings) to an application. However, data from more generic sensors such as magnetometer, accelerometer and gyroscope are less regulated, and often can be accessed by applications without users' permission. Unmonitored access to these sensors' data has recently opened the door to a multitude of side-channel attacks. Prior efforts have unveiled security and privacy concerns that result from applications having access to motion sensors' data. Such works propose attacks targeted at inferring users privacy-sensitive data such as touch actions and keystrokes [3–5], passwords and PIN codes [4, 6], and application and webpage fingerprinting [7, 8]. Unfortunately, side-channel attacks continue to be a major source of concern to users and developers due to frequent software updates and new devices introduced in the market.

With the ever-increasing popularity of large-screen handheld devices, many manufacturers have started equipping their products with *stylus pencils* to improve user experience. In this thesis, we study the recently launched "Apple Pencil" used with one of Apple's most popular devices, the iPad Pro. The Apple Pencil lets users write, draw, and make selections in a variety of generic and custom-designed applications with ease. The second generation of this product launched in 2018 offers a convenient new feature where the pencil pairs with the iPad to charge wirelessly. The pencil's body is embedded with multiple magnets, allowing it to magnetically attach to the iPad's body. In this thesis, we present a novel side-channel attack on the Apple Pencil. The magnetic readings sensed by the iPad's magnetometer are impacted when the user interacts with the screen using the pencil. We show that, given that


(a)



(b)

Figure 2.1. Example of magnetometer readings when a user writes on the iPad using Apple Pencil.

the Apple Pencil is often used to fill text fields in applications [20], an adversary can infer a victim's private data such as passwords, notes, text messages and signatures by only tracking the movement of the pencil through the motion sensors' data.

To illustrate our idea, Figure 2.2b presents the recorded magnetometer data in X, Y, and Z directions when a user writes the character 'a' on the iPad with the Pencil, as shown in Figure 2.1a. We show that an adversary can infer what users are writing on the iPad's screen by observing the fluctuations in the magnetic readings by *only* having access to motion sensors' data. We note that the attack does not rely on any touch information (i.e., the pencil tip position) since iOS does not allow third-party applications running in the background to access this information.

As observed in Figure 2.1, we can readily determine when the pencil is used to write on the screen from the magnetic data. However, to infer the *contents* written on the screen, the magnetic readings require detailed analysis. In our preliminary experiments, we observe various subtle fluctuations in magnetic readings characterized by different letters and words written on the screen. To illustrate, we present the recorded magnetometer data in Figure 2.2 when the letters 'a', 'b' and 'c' are written on the iPad's screen (Figure 2.2a). Figure 2.2b shows the magnetometer readings in three axes, where fluctuations in the magnetometer readings are different for each of the three characters. One intuitive approach to extract

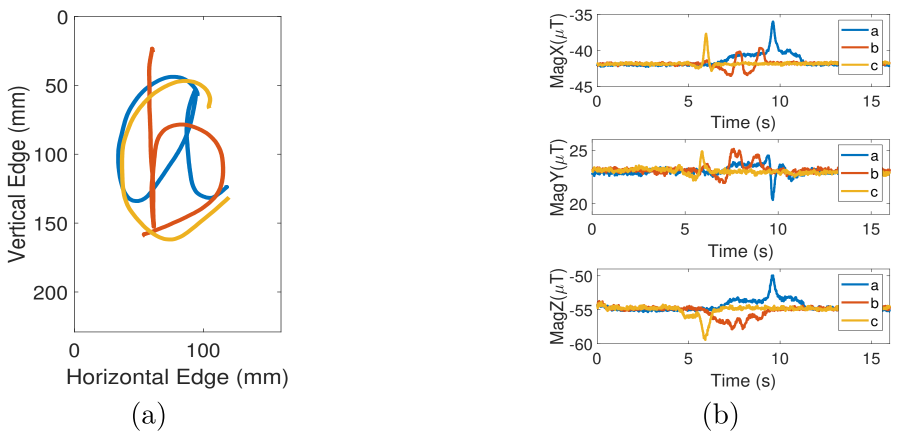


Figure 2.2. The magnetometer readings change when a user writes different characters on the screen.

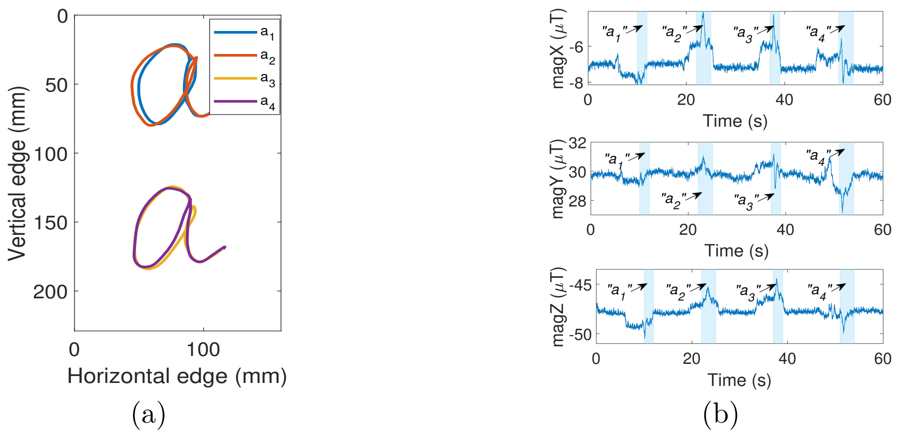


Figure 2.3. The magnetometer readings change when a user writes the same character on the screen.

information from these readings is to train a learning model. This model can then be used to identify useful patterns (e.g., letters, numbers, shapes) from similarities between the magnetic data over time for a given set of characters. Yet, this approach works when the pencil's magnetic impact is consistent when it is held at different locations or with different orientations on the screen for a given character. To determine whether this is the case for Apple Pencil and iPad Pro, we wrote the character 'a' with the pencil at different locations and orientations on the iPad's screen. Figure 2.3a illustrates two letters written at different locations (a_1 and a_3) and two letters in the same location when the pencil is rolled by 180°

(a_3 and a_4). We observe that magnetometer readings vary significantly, although the traces of characters are visually the same.

Based on these observations, we design a tracking algorithm to track the pencil's tip movement using the magnetic field data to identify users' writing. Building such an algorithm requires finding the correct mapping between a given magnetic reading and the pencil tip's location and its orientation with respect to the screen. A common method for finding such mappings is through war-driving. Unfortunately, the mapping space is five-dimensional since we want to track the pencil's 2D location on the screen *while* keeping a record of its 3D orientation. War-driving for all possible locations and orientations of the pencil on the iPad leads to a huge search space and necessitates a huge amount of human effort.

To address this, we reduce the war-driving space by building a 3D magnetic field map *around the pencil* instead of determining the magnetic field for the screen itself. We collect magnetometer data while writing with different orientations of the pencil at different locations on the screen. We employ a computer vision-based approach for pose-estimation to track the pencil's 3D orientation since iOS's touch API does not provide full information about the pencil's 3D orientation. We apply Kalman filtering [21] and smoothing to the estimated pencil orientations to remove noisy data. We then use this orientation along with the pencil tip location and magnetic data to build the magnetic map with respect to the pencil. Lastly, to further improve our magnetic field map precision, we adopt a magnetic vector field reconstruction technique [22], which uses the divergence and curl properties [23] of the magnetic fields for optimizing the reconstruction process. This map generation is conducted offline and does not require information collected from the target user.

Using the designed magnetic field map, we build a multi-dimensional *particle filter* [24] to track the status of the pencil on the iPad's screen, which solely operates with the data collected from motion sensors. The particles' state includes the pencil tip's location and the 3D orientation of the pencil represented as quaternion vectors [25]. We use the human writing behavior to guide the particles initialization and transition (components of our particle filter). Additionally, we used KLD resampling [26] to improve efficiency and incorporated particles' history to handle scattered particle clusters that may exist at the beginning of the tracking process. For an end-to-end attack, we also analyzed the variance in the magnetometer, accelerometer, and gyroscope readings to estimate when the user begins and ends writing on the screen.

We implement our system, S^3 (**S**ide-channel attack on **S**tylus pencil through **S**ensors), on Apple 11" iPad Pro running iOS 12.0. We evaluate S^3 with 10 subjects, where the subjects write different letters, numbers, shapes, and words at different locations of the screen. We show that an adversary (randomly chosen from the subjects) is able to correctly identify 93.9%, 96%, and 97.9% of the written letters, numbers, and shapes, and English words with an accuracy of 93.33%.

The main contributions of this work are summarized as follows:

- We unveil a privacy leakage through motion sensor data in modern tablet devices due to the introduction of stylus pencils with embedded magnets.
- We design a novel tracking algorithm by constructing a magnetic map of the pencil and building a sophisticated multi-dimensional particle filter that can track users' writing.
- We implemented our system on an iPad Pro with Apple Pencil. A thorough evaluation with 10 human subjects demonstrates its high accuracy in uncovering letters, numbers, shapes, and words in a variety of scenarios, without significant overhead.

2.2 Threat Model

We use Apple's iPad and Pencil (2nd generation) to demonstrate a proof-of-concept of inferring information about what the user is writing from motion sensors data. However, our attack can be a threat to any mobile device with a stylus pen support using embedded magnets. The sensitive information being leaked out from user writings can be an unprecedented threat to the confidentiality of user data processed by the writing apps; an adversary can infer passwords, text messages, secure access codes, and other secrets of the user.

We consider an adversary that controls a malicious native application installed on the victim's device, which has access to the motion sensors data and runs in the background during data collection. The native application does not have any information about other running applications and does not have access to system resources. To detail, the iOS applications, by default, have access to the motion sensor data, and only web applications from iOS 12.2 and onwards require explicit user permission. Starting from iOS 5, an application can stay active in the background if it performs specific tasks such as playing audio, receiving updates from a server, and using location services. Given these facts, an adversary can easily mimic a benign legitimate app such as a fitness and activity tracker to stay active in the background to collect motion data. The application periodically logs the motion sensors' data, including accelerometer, gyroscope, and magnetometer readings. The recorded sensor data is stored locally and sent to a remote server for processing. These processes incur minimal energy and computation overhead in order to remain undetected (Detailed in Section 3.7).

We additionally assume that the adversary can obtain or learn a model to capture the users' writing behavior, which is used to predict how the Pencil's movement changes over time through Pencil's previous positions. The adversary could learn such a model by collecting handwriting samples (various letters, numbers, shapes, etc.) from a small number of people (up to 3 users is sufficient for the attack to succeed as detailed in our evaluation in Section 3.7). To build this model, an adversary uses a computer vision-based technique (See Section 2.5.1) to track the Pencil's orientation while logging the Pencil's location and motion sensors' data. We note that the handwriting samples *are not collected from the potential attack victims* and are solely obtained from the adversary and their accomplices. After the model is learned, the adversary uses the model to infer the free-form handwriting of unaware users *solely by collecting motion sensor data.* We emphasize that the Pencil location and orientation data are collected only during the training process. An adversary does not need to access this information at attack time to infer user writings.

2.3 Pencil Tracking: A First Look

In our quest for inferring the user's writing from the sensors' data, we begin by exploring the simplest case: *assume that the user always holds the Pencil in a fixed orientation, i.e., vertically such that the Pencil altitude remains* $90°$*.* In this case, to track how the Pencil moves on the screen, we first determine a mapping between the magnetic readings and the location of the Pencil tip. We define this mapping as the *2D magnetic map.*

2.3.1 2D Magnetic Map

The 2D magnetic field map would be a function, f, that returns the magnetic readings corresponding to a given location on the screen i.e.

$$(m_x, m_y, m_z) = f(x, y) \tag{2.1}$$

where x and y are the coordinates of the Pencil tip location and m_x, m_y and m_z are the magnetic reading in x, y and z directions. We consider the shorter edge of the iPad as x-axis and the longer edge as y-axis.

To build this map, we draw on the screen using the Pencil such that we cover the entire screen. We use a custom drawing application that logs the x and y coordinates of the Pencil on the screen using iOS touch API, at a rate of 120Hz. The magnetic data is recorded in the background at a frequency of 50Hz. The timestamps recorded with the magnetic and touch data are used to align the magnetic readings with each of the touch samples. We start collecting the magnetic data a few minutes before drawing (when the Pencil is detached from the iPad and is not on the screen) to sense the ambient magnetic field. The average of the ambient magnetic data is subtracted from the collected data to eliminate the magnetic impact of the surroundings. Figure 2.4 shows the 2D magnetic field map in the x, y and z directions.

2.3.2 2D Tracking

Once we have built the 2D magnetic map, we ask the question: *how can we track the Pencil location using only the magnetic data?* Here we borrow a standard tracking algorithm, *particle filter*, to track the Pencil's location. Particle filtering is a probabilistic approximation algorithm for state estimation problems [24]. In particular, for object location estimation, particle filter operates by maintaining a probability distribution for the location estimate at time t, which is represented by a set of weighted samples called particles. The state of these particles is updated based on a motion model for the object, and the weights for these particles are assigned at each timestamp by a likelihood model for the recorded sensor data.

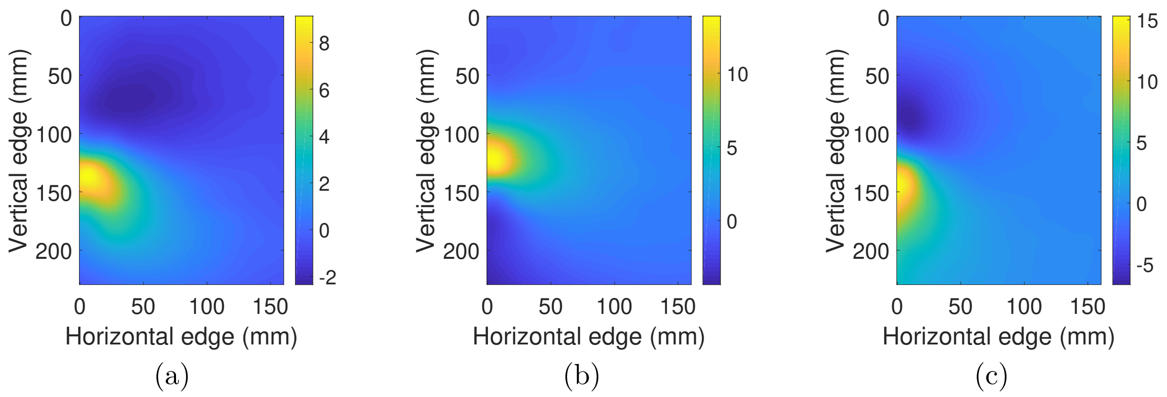


Figure 2.4. 2D magnetic map: the magnetic impact of the Pencil at different locations on the screen in (a) X, (b) Y, and (c) Z dimensions.

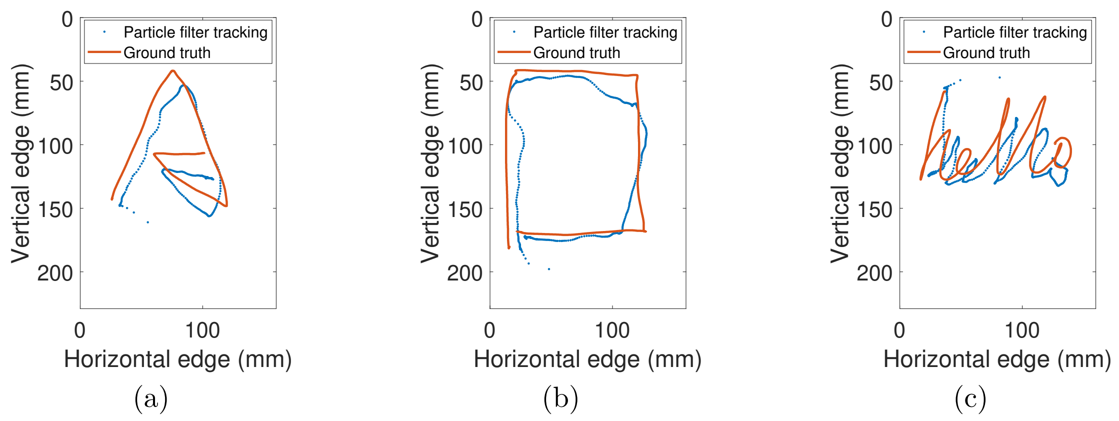


Figure 2.5. Tracking results from a simple implementation of particle filter.

The particles are then resampled at each timestamp using importance sampling to choose a new set of particles according to the weights of the prior samples.

We define the state vector for the particle filter as s_t, representing the Pencil tip location. Here, t represents the timestamp, which is updated at a frequency of 50Hz. For the simple case, we define the Pencil movement model as:

$$s_{t+1} = s_t + b_t \tag{2.2}$$

where b_t is a random perturbation added to the state, with a Gaussian distribution.

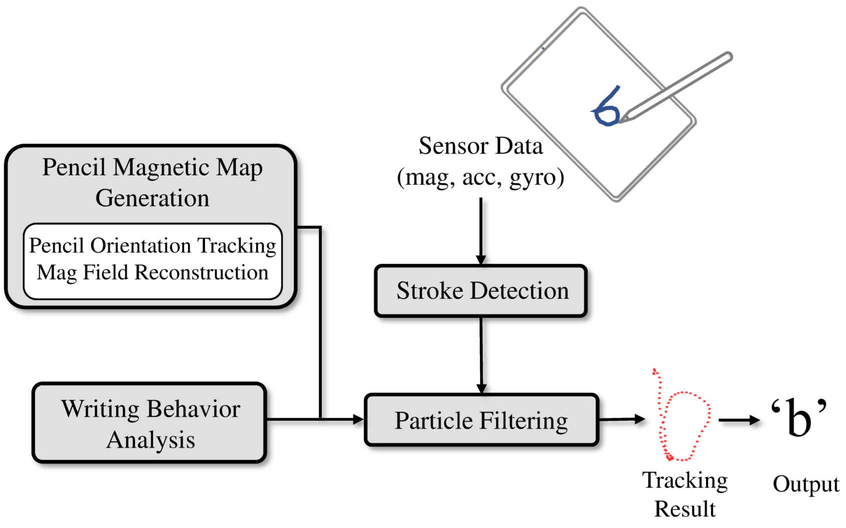

Pencil Magnetic Map Generation
Pencil Orientation Tracking
Mag Field Reconstruction
Sensor Data (mag, acc, gyro)
Stroke Detection
Writing Behavior Analysis
Particle Filtering
Tracking Result
‘b’
Output

Figure 2.6. System architecture.

Weights are assigned to each particle at each timestamp with the function:

$$w_t^{\mathrm{i}} = \exp\left(\frac{(m_t - r_t)^2}{\sigma}\right) \tag{2.3}$$

Here r_t is the magnetic readings obtained from the magnetometer at time t and m_t is the queried magnetic readings given state is s_t, using Equation 2.1. The location of the Pencil tip at each timestamp is the weighted average of the particles.

Figure 2.5 shows the results of this simple implementation of particle filter when used to track a letter (*‘A’* - Figure 2.5a), a shape (a square - Figure 2.5b) and a word (*‘hello’* - Figure 2.5c). Even though we assumed that the Pencil orientation is fixed for this case, the tracking results show promise that we can infer what the user is writing.

2.4 System Architecture

This section presents the functional overview of our system S^3as shown in Figure 2.6. We will detail each component of our system in Section 2.5.

To launch a realistic attack, our system should find out what the user is writing in their natural handwriting style, unlike the fixed orientation case in the previous section. For this purpose, we need a mapping between the magnetic data collected and the status of the Pencil, which now includes location and orientation, i.e., 5 degrees of freedom ($\langle x, y \rangle$ location and 3D orientation of the Pencil). We generate this map through war-driving and reduce the effort involved by building the magnetic map around the Pencil. We extend computer vision techniques to track the Pencil's orientation by attaching a checkerboard on top of the Pencil and recording the Pencil movement with a camera. A magnetic field reconstruction technique [22] is used to remove noise from the collected magnetic data and generate a continuous magnetic field map for the Pencil. This magnetic map for a given iPad and Pencil can be built solely by the attacker. It is a one-time effort since the same map is used to track the user's handwriting in different locations and environments.

We also build a model for writing behavior by writing different letters, numbers, and shapes at different locations on the screen multiple times while tracking the Pencil orientation with a camera, as previously mentioned. This model finds the relationship between orientation and location changes while writing via linear regression. It predicts how these parameters should change at a given timestamp with respect to the Pencil's previous position. Similar to the magnetic map, this writing behavior model is also trained on data collected from the attacker and their accomplices and does not involve the victim.

The motion sensors' data (including magnetometer, accelerometer, and gyroscope data) collected by the attacker's application is sent to a back-end server for processing. We apply stroke detection on this motion data to detect the start and end of a stroke. The magnetic data corresponding to the detected stroke is then fed into a multi-dimensional particle filter (Section 2.5.2), together with the writing behavior model. The writing behavior model helps in predicting the next state of the particle filter. The particle filter outputs the tracking traces of the stroke. An attacker then looks at the tracking result to guess what the user wrote.

2.5 System Design

This section describes the details for tracking when the users write with their natural handwriting style. To this end, we first describe our approach for building the Pencil's magnetic map. This is followed by the details of our tracking algorithm and stroke detection.

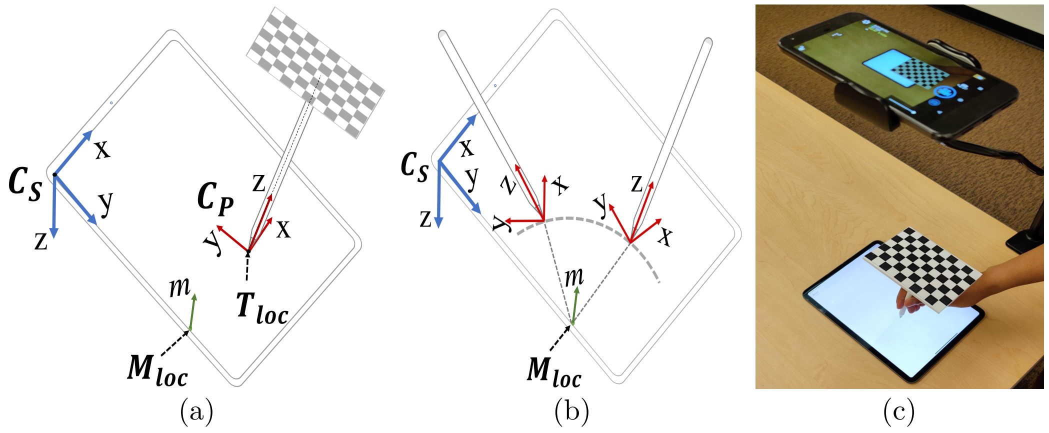


Figure 2.7. (a) Screen and Pencil coordinate systems and the location of the magnetometer. (b) An example that the location of the magnetometer in the Pencil coordinate system is the same, even when the Pencil's locations and orientations are different. (c) Setup for tracking the Pencil orientation.

2.5.1 Pencil Magnetic Map Generation

The magnetic field map needed for tracking the Pencil movement is a function, f, mapping the location of the Pencil tip and its 3D orientation to the magnetic readings in x, y and z directions i.e.

$$(m_x, m_y, m_z) = f(x, y, orientation) \tag{2.4}$$

To build this magnetic map, we first want to introduce the coordinate systems for the iPad screen and the Pencil and clarify the notion of the Pencil's orientation.

Figure 2.7a shows the coordinate systems for the iPad screen, C_s. The top-left corner of the iPad screen is the origin for C_s. The Pencil tip location reported by the iOS touch API is also in the screen coordinate system. The Z axis of C_s points downwards based on the right-hand rule. M_{loc} and T_{loc} are the locations of the magnetometer and Pencil tip in C_s, respectively. We assume that M_{loc} is known beforehand. Figure 2.7a also shows the Pencil's coordinate system, C_p. The Z-axis of the Pencil runs vertically through the Pencil's body. We define the orientation of the Pencil as its axes expressed as vectors in the C_s denoted by X_{ps}, Y_{ps} and Z_{ps}.

We can now define equation 2.4 as

$$(m_x, m_y, m_z) = f(T_{loc}, X_{ps}, Y_{ps}, Z_{ps}) \tag{2.5}$$

For the free-form writing case, we need to move the Pencil in all possible locations and orientations to get the magnetic impact for each location and orientation. This requires a huge amount of human effort. Hence, we focus on opportunities to reduce the war-driving space for building the magnetic field map.

So far, we have considered building the magnetic map from the iPad's perspective, i.e., the Pencil's location and orientation, and the magnetic readings are in C_s. Another way to look at this problem is to build the map from the Pencil's perspective. Recording the Pencil movement's magnetic impact on the screen is essentially the same as keeping the Pencil stationary and moving the magnetometer around it. In other words, for any position of the Pencil in C_s, we can find the corresponding magnetometer position in C_p. Given that we know the location of the magnetometer, M_{loc}, in C_s, if the Pencil tip is at location, T_{loc}, the 3D position of the magnetometer in C_p, M'_{loc} can be represented by

$$M'_{loc} = P_s * (M_{loc} - T_{loc}) \tag{2.6}$$

where P_s is the tuple (X_{ps}, Y_{ps}, Z_{ps}) representing Pencil's axes in screen space.

Similarly, we can also transform the magnetic readings in C_s, m to magnetic readings in C_p, m' by

$$m' = P_s * m \tag{2.7}$$

Building a map around the Pencil reduces the search space for war-driving since M'_{loc} can be the same for more than one Pencil position in C_s. This means that if we express equation 2.6 as

$$T_{loc} = M_{loc} - P_s^T * M'_{loc} \tag{2.8}$$

we can show that there are multiple pairs of T_{loc} and P_s which would correspond to the same M'_{loc} in C_p. Figure 2.7b shows an example of this case. Even though the T_{loc} and P_s

are different for the two Pencils, M'_{loc} is the same for these two Pencils in C_p. Due to this redundancy, if we cover one of these Pencil positions while war-driving, we also determine the magnetic impact for a large number of other positions automatically. This greatly reduces the number of positions we need to cover through war-driving. Additionally, instead of collecting only one sample for each Pencil location and orientation, we can collect multiple samples for a position in C_p when we move the Pencil at different locations on the screen.

Considering the benefits mentioned above, we build the magnetic map around the Pencil, which can now be expressed as

$$m' = f(M'_{loc}) \tag{2.9}$$

For this purpose, we use our setup shown in Figure 2.7c to track the orientation of the Pencil while randomly drawing on the screen for a few hours. We record the Pencil movement with a camera placed at a distance above the iPad and looking down upon the screen. In this random drawing, we rotate and move the Pencil such that we can cover the entire screen and maximum possible orientations. Similar to the procedure for building the 2D magnetic map, we collect the magnetic data in the background at 50Hz and touch data, T_{loc}, from iOS API at 120Hz. The touch API only provides information about the azimuth and altitude angles [27] of the Pencil. However, the Pencil rotation around its Z-axis also affects the magnetic readings sensed by the magnetometer. Hence, the Pencil orientation, P_s, cannot be obtained without any ambiguity directly from the touch API. Therefore, we employ a computer vision approach to track the orientation.

The camera acts as a bridge between the screen and the Pencil. By finding the camera's orientation in C_s and then finding the Pencil orientation in the camera's coordinate system, we can finally find the orientation of the Pencil in C_s. To find the orientation of the camera in C_s, we use a standard camera calibration technique [28]. Once the camera is calibrated, we use PnP algorithm [29] to find the orientation of the checkerboard in the camera's coordinate system.

Due to the noise in camera calibration and pose estimation steps, the orientation obtained is noisy. To remove this noise, we apply Kalman filtering [21] and smoothing to the orientation obtained over time. Since the checkerboard's origin is attached to the Pencil end, the 3D axes

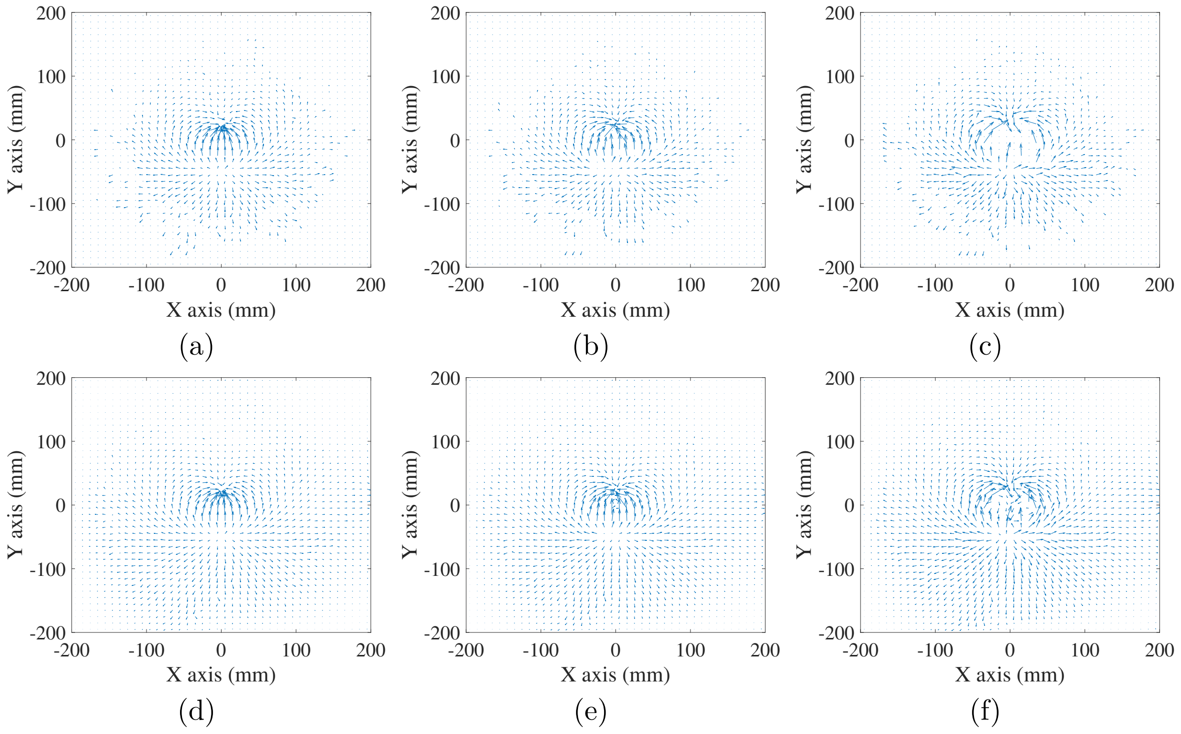


Figure 2.8. Pencil magnetic map for $z = -5$mm (left), $z = 0$mm (center), and $z = 5$mm (right), before (top) and after (bottom) optimization.

of the checkerboard in C_s obtained after smoothing are indeed the Pencil axes in C_s as well. Therefore, by tracking the checkerboard axes, we can track the orientation of the Pencil as it moves on the screen.

We now have the touch data from the iOS API, the orientation data from the camera, and the magnetic data from the magnetometer. For each touch sample, we transform the M_{loc} and sample's corresponding m to C_p using equations 2.6 and 2.7 respectively. These transformed magnetic readings in C_p are finally quantized into $5mm$ by $5mm$ grids based on the 3D location of the M_{loc} corresponding to the reading. The average of all the magnetic data samples is computed for each grid. Figure 2.8 (top) shows the XY plane of this 3D map built around the Pencil for $z = -5mm$, $z = 0mm$ and $z = 5mm$ in C_p.

Here, we emphasize that we remove the magnetic impact of the ambient environment from the collected magnetic data before generating this map. As a result, the same magnetic map can be used to track the Pencil movement while the victim is using their iPad in different locations. Hence, this process is a **one-time effort** and does not involve any input from the victim. We also note that the setup shown in Figure 2.7c for Pencil's orientation tracking is only used for offline map generation and is not needed at attack time to infer victim's handwriting.

Map Optimization: As shown in Figure 2.8 (top), some of the grids in the 3D map have missing or noisy data. Instead of using a simple interpolation approach, we adopted a vector field reconstruction technique that considers the divergence and curl of the vector field [22].

According to Maxwell's equations bounding the magnetic field [30], the regularization parameter λ_c should be small while λ_d should be large. Considering the noise in the magnetic readings, we perform a grid search to determine the optimal values of regularization parameters, guided by the optimal values given in [31] for different signal to noise ratios. The resulting 3D magnetic map from this approach is continuous with reduced noise. Figure 2.8 (bottom) shows the XY plane of this 3D map for $z = -5mm$, $z = 0mm$ and $z = 5mm$ after reconstruction.

2.5.2 Pencil Tracking

With the reconstructed 3D map, we are now prepared to describe our system for tracking the Pencil's location and orientation as a user writes on the iPad screen. Similar to the 2D case, we use particle filter to track the Pencil movement. However, now the state vector for the particle filter also has to take into account the Pencil's orientation.

Earlier, we defined the Pencil orientation as its axes, (X_{ps}, Y_{ps}, Z_{ps}) , in C_s. This definition is intuitive and works well for magnetic map generation but using this definition in particle filter is complicated. Another common way to represent the orientation of a rigid body is to use the Euler angles, which represent the 3D orientation of an object using a combination of three rotations about different axes [25]. Although intuitive, Euler angles suffer from singularities and give erroneous orientation representation when used to track an object's successive rotations. Fortunately, there is another formal representation for 3D rotation, which is the quaternions [25], which allows easier interpolation between different orientations. Quaternions encode any rotation in a 3D coordinate system as a four-element vector, where the squared sum of the four elements is equal to 1. Using quaternions, we can describe how the orientation changes sequentially more efficiently and smoothly [32]. Hence, for using the quaternion representation to describe the Pencil's orientation, our particle filter has to consider 6 parameters ($\langle x, y\rangle$ location of the Pencil tip and the Pencil orientation as 4-element quaternion vector) to track the Pencil movement, instead of 5.

To consider the orientation, we define the state vector for each particle at time t as:

$$s_t = < x_t, y_t, q_{1t}, q_{2t}, q_{3t}, q_{4t} > \tag{2.10}$$

Here, q_{1t}, q_{2t}, q_{3t} and q_{4t} represent the four scalar values for the quaternion representation of the Pencil's orientation at time t.

So far, we have considered that the Pencil can move in any orientation. However, in practice, the movement of the Pencil is limited by the range of motion of the human hand and wrist. Therefore, based on our observations of the human handwriting behavior, we limit the range for the Pencil's possible orientation. We assume that the altitude range of the Pencil is 30° to 90°. Similarly, for the azimuth angle, we specify the range to be within 60° to 170°. We do not limit the rotation of the Pencil around its Z axis. These ranges are determined by analyzing the minimum and maximum values of these angles observed in the data collected by the attacker and their accomplices for training the writing behavior model (described below). We specify the ranges for orientation in terms of the altitude and azimuth for ease of understanding, but these ranges are actually checked in the quaternion representation.

Algorithm 1 describes the algorithm for Pencil tracking.

Particles initialization: We begin by initializing the particles S_0^{i} where $\mathrm{i} = 1, 2, ...N$. Since 6 parameters now determine the state, we need a huge number of particles to cover the entire state space. We employ a grid search approach on data collected from a small set of users (e.g., data collected for training writing behavior model which is not included in our evaluation set) to determine the optimal number of particles initially. To detail, we generate candidates from a grid of parameter values specified from 10000 to 10000000 with increasing the number of particles by 10000 in each iteration. We evaluate each of these particle values on the users' data and use the parameter that leads to highest accuracy at the elbow point [33] to prevent overfitting. Based on our search results, we set the number of

Algorithm 1 Pencil Tracking

```
1: Initialize particles $S_0^{\text{i}}$ where i $= 1, 2, ...N$ using orientation constraints
2: Compute the weights
3: Select top $K$ particles with the highest weights
4: for t $\geq 1$
5: Sample $S_t^{\text{i}}$ using $S_{max(t-3,1):t-1}^{\text{i}}$ based on writing behavior model
6: Update particle history $S_{0:t}^{\text{i}} \leftarrow (\overline{S_{0:t-1}^{\text{i}}}, S_t^{\text{i}})$
7: Compute the weights
8: Resample $S_t^{\text{i}}$ to obtain $K_a$ updated particles
9:
10: end for
11: Track Pencil using $S_{0:t}^{\text{i}}$
12: function ComputeWeight(s)
13:     $P_s \leftarrow s$
14:     $M'_{loc} \leftarrow P_s * (M_{loc} - T_{loc})$
15:     Query Pencil magnetic map using $M'_{loc}$ to obtain $m'$
16:     $m \leftarrow P_s^T * m'$
17:     $w \leftarrow \exp(-\frac{(m-r)^2}{2\sigma^2})$
18:     return $w$
19: end function
```

particles, N, to 5000000. The x and y location for these particles are uniformly drawn from within the range of the iPad screen. Altitude (θ_1) , azimuth (θ_2) and rotation around Z axis (θ_3) are drawn uniformly from the range $30° - 90°$, $60° - 170°$ and $0° - 360°$ respectively. From these values for the three angles, we compute the axes representation for the Pencil in C_s. These axes are converted into quaternions and are included in the state vector. Once we have found S_0^{i} meeting our criteria, we compute their weights using the $ComputeWeights$ function. For efficient computation, we specify a bound on the maximum number of particles at each timestamp as $K = 50000$. This bound is also found through the same grid search approach described above. Therefore, from S_0, we choose K particles which have the highest weights.

Movement model: We define the model for Pencil's movement for natural handwriting as:

$$s_{t+1} = s_t + v_t + b_t \tag{2.11}$$

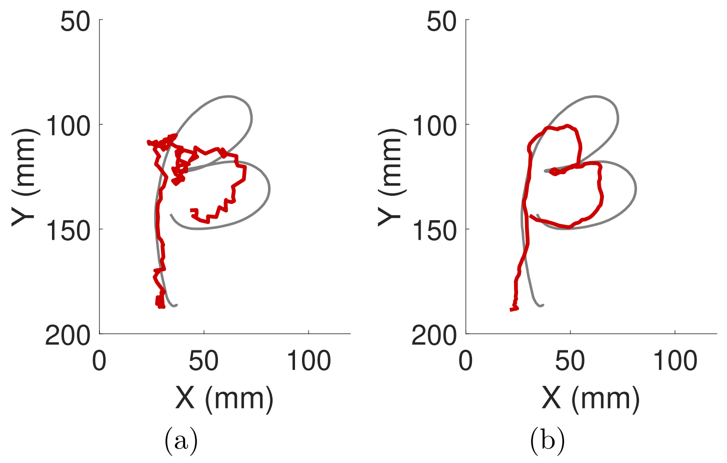


Figure 2.9. Tracking results without (a) and with (b) writing behavior model.

where v_t is the change in particles' states, and b_t is the random perturbation added to the state. Previously, for the fixed orientation case in Section 2.3.2, we used $v_t = 0$ since the state space was small. For this case, we build a **writing behavior model**, which uses the changes in state from the last three timestamps to predict the state change for current timestamps using linear regression. To train this model, we write different letters, numbers, and shapes at different locations on the screen. We use a camera to track the Pencil's orientation and iOS touch API to track location, as described in Section 2.5.1. This camera setup is only used while collecting training data for this model and is not used at attack time. Data collected from a small number of users (3 in case of our evaluation) is sufficient for training this model. We capture the relationship between location and orientation change in human handwriting from this location and orientation data. Figure 2.9b shows the result when this model is used in state transition. We can observe that the tracking result is more accurate and smoother than simply adding random perturbation to the last state (Figure 2.9a).

Computing particle weights: Here we transform the quaternions in the state vector to Pencil's axes, P_s, in C_s. We use these axes to find M'_{loc} using equation 2.6 in Section 2.5.1 to query our Pencil magnetic map. In this equation, T_{loc} is the x and y location in the state

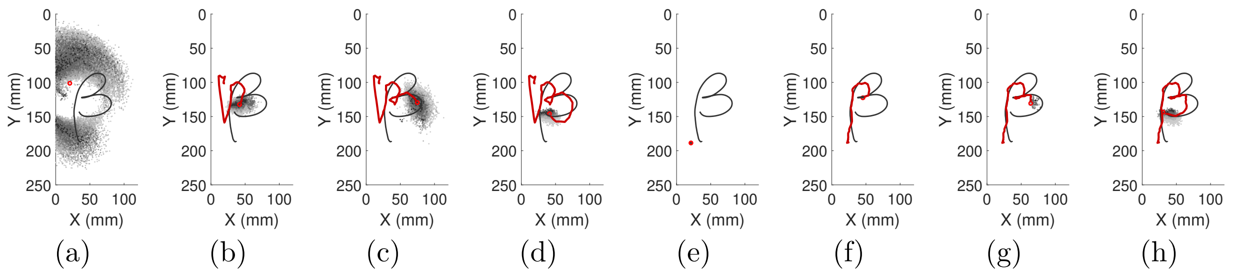


Figure 2.10. (a-d) shows how adaptive particle resampling chooses the number of particles over time. (e-h) show the estimation of Pencil's location when history of particles is used.

vector. The result of the query is m', which is the magnetic reading in C_p. This reading is converted to C_s using:

$$m = P_s^T * m' \tag{2.12}$$

Weights are assigned with the function:

$$w = \exp\left(\frac{(m-r)^2}{2\sigma^2}\right) \tag{2.13}$$

where r is the magnetometer reading corresponding to the given state.

Adaptive particle resampling: While a huge number of particles, in our case $K = 50000$, are needed to determine the initial Pencil position accurately, a smaller number of particles can suffice for tracking once the particles' states start to converge. For this purpose, we use the Kullback-Leibler distance (KLD) [34] resampling [26] approach to decide the number of particles, K_a, on the fly for every timestamp. KLD resampling operates by choosing the number of particles to resample such that the KLD between the particles' distribution before and after resampling does not exceed a predefined threshold. If the particles are widely spread, a large number of particles are used, whereas if the particles are more concentrated, a smaller number is resampled. We use a batched approach to increase the sample size for improving the efficiency of the resampling process. Figures 2.10a- 2.10d show how the resampling phase operates over time. In Figure 2.10a, the particles are scattered into two separate clusters;

hence a large number of particles is resampled. Overtime, once the particles start converging, the number of samples selected at each timestamp decreases as shown in Figure 2.10b- 2.10d.

Pencil location estimation: In addition to keeping track of the state, each particle also carries its history $S_{0:t-1}$ along with it for all timestamps $t \geq 1$. For the simple case of location tracking, we directly used the centroid of the particles to determine the Pencil's position at each timestamp. However, in this case, given the huge state space, the particles might exist in various clusters in different regions of the space, making it difficult to track using just the centroid. For example, in Figure 2.10a, there are two clusters of particles, and the centroid of these clusters (shown as red circle) does not match well to the ground truth Pencil location (i.e., bottom part of the character 'β'). In contrast, if we choose the maximum weighted particle in the last timestamp and use its history as the location estimate, we observe that all the particles converge at the beginning. This means that using the history of the particles returns us a path for the Pencil's movement with high accuracy. Figure 2.10e-2.10f shows the path retraced from the history of particles (red circle in Figure 2.10f).

2.5.3 Stroke Detection

To track a user's writing, we need to identify the precise period during which the user is writing from the collected motion sensors' data. To achieve this goal, we design a *two-step stroke detection* algorithm.

Step 1: Rough estimation: As shown in Figure 2.11(a), the magnetic field around the iPad will fluctuate when a user is writing, while it is stable when the Pencil moves away from the screen. Using this fluctuation, we first generate a rough estimate of the period when the user is writing. The basic idea is to use a sliding window with an appropriate size that contains a sequence of magnetometer data and move the window along the timeline. We mark the midpoint of this window as the starting timestamp of a stroke when the magnetic data variance is larger than a predefined threshold. Similarly, the ending timestamp is marked as the center of the window when the variance is below the threshold. We analyzed the data collected for training the writing behavior model for determining the optimal values for the window size and the threshold and set them to 100 and 0.12, respectively.

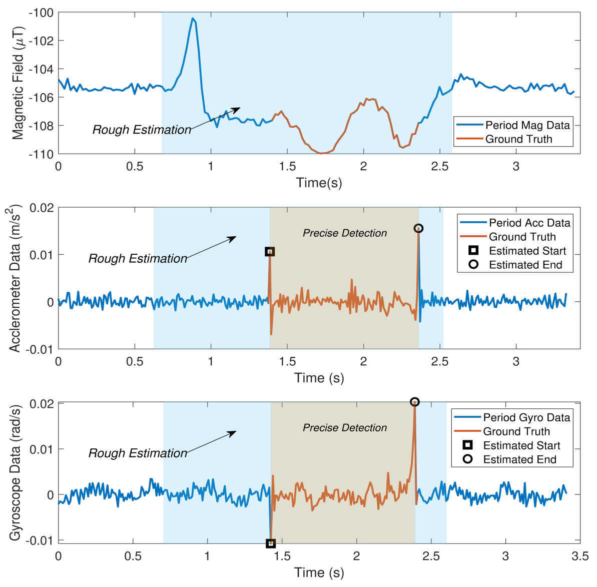


Figure 2.11. Stroke detection to determine the beginning and end of a stroke.

Step 2: Precise detection: A rough estimation is able to help us identify time periods which may contain a stroke. However, since the magnetic field starts fluctuating when the pencil is moving close to the iPad, it is difficult to identify the precise beginning and end of a stroke through magnetic data alone. Therefore, we utilize accelerometer and gyroscope data to detect the strokes more precisely. The intuition behind this step is that the iPad will vibrate slightly at both moments when the Pencil touches and is lifted from the screen. As shown in Figure 2.11(b) and (c), the beginning and end of the stroke are visible as prominent peaks in the accelerometer and gyroscope data. Hence the same threshold-based approach can be used on this data to identify the stroke with higher precision.

2.6 Evaluation

2.6.1 Data Collection and Implementation

We evaluated S^3 on an Apple 11" iPad Pro running iOS 12.0. We implemented two applications for conducting our experiments. The first application, A, acts as the malicious application installed on the victim's iPad, mimicking a legitimate application (such as a fitness tracker) that accesses the motion sensors' data and stays alive in the background (e.g., by use of location services). Application A logs the magnetometer readings at 50Hz, and accelerometer and gyroscope data at 100Hz in the background. The second application, B, mimics a chatting application with a text input region in the lower half of the iPad screen when it is used in landscape mode. This is where the user writes using the Apple Pencil. The input region is divided into 3 equally sized grids. This application acts as the application under attack from application A. The sensor data collected from application A is stored on the iPad during the data collection process and is transferred to a server for analysis. During our experiments, we observe that on average, with a fully charged battery, this application consumes less than 10% of the battery while logging the motion sensor's data, over a duration of 3 hours. We use our own custom application (application B) for writing to record the touch API data for ground truth. We also tested these applications on the latest iOS version (14.0) to validate our assumptions regarding required permissions.

Three authors (pretending to be attackers) collected magnetic data for the Pencil's magnetic map generation while randomly drawing on the iPad screen for a total duration of 3 hours. We used the setup shown in Figure 2.7c to record the orientation of the Pencil and the touch API's data for recording the Pencil's location. The three authors also wrote/drew the 26 lowercase alphabet letters (*a*-*z*), 10 numbers (0-9), and five different shapes (square, triangle, circle, heart, and star) twice in each grid in the input region of the application B. This data was used to train the writing behavior model described in Section 2.5 and was not a part of the evaluation set.

To evaluate S^3, we conducted experiments with 10 subjects, recruited by advertising on the university campus after IRB approval. These subjects included 6 females and 4 males. During the experiments, subjects were seated next to a table on which the iPad was placed. We asked each subject to use the Pencil to write the 26 lowercase alphabet letters (*a*-*z*), 10 numbers (0-9) and five different shapes (square, triangle, circle, heart and star) twice in each grid in the input region. The subjects were guided to write in their natural handwriting style and we did not impose any restriction on the size of the strokes within the input region. In total, we collected the motion sensor data for 1560 letters, 600 numbers and 300 shapes.

In addition to letters, numbers, and shapes, we also tested our system's ability to track words. We asked five subjects to write words of lengths varying between three to six letters, where each word consisted of lowercase letters randomly selected from the English alphabet. We instructed each subject to write 30 words for each of the four word lengths. We also investigated our system's performance in detecting legitimate English words. For this purpose, we asked these subjects to write 30 words each, randomly selected from the 500 most commonly used English words [35]. These words represented a diverse set of words varying in length between 3 to 8 letters with an average length of 5 letters.

To demonstrate the practicality of our system, we also conducted experiments with different positions of the iPad and analyzed the impact of various environment settings.

We randomly selected one of the volunteers to act as an attacker and showed S^3's Pencil tracking results to the attacker for each of our experiments. The attacker was given three guesses to identify each stroke. Hence our system's accuracy is determined as the number of strokes correctly identified by the attacker in the first three guesses. We use this top-k guess method considering the similarity in shapes of some letters in the English alphabet.

2.6.2 Performance Results

We answer the following questions to evaluate S^3's performance:

1. How well can S^3 detect the letters, numbers and shapes?
2. What is the detection accuracy for different letters, numbers, and shapes, and why?
3. Is S^3 able to detect words?
4. Can S^3 detect continuous strokes?
5. How does the location of the Pencil tip affect the performance of S^3?
6. Does the performance of S^3 change across users?
7. How does the positioning of the iPad affect S^3's performance?
8. How do the environmental factors impact the performance of S^3?

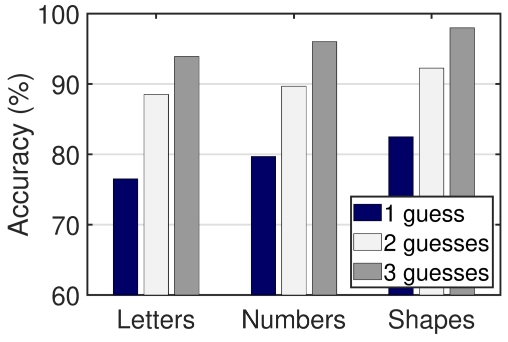


Figure 2.12. Overall accuracy of the correctly guessed letters, numbers and shapes in 1, 2 and 3 guesses.

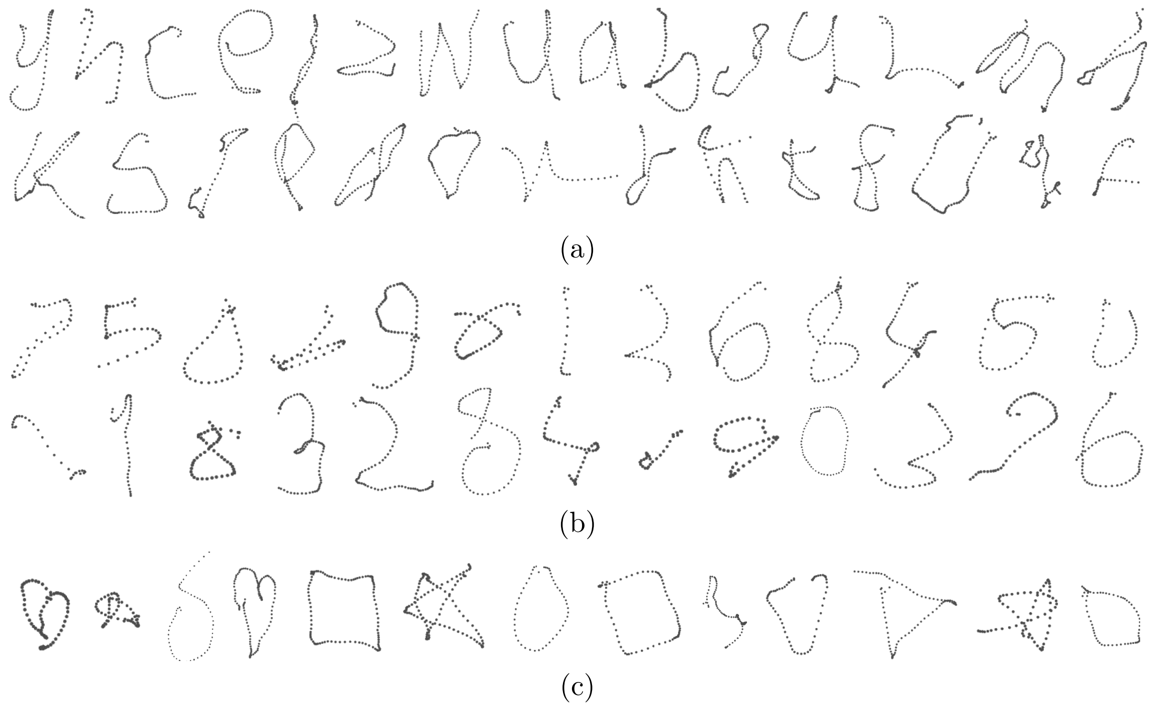


Figure 2.13. Tracking result examples: (a) letters, (b) numbers, and (c) shapes.

9. How does S^3 perform compared to other machine learning techniques?

(1) How well can S^3 detect the letters, numbers, and shapes?

We evaluated our system's performance by randomly selecting one of the volunteers acting as an attacker to guess what the users wrote from S^3's Pencil tracking results. The tracking results for different strokes were shuffled for this process to remove any possible bias. We

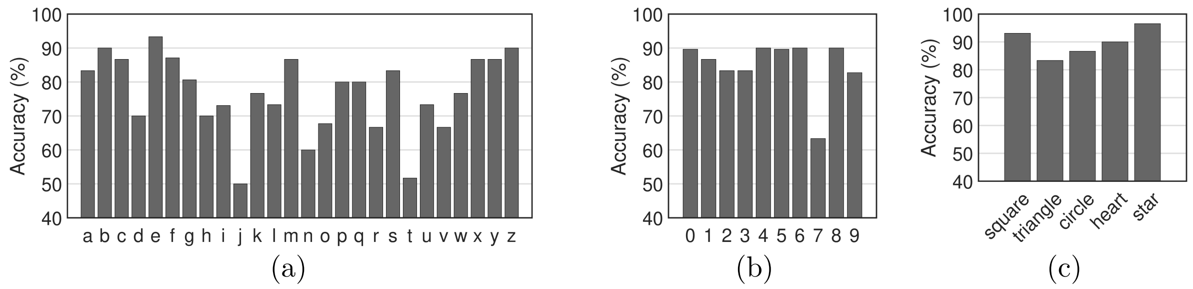


Figure 2.14. Accuracy of the first guess for (a) each letter, (b) each number, and (c) each shape.

recorded the top 3 guesses from the attacker for each stroke. Figure 2.12 shows the overall detection accuracy when the attacker's 1^{st}, 2^{nd} or 3^{rd} guess is correct for different strokes. The attacker correctly guessed 93.9%, 96%, and 97.9% of the letters, numbers, and shapes, respectively. This detection accuracy is based on whether any of the 3 guesses for a stroke matched with the actual letter, number or shape. Figure 2.13 shows a set of examples from the Pencil traces generated by S^3 solely using the motion sensor data. We encourage the reviewers to guess what letter, number, or shape is shown in each example. We give the correct answers at the end of this chapter.

(2) What is the detection accuracy for different letters, numbers, and shapes and why?

We analyzed how different letters, numbers, and shapes contributed to the overall detection accuracy reported earlier. Figure 2.14 shows the detection accuracy for each letter, number, and shape based on the attacker's first guess. Here, we can clearly see that some letters, numbers, and shapes are detected better than the others in the first guess. For example, letters like 'b', 'e', and 'z' have a high detection accuracy. In contrast, letters like 'i', 'j', and 't' have lower accuracy. We found that these letters are usually written in two strokes. Since S^3 does not consider two strokes separately, the letter's shape is not obvious in the Pencil's trace. This can also be seen in Figure 2.13a, where it is difficult to recognize the letter 'j' (third trace in the bottom row). Apart from this observation, some letters like 'h' and 'n' have a low accuracy of 70% and 60% respectively because of the similarities in their shapes. The second letter in the top row of Figure 2.13a can be inferred as ‘h’ although it is actually ‘n’. We observe similar patterns for ‘r‘ and ‘v’ (67% accuracy for both) whose traces are similar to each other as shown in Figure 2.13a (‘v’ - third last letter in row 1, ‘r’ - 8th letter in row 2). Similarly, the number ‘7’ can be confused with the number ‘1’. We note that the letters, numbers, and shapes whose traces are distinct from others are detected with high accuracy in the first guess. Examples of such cases are letters ‘m’, number ‘8’, and star shape.

Given that we did not impose any restriction on subjects’ handwriting styles, we observed that the same strokes written by the same subject often varied slightly in size or style (e.g., slanted). However, since S^3 does not rely on a specific handwriting style for accurate recognition, these variations do not affect its accuracy.

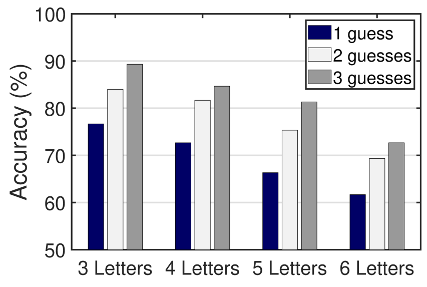


Figure 2.15. S^3’s accuracy in detecting 3 to 6 letter words.

(3) Is S^3 able to detect words?

To answer this question, we shuffled the tracking results for all the words collected from the 5 subjects and presented them to the attacker as described previously. Figure 2.15 shows S^3’s detection accuracy for the random words of varying lengths. The attacker correctly guessed 89.3% and 84.67% of the 3 and 4 letter words. Given the completely random nature of these words, we observe that the number of correctly guessed words decreases as the words’ length increases. This is due to the similarity in the tracking results of different letters, as described above, which becomes more prominent in longer words. However, human handwriting in practice mainly consists of legitimate English words where little randomness is involved. For our experiments with the commonly used English words (three to eight-letter words), the attacker could guess the words with an accuracy of 93.33% in the first three guesses. Figure 2.16 shows a set of examples of the tracking results for this case. We found

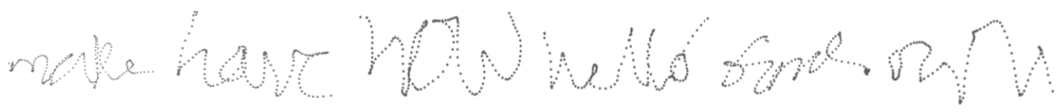


Figure 2.16. Examples of tracking results for the words.

that the tracking results, in some cases, are noisy, making it difficult to guess all the letters in words. An example of this observation is seen in the rightmost word in Figure 2.16. Yet, the attacker can guess the words correctly in these cases since she is able to infer a few letters of the word from the tracking results, which allows her to figure out what the complete word is.

(4) Can S^3 detect continuous strokes?

To evaluate continuous stokes, we conducted an end-to-end attack with S^3. To detail, we consider a real-world scenario where the victim uses the Apple Pencil to write a text message in a chatting application. The attacker's goal is to infer the text message (English sentences) based on the magnetometer's readings collected by the malicious application running in the background.

In these experiments, we invited three volunteers to write 10 different text messages on the iPad while sitting and holding it in hand. The subjects were asked to write valid English sentences, used in common text message conversations (without punctuation and emojis), with up to 8 words in each case. The number of words in the sentences written by the volunteers ranged from three to eight, with a median of five words. S^3's stroke detection component was used to separate the magnetic readings for different words within each sentence and fed into the particle filter. The attacker was able to guess 66.67% of all the words collected in the first three guesses through the tracking results (42.5% in the first guess and 59.1% in two guesses). We found that the estimated beginning and end of some strokes are not very precise when the time gap between different words while writing a sentence is small. Therefore, the tracking results for these strokes are noisy, making it difficult for the attacker to guess the correct word. However, the attacker was able to guess 80% of the sentences correctly when the tracking output of all the words in a given sentence were shown together as part of a single sentence. Thus, the context helps the attacker infer the complete sentence correctly even when the tracking results of individual words are noisy.

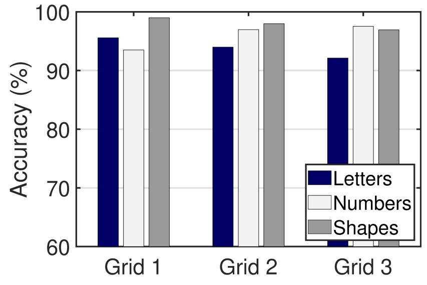


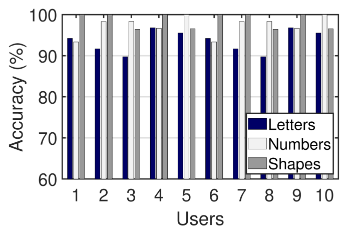
Figure 2.17. Accuracy of detecting letters, shapes, and numbers across different locations on the iPad screen. Grid 1, 2, and 3 are the left, center, and right part of the input region.

(5) How does the location of the Pencil tip affect the performance of our system?

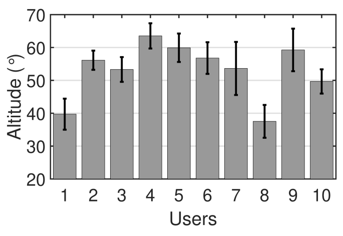
As mentioned earlier, we split the input region into 3 grids during the experiments. This enabled us to evaluate S^3's performance when a user writes on different iPad screen locations. Figure 2.17 shows the detection accuracy of letters, numbers, and shapes in the 3 grids. Since grids 2 and 3 are close to the magnetometer location, we observe that the range for magnetic fluctuations caused by the Pencil is large in these grids, whereas grid 1, which is farther from the magnetometer, is less sensitive to the Pencil's movement. However, we observe that our system performs consistently well across all grids despite detecting only small fluctuations in the grid 1. This observation clearly demonstrates that small changes in magnetic reading can be tracked by S^3.

(6) Does the performance change across users?

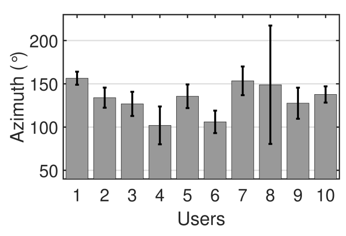
We show in Figure 2.18a how well S^3 performs across different users. The detection accuracy is equal to or greater than 90% for letters, numbers, and shapes for all users. From the touch API data recorded for ground truth, we observe that the range for altitude and azimuth angles vary broadly across 10 users for each category. Figure 2.18b and Figure 2.18c show the average altitude and azimuth angles along with their standard deviation for each of the 10 users. Though we defined a fixed range for these two angles, the consistent good



(a)

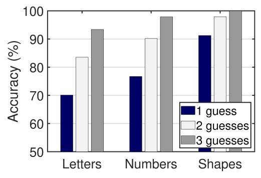


(b)

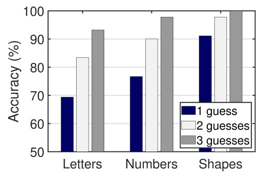


(c)

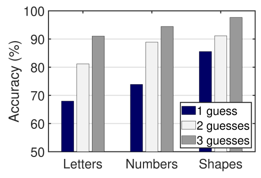
Figure 2.18. (a) Detection accuracy of letters, shapes, and numbers for different users. (b) Distribution of the altitude and (c) azimuth angles for the strokes written on the screen.



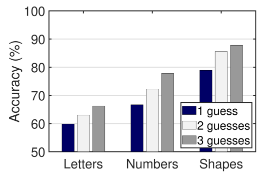
(a)



(b)



(c)



(d)

Figure 2.19. Accuracy of the correctly guessed letters, numbers and shapes in 1, 2 and 3 guesses when the user is holding the iPad in hand (a) while sitting, (b) while standing, (c) while laying down, and (d) while walking.

accuracy for all users shows that our tracking algorithm can accommodate a broad range of Pencil orientations and is not affected by the way a user holds the Pencil while writing.

(7) How does the positioning of the iPad affect S^3's performance?

We evaluated S^3's performance with a variety of iPad positions. Apart from the experiments with the iPad placed on the table, we conducted experiments with three volunteers in the following scenarios: 1) iPad is held in the hand while sitting, 2) iPad is held in the hand while standing, 3) iPad is held in the hand while laying down, and 4) iPad is held in the hand while walking. In each scenario, we asked the volunteers to write letters, numbers, and shapes on the iPad screen, as described earlier. Figure 2.19 shows the detection accuracy for letters, numbers and shapes in the top three guesses for each scenario. While a user is holding the iPad in hand, the writing causes small movements in the iPad body that introduce minor fluctuations in the motion sensors' data. However, a detection accuracy of higher than 90% for all strokes shows that S^3 is resilient to these small fluctuations and can still identify the

Table 2.1. S^3's detection accuracy in different environmental settings.

Scenario	Letters (%)	Numbers (%)	Shapes (%)
Door opening and closing	93.37	95.55	96.67
People walking around in the room	93.80	96.58	97.86
Watch with metallic bracelet	91.88	95.55	96.67
Smart folio iPad case	91.67	94.44	96.67

Pencil's magnetic impact in scenarios 1, 2, and 3. For the scenario when the user is writing on the iPad while walking, we observe that S^3's detection accuracy reduces (66.2%, 77.8% and 87.8% for letters, numbers and shapes respectively) because the continuous change in ambient magnetic readings makes it difficult to separate the magnetic impact of the Pencil and the ambient environment.

(8) How does the environment impact the performance of S^3?

In our experiments, we observed that the magnetic field sensed by the iPad's magnetometer is not affected by regular objects such as books, clothes, etc., in the surroundings. The experiments described above were all conducted in a lab setting where electrical devices like computers, laptops, monitors, etc., were present in the surroundings. Based on these results, we concluded that the existence of these devices did not impact the performance of our system. Similarly, events such as opening or closing the door, people walking around, and the movement of furniture in the environment had no significant effect on S^3's performance.

We also evaluated how S^3's performance is affected by the presence of magnetic objects in the environment. We invited three volunteers to perform experiments in the following scenarios: 1) while wearing a watch with a magnetic bracelet on their non-dominant wrist, and 2) while writing on an iPad with a smart folio case. In both scenarios, the subjects were instructed to hold the iPad in hand. Table 2.1 shows our system's detection accuracy in the top 3 guesses for each scenario described above. In scenario 1, even though the magnetic bracelet is very close to the Pencil while the user is writing, its impact on the magnetic readings is much smaller than the Pencil's magnetic impact. As a result, despite the interference, S^3 is able to accurately track the Pencil position. In the case of the smart folio iPad cover, although the magnets in the cover affect the magnetic readings sensed by the iPad, their magnetic impact stays consistent over time, allowing S^3 to separate the impact of the Pencil as it moves over time.

(9) How does S^3 perform compared to other machine learning techniques?

To conduct a comparative evaluation with our system, we implemented three machine learning methods to infer the users' writing from the motion sensors' data. These methods included 1) k-nearest neighbors (k-NN) regression model, 2) Long Short Term Memory networks (LSTM) model for classification, and 3) an ensemble of LSTM models for classification trained for different locations of the iPad screen. We selected these algorithms as they capture the order dependence in our dataset and consider the nearest neighbors or previous states while guiding us about the pencil location over time (we detail the models in the Appendix.) The tracking results generated from the k-NN model allowed the attacker to guess only 11.54%, 18.67%, and 3% of the letters, numbers and shapes, respectively. The low accuracy is due to the fact that the magnetic readings can be similar for different Pencil locations and orientations and the k-NN model fails to capture the relationship between Pencil's previous states and its current location and orientation. Hence, the tracking results generated from this model are hardly legible. The LSTM model similarly achieved an accuracy of 7.69%, 10%, and 21.67% for letters, numbers, and shapes. This model fails to precisely infer useful information about the users' handwriting due to the variations in magnetic readings caused by changes in the Pencil location and orientation, even when a user is writing the same character (as illustrated in Figure 2.3), To guide the LSTM about the Pencil location, we also trained 3 separate LSTM models on data collected from each of the 3 grids in the input region. We observe that this approach slightly improves the LSTM model's accuracy and yields 11.54%, 16.67%, and 29.67% for letters, numbers, and shapes, respectively. However, this model also yields less accurate results than S^3's particle filter algorithm despite having a smaller search space for the Pencil's location. This is because this model fails to capture the combined impact of the changes in Pencil's location and orientation on the magnetic readings from the given data. In contrast to these models, S^3 can accurately track the Pencil's movement over time solely through the magnetic readings without requiring any large training dataset.

2.7 Limitations and Discussion

In this section, we discuss limitations and opportunities for the improvement of our system.

Limitations: S^3 currently can detect individual letters, numbers, and words. As we have mentioned in the evaluation, S^3 can infer sentences with a good accuracy when the tracking results of its words are analyzed together. Although this approach allows an adversary to infer commonly used English sentences through their semantic content, it might not be feasible for complex sentences when the semantic connections among words are lost or difficult to infer. A natural extension to our work would be to incorporate language models [36] for improving the detection accuracy of complex words and sentences. Since language models capture long-term dependencies, hierarchical relations and sentiment of the language, a sentence drawn from the modelled language, regardless of its complexity, can be inferred from an initial imprecise guess with a high accuracy.

We also demonstrated that S^3's performance is resilient to subtle changes in the environment. However, continuous changes in the ambient magnetic field, such as when a user travels in a vehicle, might interfere with Pencil's magnetic impact, making it difficult for S^3 to infer the users' writing.

In S^3's stroke detection process, more complex models through Recurrent neural networks (RNN) and Long-short term memory (LSTM) can be learned to capture the underlying sequence patterns in sensor data for detecting the beginning and ending timestamps of each stroke more accurately. Furthermore, our attack has a human in the loop for guessing what the victim is writing. Future work will expand our analysis to support models built on computer vision techniques to recognize letters, and natural language processing techniques (NLP) to infer words and sentences to infer user writings without human interaction.

Defense Techniques: We now discuss possible defenses for the side-channel attack presented in this chapter. First, we observed in our preliminary experiments that if the sampling rate for accelerometer and gyroscope data is decreased, say to 50Hz, detecting the beginning and end of the strokes becomes very difficult. Similarly, if the magnetic data is sampled at a frequency of 5Hz, the tracking results become less tangible. Hence, a potential defense against our attack would require iOS to reduce the available sampling rate for accelerometer, gyroscope, and magnetic data when a user interacts with the iPad using Apple Pencil. Another potential defense would be to pause the motion sensors when a user interacts with the Pencil. However, both solutions heavily affect the operation of legitimate applications running in the background, which use motion sensors for activity, context, and gesture recognition. Another possible defense is to apply magnetic shielding [37] to the Apple Pencil. However, this will greatly impact the weight and hence the usability of the Pencil. Future work will analyze the trade-offs between these defense techniques and applications' access patterns to motion sensors.

2.8 Related Work

Side-channel attacks through motion sensors: Several recent works have demonstrated potential privacy leaks from mobile sensors. Wang et al. explored how motion sensors' data collected from smartwatches are used to infer what a user is typing on a keyboard [3]. Another work demonstrated that the changes in the accelerometer readings are powerful enough to help extract passwords typed on smartphone touchscreens [6]. Michalevsky et al. showed the MEMS gyroscopes in modern smartphones could sense acoustic signals, which can be used to identify and parse speech [38]. A recent work unveiled a side-channel attack leveraging smartphone accelerometers to eavesdrop on the smartphone speaker and reconstruct the audio signals [39]. Das et al. [40] combined multiple motion sensors and used inaudible audio stimulation to fingerprint different users by measuring anomalies in the signals. Another line of work leveraged magnetometers to infer private information. It is shown that the magnetometer of a smartphone placed next to the hard drive of a computer allows an attacker to infer patterns about the system details, such as the type of operating system and applications used [41]. Block et al. leveraged magnetometers' ability to detect locations of users' devices within commercial GPS accuracy [42]. Researchers also demonstrated that electromagnetic field measured by smartphone magnetometers could be exploited for webpage [8], and device [43] fingerprinting. Compared with existing side channels, we introduce a new side-channel attack to identify users' handwriting by analyzing the magnetic impact of the embedded magnets in modern stylus pencils sensed by device magnetometers.

Handwriting tracking through motion sensors: There have also been efforts that explored the use of motion sensors for eavesdropping on users' handwriting. Motion sensor readings collected from a smartwatch are used to infer what the user is writing [44, 45]. However, these approaches require a compromised smartwatch to be worn on the victim's writing hand. In contrast, S^3 does not require any extra device, and tracks the Pencil movement with high accuracy without restricting the victim's handwriting style. Another work introduced *Finexus* to track fingertip movements in 3D space by instrumenting the fingertips with electromagnets and measuring the corresponding magnetic field changes using four magnetometers [46]. Although *Finexus* tracks the fingertip movements with millimeter level accuracy, it requires multiple magnetometers for precision. Our system tracks the pencil tip and achieves high accuracy in inferring handwriting with a single magnetometer. Similarly, *TMotion* presented a self-contained 3D input device that enables interactions in 3D space around smartphones by embedding a permanent magnet and an inertial measurement unit (IMU) in the stylus pen [47]. While this work is effective at tracking the stylus, it requires attaching an extra wand on top of the stylus pencil to accommodate a magnet and an extra IMU sensor, impacting the stylus pen's overall usability. In contrast, our system requires no such hardware for accurate pencil tracking. Lastly, a recent work introduced a sensing system for eavesdropping on handwriting by analyzing the magnetic field changes produced by stylus pens [48]. However, a commodity smartphone with a magnetometer must be placed within $20cm$ of the victim's device to sense the magnetic field. In contrast, S^3 uses the magnetometer on the victim's device, resulting in a higher detection accuracy, and does not require the attacker to be present in close proximity of the victim.

2.9 Details of the ML Models

K-Nearest Neighbors (k-NN) regression: We trained a k-NN model on the data collected for generating the magnetic map for the Pencil and evaluated it on the data collected from the subjects in our user study. The model was designed to predict the Pencil's location and orientation at time t when given the magnetic readings for the previous three timestamps. We used a grid search to find the best value for k within the range of 1 to 5 on our training data and chose the value which resulted in the highest accuracy. The attacker was shown the tracking results generated from predicted Pencil location trace for each stroke.

Long Short-term Memory Network (LSTM): We implemented a LSTM model for predicting which letter, number or shape corresponds to magnetic readings collected for a stroke. We used TensorFlow's Keras library [49] for implementing the LSTM network. The model required a three-dimensional input of the shape [samples, time steps, features] where samples is the number of strokes, time steps is the number of magnetic samples for each stroke (fixed to 200 samples) and the number of features is 3 representing the x, y and z axis of the magnetometer readings. The output of the LSTM model was a 26 element vector (10 and 5 in the case of numbers and shapes respectively) representing the probability of a given stroke being any of the 26 letters (10 numbers or 5 shapes). We defined the model as sequential model with two LSTM hidden layers followed by a dropout layer to reduce over-fitting on training data. The features extracted from the LSTM layers were fed into a dense fully connected layer followed by a final output layer used to make predictions. We trained this model on the data collected from the 10 subjects in our user study using a leave-one-out cross validation approach [50].

In an attempt to improve the LSTM network's performance, we limited the search space for the Pencil's location by implementing separate models for each of the three grids in the input region of the screen. The same network architecture was used for the three models. For final inference, we used the output of the model with the highest confidence score.

Ground truth for examples in Figure 2.13 in order: Letters are y, n, c, e, l, z, w, u, a, b, g, q, v, m, h, k, s, i, p, d, o, x, r, h, t, f, j, q, and f. Numbers are 7, 5, 0, 6, 9, 6, 1, 2, 5, 8, 4, 5, 0, 3, 1, 8, 3, 2, 8, 4, 6, 9, 0, 3, 7, and 6. Shapes are heart, star, circle, heart, square, star, circle, square, heart, heart, triangle, star, and square.

Ground truth for examples in Figure 2.16 in order: make, have, now, hello, time, own.

3. LOCIN: INFERRING SEMANTIC LOCATION FROM SPATIAL MAPS IN MIXED REALITY

3.1 Introduction

Mobile mixed reality (MR)[1] has become increasingly popular over the last decade with the release of dedicated headsets and apps that blend virtual content into users' real-world environments. Apart from gaming and entertainment, mobile MR applications have recently found utility in enabling interactive healthcare, education, and e-commerce experiences [51–53]. For instance, e-commerce apps like IKEA place [54] allow customers to experience how a product fits in their environment before purchasing.

MR apps require an elaborate description of the user's surroundings in three dimensions (3D) to localize the MR device and enable realistic assimilation of virtual content in the user's environment. To capture the user's 3D environment, common MR devices, including dedicated headsets and even off-the-shelf smartphones, are equipped with specialized sensors such as depth cameras and LiDAR sensors. For instance, HoloLens 2 equips a depth camera, and Apple's latest iPhone and iPad Pro employ a LiDAR sensor to capture the real-time depth of the surrounding space and objects [55]. These sensors capture the distance between the device and physical points in the environment to generate a *3D spatial map* of the environment.

As mobile MR adoption grows [56], there is increasing concern about the security implications of 3D spatial maps accessed by mobile apps [57, 58]. All MR apps require explicit user permission for camera access to deliver their functionality, i.e., integrate virtual content in the user's environment. Once camera permission is granted, MR apps on popular MR platforms [59–61] have access to the 3D spatial maps. MR apps' access to 3D spatial maps opens doors to a new type of reconnaissance attack where an adversary-controlled malicious app exploits the 3D spatial map of the user's environment to infer user's indoor locations (i.e., *semantic location*).

An adversary's ability to locate users enables them to launch physical attacks and case a target's environment for burglary and assault. With mobile MR use cases in entertainment, education, and retail, such threat broadly applies to our homes, businesses, educational

[1]↑We use MR as an umbrella term for augmented and virtual reality.

facilities, and many others. Moreover, an adversary can combine the user's indoor location information along with the object and semantic properties of the user's environment to build a profile for delivering personalized ads and recommendations. An adversary can also exploit this to understand users socio-economic status, accessibility requirements, and product preferences, as well as their identity, routines, and activities.

In this chapter, we study how an adversary can exploit 3D spatial maps from MR devices to infer a user's indoor location. Prior work has explored indoor location inference from 2D and RGB-D images [62–67]. However, the direct application of existing approaches to spatial maps is impractical because location inference from spatial maps requires a different type of feature extraction and optimization process due to its sparse, non-uniform, and dynamic nature. To achieve this, a recent work [68] built a location classifier from spatial maps. This approach, however, suffers from two main limitations. First, it only uses the high-level geometric features of spatial maps without leveraging their semantic context. The lack of such semantics leads to poor accuracy in inferring different indoor environments. Second, it compares a given location of a user with a labeled database of that user's previously visited locations with the goal of finding whether the user has been in that location before. From an attack perspective, it fails to infer the location of users without knowing a priori the spatial maps of their indoor environments. These limit its attack practicability, ultimately making it infeasible to conduct a location inference attack in practice.

To this end, we present LocIn, a location inference attack on mobile MR devices which exploits 3D spatial maps to infer a user's location. We observe that indoor environments are uniquely characterized by their semantic context (e.g., objects and surfaces in the environment). Yet, unlike pixel arrays in images, a 3D spatial map is a set of unordered points with non-uniform density, making detecting objects and surfaces in the environment challenging. Therefore, we introduce a new location inference learning representation that composites the geometric and contextual patterns embedded in the spatial map to infer a user's location. We design a multi-task learning approach and build an end-to-end encoder-decoder network that can successfully infer the user's location from spatial maps captured by various MR devices.

LocIn first preprocesses the 3D points in a 3D spatial map through farthest point sampling [69] to remove outlier points and subsample the map to a fixed number of points. The preprocessed map is then fed as input to LocIn's spatial encoder, which extends a hierarchical neural network [70] to extract a spatial feature representation of the map. This feature representation embeds geometric and contextual patterns of the user's environment. This representation is invariant to dynamic changes (e.g., changes in viewing angle or size of the map) in spatial maps as users interact with the MR apps. Lastly, LocIn's encoder output is fed to LocIn's multi-task location decoder. The multi-task decoder performs 3D object detection and semantic segmentation to generate 3D bounding boxes of objects and point-wise labels for objects and surfaces in the environment. It then integrates the object and semantic context into a classification network to predict the location type (e.g., bedroom, office) of the input spatial map.

We present three studies to evaluate LocIn's effectiveness. In a first study, we evaluated LocIn on a dataset [71] consisting of ~$1,500$ spatial maps collected via an iPad Air 2 equipped with a depth sensor belonging to 13 unique indoor location types where MR devices are typically used. To demonstrate LocIn's effectiveness on MR devices with different depth sensing techniques, in a second study, we evaluated LocIn on a dataset [72] collected through an iPad Pro equipped with a LiDAR scanner consisting of ~5,000 spatial maps. The spatial maps belonged to 9 unique indoor location types in real-world homes. Finally, to show LocIn attack's practicality on dedicated headsets, we evaluated LocIn on our dataset of spatial maps collected via HoloLens 2. LocIn correctly predicted the location types from the spatial maps with an average accuracy of 84.1%. We also show the LocIn attack is robust against varying sparsity (number of points) and size of the 3D spatial maps.

In summary, we make the following contributions:

- We present a location inference attack on mobile mixed reality devices that exploits the 3D spatial maps captured from users' environments to predict their location type.
- We design LocIn, a multi-task learning framework that leverages an encoder-decoder architecture to infer the user's location by integrating the geometric and contextual patterns embedded in the spatial maps.

- We evaluate LocIn on 3D spatial maps captured using three MR devices from 13 unique location types. We demonstrate that LocIn can infer a user's location with an average accuracy of 84.1%, and it is robust against varying sizes and sparsity of the spatial maps.

3.2 Background

Mixed Reality. The influx of reality-altering headsets and applications has brought mixed reality (MR) to the spotlight in recent years. Consequently, mobile industry has been striving to enable comfortable and realistic MR experiences for users. For instance, Google and Apple released ARCore [60] and ARKit [61] to enable MR app development for smartphones and tablets. Moreover, dedicated standalone headsets, such as HoloLens, have recently emerged with their own MR development platforms, e.g., Windows Mixed Reality Toolkit [59].

MR devices integrate virtual reality (VR) and augmented reality (AR) to allow users to visualize and interact with both real and virtual content in their own physical environment. MR use-cases range from social media apps (e.g., Snapchat filters [73]) and games (e.g., Pokemon Go [74]) to e-commerce (e.g., IKEAPlace [54]) and educational apps (e.g., chemical molecular structures [75], human anatomy [76]).

To enable mixed reality experiences, MR devices embed multiple sensors, such as RGB cameras and microphones that capture input from the user's surroundings. These devices are often equipped with an inertial measurement unit (IMU) to enable tracking of users' head and body movements. Some recent MR devices are also equipped with specialized depth sensors. For instance, HoloLens 2 uses a depth camera while the latest iPhones and iPads employ a LiDAR sensor to build an understanding of their surroundings [55, 77].

3D Spatial Maps. MR devices need to accurately locate themselves in relation to the physical world and understand the objects and surfaces in the environment to seamlessly superimpose digital content in the user's surroundings. Most MR devices rely on camera and sensing-based localization techniques to locate themselves in the user's environment [78]. These techniques require access to a detailed 3D digital representation of the user's environment, which is used by MR apps to overlay virtual content in the user's surroundings. To this end,

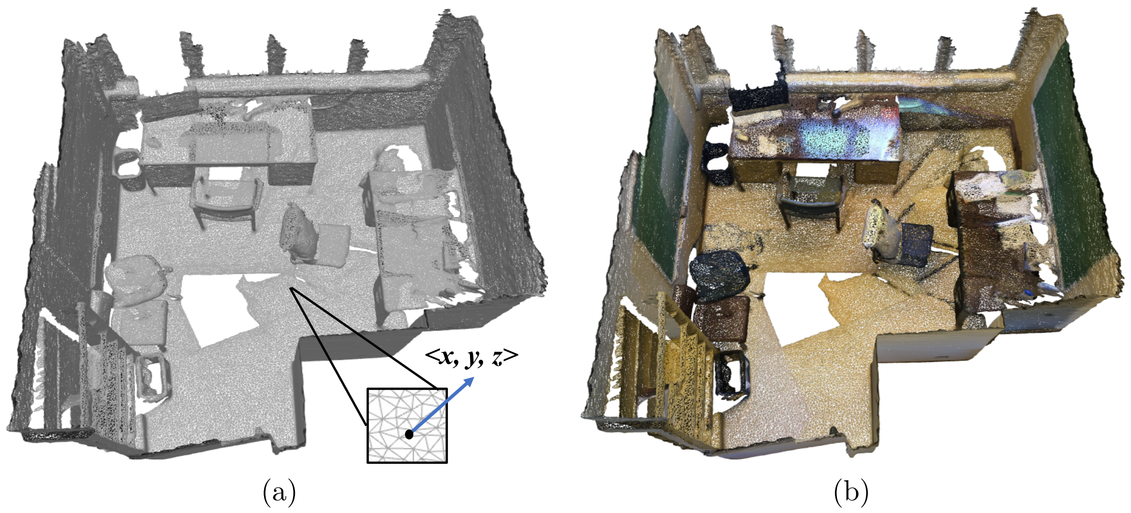


Figure 3.1. An example of a spatial map captured with an MR device in an office (a) without and (b) with color.

commonly available MR devices - dedicated headsets and even off-the-shelf smartphones - are equipped with depth sensors that measure the distance between the device and points in the real environment. These depth sensors include Time-of-Flight (ToF) cameras (e.g., HoloLens), and LiDAR scanners (e.g., on iPhone 13 and iPad Pro).

The 3D mapping of the user's environment is shared with MR apps to allow virtual or augmented content to interact with the physical world, e.g., anchoring a virtual object on user's desk. MR devices provide the 3D representation of the users' environment to MR apps as a *3D spatial map*. The 3D spatial map is represented by a set of 3D points $\{P = p_1, \ldots, p_n\}$, where each point p_i is a vector of its (x, y, z) coordinate in space. Figure 3.1 shows an example of a 3D spatial map captured by iPad Pro, with and without the color information. Some MR devices also include color and normal vectors (e.g., HoloLens [79]) representing orientation for each point in the 3D spatial map.

3.3 Problem Statement and Threat Model

3.3.1 Motivation

MR Permission Models. The MR devices leverage the same permission models as traditional mobile operating systems e.g., iOS and Android, to control apps' access to sensitive data [80–82]. Since access to the camera is essential for MR apps to visually integrate virtual content in the user's environment, users must grant camera permission to allow MR apps to function as intended. Prior works have highlighted that sensitive data such as credit cards details, sensitive documents, or bystanders' facial identities could be revealed from images and videos, captured by MR devices [83–86]. Consequently, several works have attempted to protect visual privacy by only sharing user-defined privacy-preserving visual features with apps [83, 85, 87], augmenting privacy markers (e.g., QR codes, RFID tags) into the real world to avoid sensitive content [84, 88], and defining fine-grained permissions for app's access to visual data [89, 90].

Significant efforts have been made to restrict MR apps from accessing users' private data through images and videos [86]. Yet, these apps must access the 3D spatial maps of the user's environment to deliver MR content. Hence, once camera permission is granted to an app, it can access the spatial map.

Security and Privacy Implications of Spatial Data. Access to 3D spatial maps of the user's environment poses serious privacy threats as these maps capture privacy-sensitive cues about the user's surroundings. For instance, a spatial map captures detailed characteristics of the environment, such as its geometric properties (e.g., length and width of the room), and embeds semantic information about the types of objects or surfaces present in it. Similar to how humans perceive an indoor environment based on its layout, objects, and surface properties [91], an adversary can extract these characteristics from the 3D spatial map to infer user's location, i.e., the type of indoor environment.

Unlike images and videos, 3D spatial maps are not easily interpretable by average MR users and, therefore, are not perceived as sensitive data yet, exacerbating their privacy threat. Moreover, spatial maps, unlike images, are not sensitive to lighting conditions, occlusions, or camera orientations/viewpoints which allows an adversary to infer private information about the user in a variety of scenarios.

An adversary can exploit spatial maps to infer a user's location (e.g., user is at their office or home), without explicitly requesting location permission from the user. The adversary could exploit this information to launch a physical attack (e.g., robbery or assault). The adversary could also gain a fine-grained understanding of the user's routine (e.g., when the user wakes up, goes to work) based on the inferred locations across time. This information can be leveraged by data brokers aiming at selling detailed user profiles to third parties.

An adversary can also combine the object and semantic segmentation information embedded in a spatial map with the inferred location to reveal additional private information about the user. First, an adversary could leverage the detected objects and the inferred location to understand users socioeconomic status, accessibility requirements, and product preferences. For instance, a wheelchair in a bedroom or handles near a toilet might indicate a user's accessibility needs. Similarly, the type of appliances found in a user's environment could reveal their income level. Second, an adversary could use information about rooms and objects sizes and timing of a collected map to distinguish locations visited by a user (e.g., bed on day-1 vs. day-2) and infer users identity, behavior, and activities. For instance, the number of beds in a room could reveal the familial structure of a user, and the presence of athletic equipment could indicate a user's hobbies. Lastly, with recent permissions for tracking user activities on mobile OS (e.g., Apple's app-tracking-permission), access to spatial maps aid adversaries in building user profiles for delivering targeted ads based on their visited locations. This is privacy critical as users are unaware of their data being collected.

Therefore, in this chapter, we set the foundation for uncovering users private information inferable from spatial maps.

3.3.2 Problem Statement

We investigate how an adversary can exploit the 3D spatial maps captured by MR devices to infer the indoor location of a user, e.g., a user is in the office or bedroom. We consider the spatial map of a user's environment, $P = \{p_1, p_2, \ldots, p_n\}$, where n is the number of points in

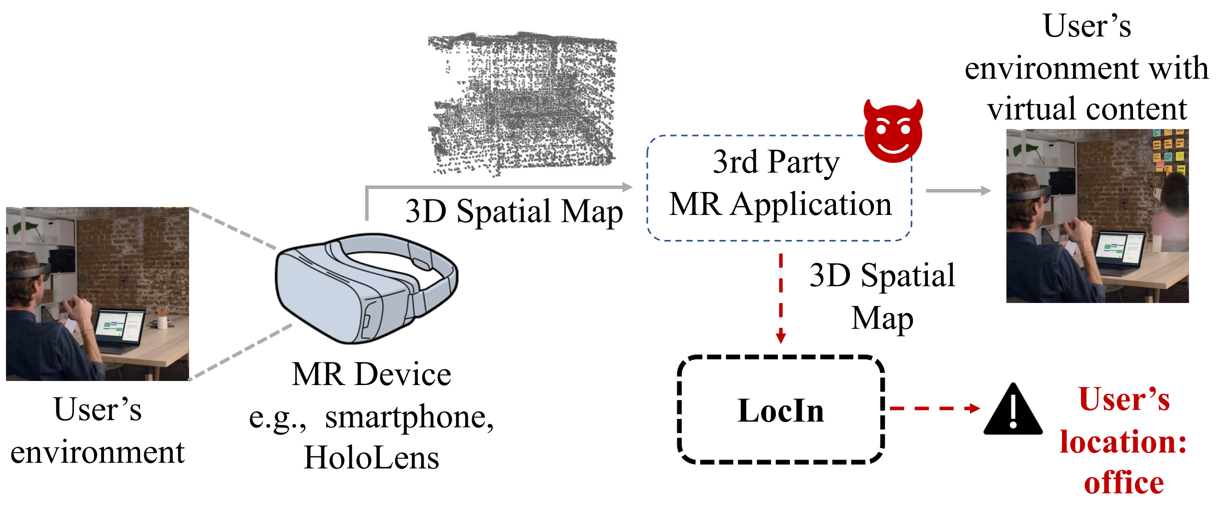


Figure 3.2. A malicious MR application accesses the 3D spatial map of a user's environment to integrate virtual content. The app can exploit this map to infer the user's location.

the map and each p_i consists of the 3D coordinates of a point. Given P, our goal is to infer the indoor location where the spatial map was captured.

To illustrate, consider the scenario in Figure 3.2 where a user installs a malicious virtual meeting app on their mixed reality device (e.g., iPad Pro with LiDAR sensor). The app requests camera permission from the user to display avatars of other meeting attendees. The user interacts with the app in their office to conduct a meeting with their colleagues. The malicious app accesses the 3D spatial map of the user's current surroundings and shares it with a remote server.

A remote adversary can exploit this spatial map to infer that the user is at their office and leverage this information to break into the house when the user is not home. The adversary could also exploit this information to gain a fine-grained understanding of the user's identity, routine, and preferences (e.g., when the user wakes up, goes to work, user's hobbies, etc.). Additionally, based on the user's inferred location, an adversary can create a detailed profile of a user to deliver unsolicited personalized ads to generate ad revenue. For example, a virtual meeting app could display ads for office furniture or work productivity tools while the user is in their office.

3.3.3 Threat Model

We consider an adversary with the goal of inferring the type of user's indoor location (e.g., bedroom, kitchen, office) while a user is using an MR device. The target device can be any MR device, capturing a 3D spatial map via the device camera equipped with a depth sensor or LiDAR scanner, e.g., a smartphone with MR capabilities and HoloLens. We assume the adversary has no prior knowledge about the target user's indoor environment, including its physical location, spatial maps from prior app usage, and location type. The adversary trains the attack model using publicly available datasets that include typical indoor location types [71, 72]. We discuss in Section 5.6 how the adversary can infer location types not observed during the training process.

An adversary provides the user with an MR app, which accesses the 3D spatial map of the user's surroundings while the user interacts with the app. The adversary can achieve this by (1) distributing the MR app on MR platforms' app stores and third-party MR forums, and (2) deceiving users into installing the MR app via phishing and other social engineering methods. We note that 3D spatial map can be accessed if the camera permission is granted to an app. As MR apps require camera access to deliver their basic functionality, camera permission is given to all MR apps. The app does not require any other permissions, such as permissions for the device location, or the images and videos recorded by the device; additionally, the 3D spatial map accessed by the app does not contain any color or normal vector information. We note that given the adversary deploys a malicious app on the target user's device, it has knowledge of the user's MR device model and its depth-sensing technology. Based on this, the adversary trains an attack model on publicly available datasets collected using similar depth sensors as the target device.

3.3.4 Design Challenges

C1: Extracting Location Cues from Spatial Maps. Given a 3D spatial map, intuitively, an adversary could train a classifier that extracts high-level features, such as structural and geometric properties (e.g., length and width or floor map) of the user's environment to infer the location. However, this is a challenging task as indoor environments of the same type vary significantly in their structural and geometric properties. For instance, the size of two bedrooms in a user's house may differ. Similarly, the geometric features of different locations may share similarities (e.g., structure of a classroom may share similarities with that of a conference room).

One potential solution to this problem is leveraging the semantic context of indoor locations to discriminate them better. To achieve this, an adversary could train a 3D spatial map model trained either on objects or 3D surfaces present in the environment. Yet, solely using objects or 3D surfaces yields incorrect location inference (demonstrated in Section 3.7) because, unlike image pixel arrays, a 3D spatial map is an unordered collection of points with a non-uniform point density, which makes detecting objects and 3D surfaces challenging. Therefore, to accurately infer the location, we build a feature extraction and loss minimization pipeline that simultaneously captures both geometric properties and the semantic context of the user's environment.

C2: Invariance to App Usage. Spatial maps captured by MR devices change dynamically in size and viewing angle when a user interacts with different MR apps. For example, while playing an MR game, a user may walk around the room, and while using a shopping MR app, a user may position products to specific locations in a real-world perspective. Hence, to infer the user's location from the captured spatial map, the location inference model must be invariant to spatial map transformations. For instance, translating or rotating the spatial map points should not change the model's location prediction. We leverage 3D point subsampling and hierarchical multi-scale spatial feature learning to overcome this challenge.

C3: Lack of Prior Knowledge and Generalization. To launch location inference attacks in practice, an adversary must be able to infer the target users' location without making any assumptions or prior information about their environments. One naive approach would be collecting spatial maps from a set of users while they are using an MR app and then manually-label them with a set of location labels. However, while this approach may yield an acceptable attack success rate for seen users, it is tedious and time-consuming and may fail to generalize to unseen users due to the bias to specific locations, and produce incorrect results (Section 3.7). To address this, we use public spatial map datasets with diverse location labels

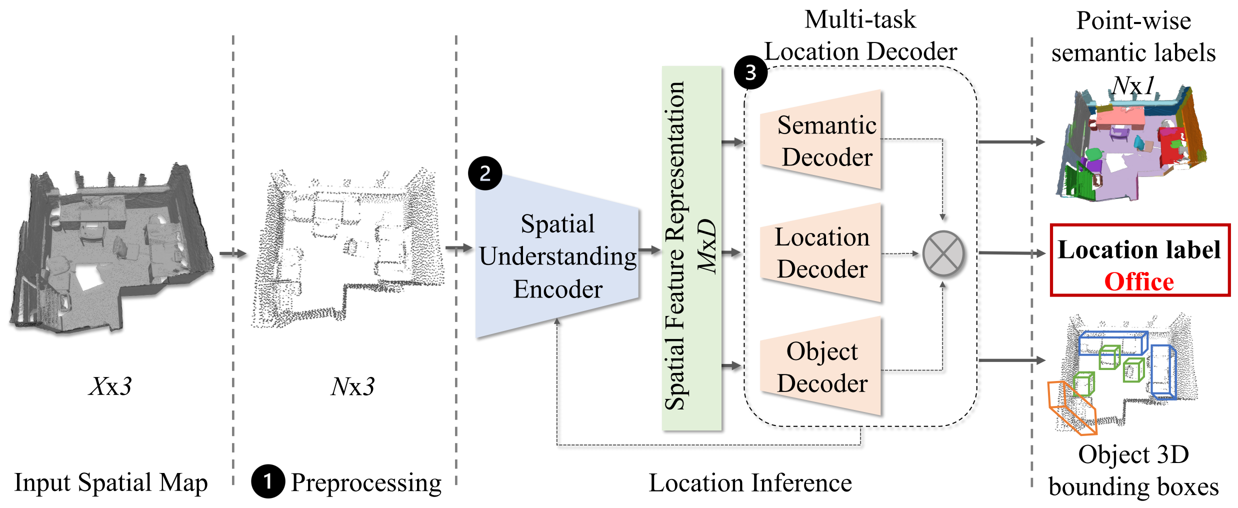


Figure 3.3. Overview of LocIn attack.

(in addition to the dataset we collected). Additionally, to eliminate the sparse, non-uniform, and noisy nature of such datasets, we leverage a data-driven upsampling method to generate dense points while improving proximity-to-surface and distribution uniformity in the 3D point representations.

3.4 LocIn Attack Overview

We present LocIn, a location inference attack for mobile mixed reality devices, which exploits 3D spatial maps to identify a user's indoor location. Figure 3.3 illustrates the overview of LocIn attack. Given that the sparsity of spatial maps varies based on the MR device and device usage duration, we first preprocess the input map by removing outlier points and subsampling the 3D points in the spatial map (❶). Here, we leverage farthest point sampling [69], resulting in a reduced size map with N points (addressing **C2, C3**).

As discussed in Section 3.3.4, inferring a user's location solely from geometric (i.e., structure and appearance of an environment) or semantic (i.e., objects and surfaces present in the environment) properties of an indoor environment can often lead to ambiguous results. To address this issue, we introduce a new location inference learning representation by combining the geometric and semantic properties of a 3D spatial map. For this, we use a multi-task learning approach and train an end-to-end encoder-decoder [92], which, in turn, successfully infers the user's location (addressing **C1**).

Specifically, we extract the geometric and semantic properties of the user's environment by extending a PointNet++-based hierarchical neural network encoder [70] (❷). The encoder extracts a spatial feature representation of the input spatial map (Z). Z is then fed to LocIn's multi-task location decoder (❸). The multi-task decoder is a composite model consisting of three components: (1) location decoder, (2) object decoder, and (3) semantic decoder. The location decoder consists of a fully-connected neural network classifier trained to predict the type of input spatial map's location. The object decoder extends a 3D object detection network [93] that detects the objects in the user's environment. Lastly, the semantic decoder leverages a segmentation network [70] to provide fine-grained information about the objects and surfaces present in the user's environment.

To ensure that the location decoder integrates the intrinsic patterns from object detection and semantic segmentation, we introduce a unified optimization function that combines the loss functions of the three decoders to train LocIn.

3.5 LocIn Design

3.5.1 Spatial Map Preprocessing

LocIn attack aims to infer the location of the user without any prior knowledge about the user's MR device. As different MR devices use different sensors to generate the 3D spatial map of the user's surroundings, the number of points in the 3D spatial map varies from one device to another. For example, the spatial map's point density captured via a HoloLens 2 with depth cameras is different from the spatial map obtained through the iPad's LiDAR scanner. Moreover, spatial maps' point density is largely dependent on the user's app usage. For instance, if the user moves quickly in their environment while interacting with the app, the spatial map has fewer points.

To infer location from spatial maps with varying densities, we transform the input spatial map to a fixed number of 3D points. Given an input map (P) of size $X \times 3$, with a 3D

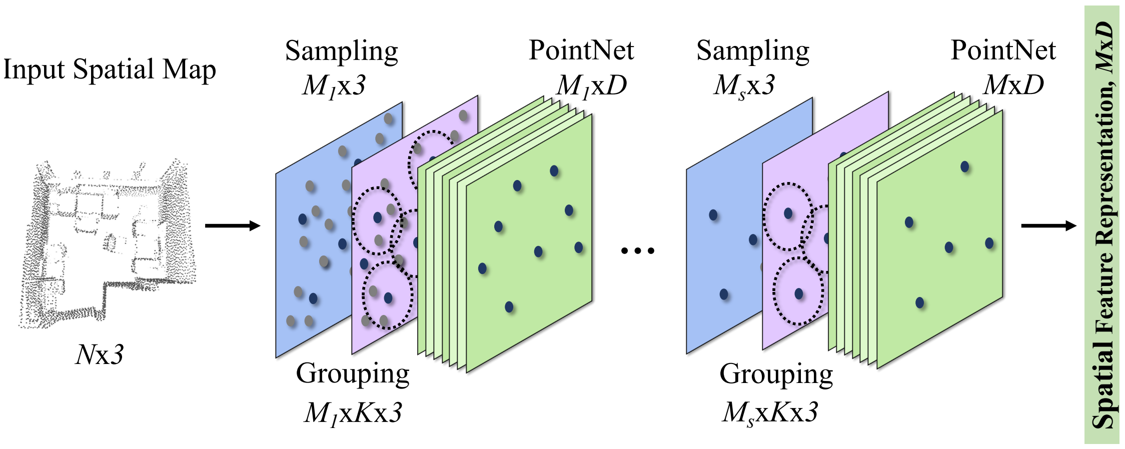


Figure 3.4. LocIn's spatial understanding encoder architecture based on a hierarchical neural network (PointNet++).

coordinate for each X point, we apply farthest point sampling [69] to the map to generate a transformed map of size $N \times 3$. Specifically, we select a random point and then iteratively sample points that are farthest from the selected samples. This ensures that the input spatial map is subsampled to a fixed size for efficient processing while providing sufficient coverage.

3.5.2 Spatial Understanding Encoder

To infer the user's location, a method is needed to understand the geometric properties and semantic context from the preprocessed spatial map. For this, we convert the input spatial map, P, into a high-level spatial feature representation Z that only encodes information relevant for uniquely identifying the user's location. For instance, for the spatial map of a bedroom, Z could represent features of objects or surfaces in the map that can help differentiate its location from others, e.g., 3D points of a bed and nightstand placed close to a wall.

To learn the spatial feature representation, instead of relying on hand-crafted geometric and semantic features, we leverage network architecture for learning point-wise features from 3D point clouds [70, 94, 95]. While LocIn's approach is not limited to a specific network, we extend PointNet++ [70] as an encoder to learn hierarchical features that preserve spatial localities in the spatial map at different contextual scales

Figure 3.4 illustrates the architecture of LocIn's encoder. The encoder's PointNet++-based hierarchical structure consists of multiple set abstraction levels with skip connections. At each abstraction level, a subset of points from the input spatial map is selected to ensure efficient computation and processed into a feature vector that represents the local context of the selected points. The set abstraction consists of three main operations: (1) sampling, (2) grouping and (3) PointNet feature extraction.

Iterative Spatial Sampling. The sampling operation in each set abstraction level leverages iterative farthest-point sampling to select a subset of the spatial map's points such that each point in the subset is the most distant from the remaining points in the subset. Therefore, given an input set of points of size $N \times 3$, the sampling operation generates a set of points of size $M \times 3$. In contrast to random sampling, the farthest point sampling ensures that the sampled points provide better coverage of the entire spatial map. This sampling operation is similar to LocIn's preprocessing step (Section 3.5.1). Yet, it is repeated at each set abstraction level with a decreasing number of samples, followed by the spatial grouping operation.

Spatial Locality Grouping. With the grouping operation, we generate groups of neighboring points for each point selected during the sampling operation. These groups represent local regions of the input spatial map used for feature extraction via PointNet [94]. To identify the neighboring points, we extend ball query algorithm [96] for the grouping operation to find all points within a specified radius of a given point. Ball query algorithm uses a divide-and-conquer approach to build a ball-tree i.e., a binary tree in which each node of the tree represents the set of neighboring points within a specific radius. Starting from the set of 3D points from the sampling layer as its root node, we iteratively build the ball-tree by selecting the farthest point from the centroid of root node points as the left child and the farthest point from this left child as the right child of the root node. The points in the root node are then assigned to the children nodes based on their distance from the node.

The grouping operation transforms the input spatial map of size $N \times 3$ into a set of groups of points of size $M \times K \times 3$. Here, M is the number of groups centered around the points selected via sampling operation, and K is the number of neighboring points found for each selected point. We note that K varies across groups, but the spatial feature vector extraction layer converts variable length groups to a fixed vector size.

Spatial Feature Vector Extraction. For each of the local spatial map regions (groups) extracted via the sampling and grouping operations, we extract a spatial feature vector that encodes the context of the local region through PointNet [94]. We first extract the coordinates of points in each of the M local regions relative to the centroid of the region. The resulting groups of coordinates are then fed as input to PointNet.

Given the set of points for a specific region, PointNet maps the points to a D-dimensional feature vector through a multi-layer perceptron (MLP) network and a max-pooling function Thus, the feature extraction layer of LocIn's encoder returns a spatial feature representation (Z) of size $M \times D$, capturing the local context of various parts of the input spatial map.

3.5.3 Multi-Task Location Decoder

We translate the problem of inferring location from the feature encoding of the spatial map into a classification task. Given the encoding Z of the user's environment obtained from the spatial encoder, LocIn's location decoder predicts the user's indoor location. For this, LocIn first extracts the high-level geometric features of the user's environment to infer its location. It then combines these geometric features with the map's contextual patterns through object and semantic decoders.

We consider pairs of spatial feature encodings and their location label,

$$\{(z_{\mathrm{i}}, y_{\mathrm{i}})\}_{\mathrm{i}=1}^{n} \sim P^{n}(z), z_{\mathrm{i}} \in \mathbb{R}^{D}, y_{\mathrm{i}} \in \Delta^{c} \tag{3.1}$$

where c is the number of location classes, and Δ^c is the set of c-dimensional probability vectors. Given this data, our goal is to learn a location decoder

$$f_t = \underset{f \in \mathcal{F}_t}{\arg\min} \frac{1}{n} \sum_{\mathrm{i}=1}^{n} L_{loc}\left(y_{\mathrm{i}}, \sigma\left(f\left(z_{\mathrm{i}}\right)\right)\right) + \Omega(\|f\|) \tag{3.2}$$

where $\mathcal{F}_t$ is a class of functions from $\mathbb{R}^D$ to $\mathbb{R}^c$, the function $\sigma : \mathbb{R}^c \rightarrow \Delta^c$ is the softmax operation, the function $L_{\text{loc}} : \Delta^c \times \Delta^c \rightarrow \mathbb{R}_+$is the cross-entropy loss

$$L_{\text{loc}}(y, \hat{y}) = -\sum_{k=1}^{c} y_k \log \hat{y}_k \tag{3.3}$$

and $\Omega : \mathbb{R} \rightarrow \mathbb{R}$ is an increasing function as a regularizer.

To infer the location from the spatial encoding, f_t can be implemented as a deep neural network that extracts high-level features of the spatial map useful for predicting its location. However, these features often lack distinctive local or global semantic patterns necessary for uniquely identifying a location (See our evaluation in Section 3.7). Moreover, the lack of labeled data (i.e., pairs of spatial maps and their location label) makes learning a generalized location classification model for various users challenging.

We observe that indoor environments are uniquely characterized by their semantic context i.e., the types of objects and fine-grained details about surfaces present in the environment. For instance, a bedroom can be uniquely identified because of the presence of a bed, and a kitchen can be identified if a stove is detected in the spatial map.

To integrate contextual information into our learning representation for location decoding, we propose a new composite learning representation for location decoding by leveraging the multi-task learning paradigm in transfer learning [92]:

$$f_s = \arg\min_{f \in \mathcal{F}_s} \frac{1}{n} \sum_{\text{i}=1}^{n} [\alpha L_{\text{loc}} + \beta L_{\text{obj}} + \gamma L_{\text{sem}}] \tag{3.4}$$

Here L_{obj} and L_{sem} are the loss functions for detecting objects and extracting semantic patterns from the input spatial encoding (detailed in Sections 3.5.3 and 3.5.3).

Through this learning representation, we learn a model to classify the location of a given spatial map while concurrently learning the contextual patterns from objects and surfaces in the map. Our representation is inspired by the teacher-student networks in knowledge distillation where the learning of a student model is guided by the teacher networks [97]. In our case, the location decoder is the student network, while the object and semantic decoders act as teacher networks. Object and semantic decoders benefit from stronger supervision

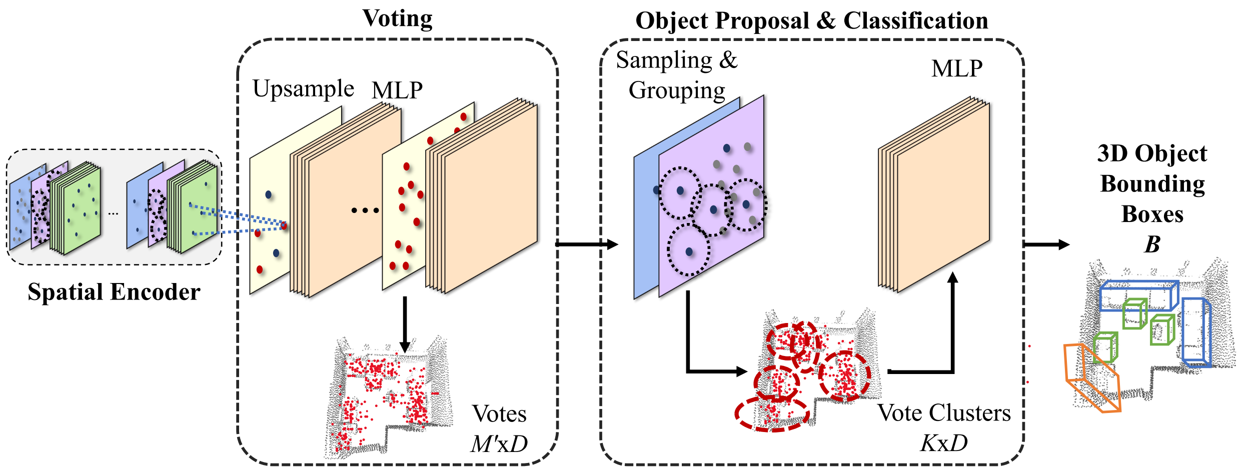


Figure 3.5. LocIn's object decoder architecture with a deep Hough voting, object proposal, and classification module.

during learning as the labeled data for these tasks includes ground truth for multiple objects and point-wise semantic class labels.We show in Section 3.7 that this composite learning representation helps improve LocIn's overall accuracy.

3D Object Decoder

The goal of LocIn's object decoder is to localize and recognize 3D objects present in the user's environment as indoor environments are characterized by the objects present within them. This process involves detecting the orientated 3D bounding box for each object from the spatial map as well as predicting the semantic class of each detected object.

To this end, we extend VoteNet [93], a voting-based network for object detection Figure 3.5 illustrates the architecture of LocIn's object decoder, consisting of two modules: a voting module and an object proposal and classification module.

Object Voting. The voting module adapts Hough transform [98] to generate votes for points in a 3D spatial map based on their distance to objects' centers. Since depth sensors used by MR devices typically only capture object surfaces, 3D object centers may not be close to any point. For accurate bounding box generation around object centers, VoteNet leverages the Hough voting to sample *seed points* which are close to object centers and generate votes based on their features.

We use the spatial encoding Z of size $M \times D$ as the input to the voting module. To ensure significant coverage of spatial map points for object detection, we first perform upsampling on the M points via multiple feature propagation layers and obtain spatial feature representation for M' points in the original map. The feature propagation layer interpolates the point features of the input points to output points by computing the weighted average of their three nearest input points features and concatenates features from LocIn's encoder (forwarded through skip-connections) through an MLP network.

Given the set of upsampled points and their features, $M' \times D$, we generate votes for each point via a shared MLP with fully connected layers. The MLP computes the vote for each point p_i by predicting its offset from an object's center Δp_i through the following regression loss minimization:

$$L_\text{vote} = \frac{1}{M_o} \sum_\text{i} \|\Delta p_\text{i} - \Delta p_\text{i}^*\| \mathbf{1}[p_\text{i} \text{ on object surface}] \tag{3.5}$$

where M_o is the total number of points lying on the object surface, Δp^* is the ground truth displacement of the point p_i from the object's center it belongs to and $\mathbf{1}[p_\text{i} \text{ on object surface}]$ indicates whether the point lies on an object surface.

The resulting votes represent the semantics of different parts of the objects in the environment.

Object Proposal and Classification. The object proposal and classification module groups and aggregates the votes generated by the voting module to generate object bounding box proposals. It first clusters the votes via uniform sampling and grouping according to spatial proximity and then processes them through a series of MLP and max-pool layers to generate the bounding box proposals. This generates a set of bounding boxes, B, for various objects in the input spatial map. These bounding boxes are represented as a multi-dimensional vector, including the center coordinates and size of the bounding box.

Object Detection Loss. The object decoder generates the bounding boxes for the detected objects and assigns the semantic class label to each object. Therefore, we train it by optimizing

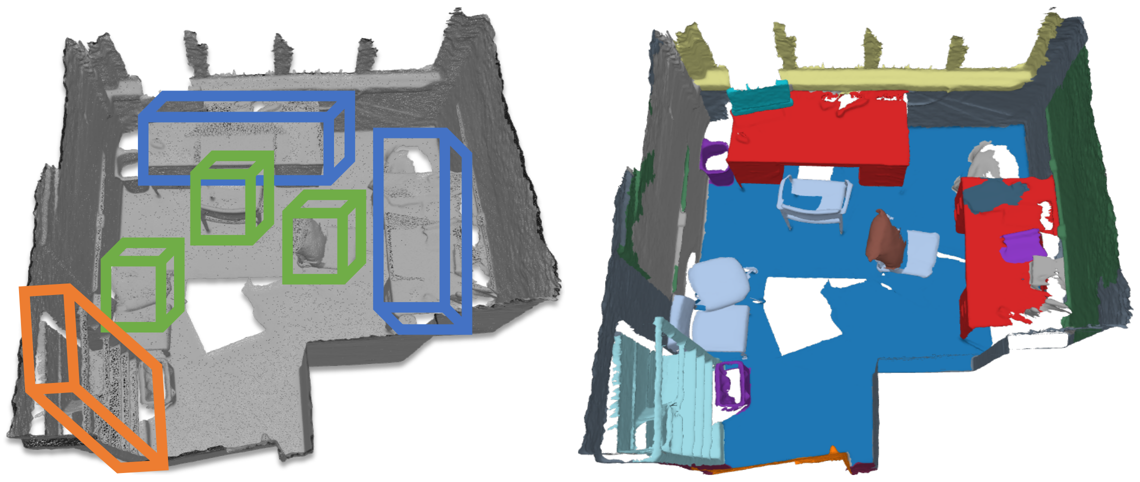

(a) (b)

Figure 3.6. An illustration of (a) object detection and (b) semantic segmentation output. The color in (b) represents the semantic label for points in that region.

a multi-task loss function consisting of the vote (L_{vote}), objectness ($L_{\text{obj-cls}}$), 3D bounding box estimation (L_{box}), and semantic classification ($L_{\text{sem-cls}}$) losses.

$$L_{\text{obj}} = L_{\text{vote}} + \lambda_1 L_{\text{obj-cls}} + \lambda_2 L_{\text{box}} + \lambda_3 L_{\text{sem-cls}} \tag{3.6}$$

Objectness loss gauges whether the detected box proposals indeed belong to an object. For this, we categorize the detected object proposals based on their distance from the ground truth object center into positive ($< 0.3m$) and negative proposals ($> 0.6m$). Objectness loss is then computed via a cross-entropy loss normalized by the total number of proposals.

The box loss optimizes the detected parameters of the 3D bounding boxes, i.e., their centers (x, y, z coordinates), size (height, width, and length), and the heading angle along Z-axis. Thus, the loss is defined as a combined regression loss of the detected bounding box's center, heading angle and size.

The semantic classification predicts the object labels through standard cross-entropy loss ($L_{\text{sem-cls}}$).

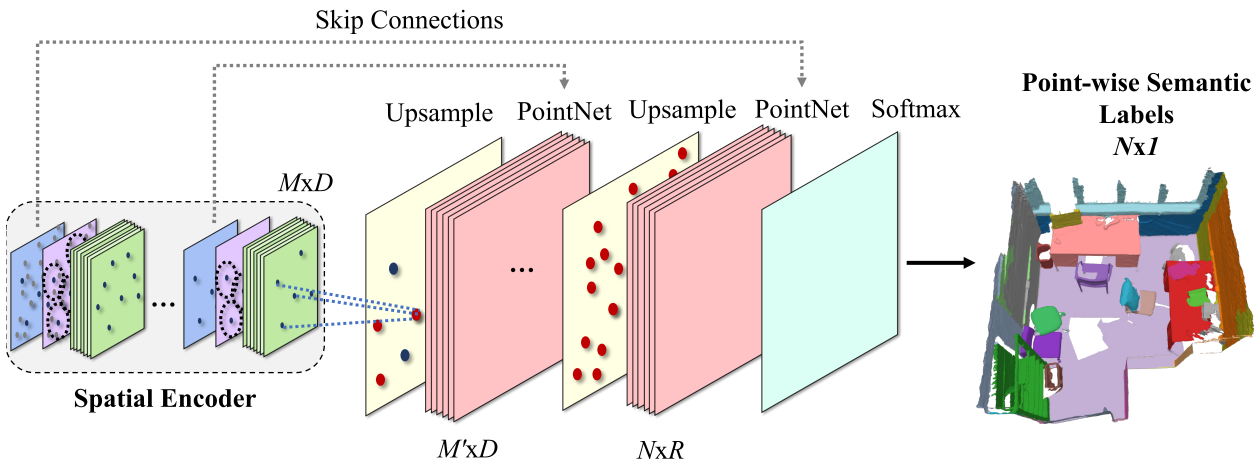


Figure 3.7. LocIn's semantic decoder consists of upsampling and PointNet layers that generate point-wise semantic labels.

3D Semantic Decoder

The objects in a given location type help distinguish it from other locations. However, the object decoder is sensitive to the sparsity of the input spatial map. The object decoder also does not consider planar surfaces, e.g., walls, floor, or ceiling of a room, or fails to detect smaller objects (See Figure 3.6a for an illustration of the object decoder's output).

To extract fine-grained contextual patterns from the user's environment, LocIn leverages a semantic decoder that performs semantic segmentation for classifying each point in the input map to its semantic object class. For instance, all points belonging to walls are assigned the same label (as shown in Figure 3.6b). The semantic decoder's output is a set of homogeneous subsets of 3D points, where each subset represents a semantically meaningful object or surface.

Figure 3.7 shows the architecture of LocIn's semantic decoder. LocIn first performs feature upsampling on the spatial encoding, Z, of the input spatial map. This process ensures that the semantic labels are generated for all points in the spatial map. We employ a hierarchical feature propagation strategy, inspired by PointNet++ [70]. The spatial feature representation from LocIn's encoder of size, $M \times D$, is propagated through a series of upsampling layers where each upsampling layer interpolates feature values, f, for the points in set abstraction level, l, of the encoder, at the coordinates of points in set abstraction level, $l + 1$.

We perform the interpolation through the inverse distance weighted average [99] based on the k-nearest neighbors for each point. Specifically, for a given point p in layer $l+1$, the interpolated features are computed as:

$$f_p = \frac{\Sigma_{\mathrm{i}=1}^{k} w(p).f_{p_\mathrm{i}}}{\Sigma_{\mathrm{i}=1}^{k} w(p)}, \text{ where } w(p) = \frac{1}{d(p, p_\mathrm{i})} \tag{3.7}$$

The interpolated features at each layer, f_p, are then processed through a PointNet layer consisting of convolution, shared fully connected and rectified linear unit (ReLU) activation layers to obtain semantically meaningful features for each point. The final interpolation layer generates R-dimensional semantic features for all N points in the input spatial map. Lastly, we apply a softmax function to the semantic features to obtain the per-point semantic labels S for the input map.

Semantic Segmentation Loss. The semantic segmentation decoder is inherently a classification network that assigns a semantic class to all input spatial map points. To train the semantic decoder, we leverage the cross-entropy loss function. Given pairs of 3D points and their semantic class label $\{(p_\mathrm{i}, s_\mathrm{i})\}_{\mathrm{i}=1}^{n}$, we compute the semantic loss by minimizing:

$$L_{\text{sem}}(s, \hat{s}) = -\sum_{k=1}^{\mathrm{j}} s_k \log \hat{s}_k \tag{3.8}$$

By minimizing this loss, the semantic decoder assigns the semantic object label to each of the N points in the map.

3D Location Classifier

LocIn leverages the contextual patterns extracted from its object and semantic decoders to perform location classification. LocIn's location classifier predicts the user's location based on the composite learned representation of the object and semantic decoders, as shown in Eq. 3.4.

The location classifier processes the spatial feature representation Z obtained from LocIn's encoder through an MLP network which concatenates the skip-linked features from the encoder's intermediary set abstraction layers with Z. The concatenated features are then fed to a series of fully connected layers followed by a softmax operation that outputs the probability vectors for the c location classes.

3.6 Implementation

We implemented LocIn in Python 3.6 with PyTorch 1.2 [100].

LocIn's Network Architecture. We implement LocIn's spatial understanding encoder with four set abstraction layers and consider the output of last layer as the encoder's spatial feature representation (Z). For LocIn's object decoder, we implement two feature propagation layers and concatenate the skip-linked features from the encoder's second, third, and fourth set abstraction layers to the feature propagation layers' output. We use the output of the second feature propagation layer as the input *seed* points for generating votes and then process the votes for object proposal and classification through two MLP networks. We use the smooth-L_1 loss [101] for computing the box regression loss (L_{box}) and PyTorch's CrossEntropyLoss [102] for objectness ($L_{\text{obj-cls}}$) and semantic classification ($L_{\text{sem-cls}}$) losses.

LocIn's semantic decoder consists of four feature upsampling layers that upsample the points and features from LocIn's spatial encoder to $N \times R$ semantic features. Lastly, LocIn's location classifier uses three fully connected layers followed by a drop-out and softmax layer to predict the probability vectors for all location classes.

LocIn Training and Inference. To train LocIn's network, we subsample the input spatial maps' 3D points to a fixed number of $N = 4096$ points. To address spatial maps' rotation and scale variations, we augment our dataset by randomly flipping the maps in the horizontal direction, rotating the 3D points by $\pm 5°$ around the map's z-axis, and scaling the map by a factor of 0.85 to 1.15. We use the Adam optimizer [103] with a batch size of 8 and a learning rate of 0.001. We adopt a grid search to set the optimal values for α, β and γ in LocIn's optimization function (Eq. 3.4). We also empirically set the object decoder's loss parameters i.e., λ_1, λ_2 and λ_3 such that each component of the loss function is similar in scale.

At inference time, LocIn takes a 3D spatial map collected by an MR app and predicts the user's location through one forward pass of its encoder-decoder network. While LocIn's object and semantic decoders generate the bounding boxes for objects and semantic segmentation of the input spatial map, we only consider the location prediction as LocIn's output.

3.7 Evaluation

We describe our experience of applying LocIn attack to 3D spatial maps collected from three popular MR devices. We show that LocIn achieves an average accuracy of 84.1% in inferring location on two publicly available spatial map datasets. We also demonstrate that LocIn's attack is generalizable to other MR devices equipped with different depth sensors.

We additionally show the effectiveness of our multi-task learning-based location decoding model through an ablation study and demonstrate LocIn's robustness against different spatial map sizes and point densities. Lastly, we compare LocIn with various baselines and prior work [68] for recognizing indoor locations and show it achieves higher accuracy.

We present LocIn's results by focusing on the following research questions:

RQ1 How effective is LocIn in inferring users' location?

RQ2 How effective is LocIn's multi-task learning decoder?

RQ3 How does the sparsity of the 3D spatial map affect LocIn's performance?

RQ4 What is the impact of a 3D spatial map's size on LocIn's effectiveness?

RQ5 How generalizable is LocIn attack?

RQ6 How does LocIn compare against baseline methods?

RQ7 How does LocIn compare to other spatial attacks?

3.7.1 Evaluation Setup and Datasets

We evaluate LocIn's effectiveness on spatial maps captured from three MR devices, (1) iPad with depth sensor, (2) iPad with LiDAR scanner, and (3) HoloLens 2. The spatial maps from two iPad versions (with depth sensor and LiDAR scanner) are from two publicly available datasets [71, 72] while we create our dataset of HoloLens 2 spatial maps – Holo3DMaps. Table 3.1 presents the detailed statistics of the three datasets.

Table 3.1. Details of the evaluation dataset.

Dataset	MR Device	# of Location Classes	# of Object Classes	# of Spatial Maps	
				Training	Test
ScanNet	iPad Air2 with depth sensor	13	18	1201	312
ARKitScenes	iPad Pro with LiDAR scanner	9	17	4482	548
Holo3DMaps	HoloLens	5	8	-	20

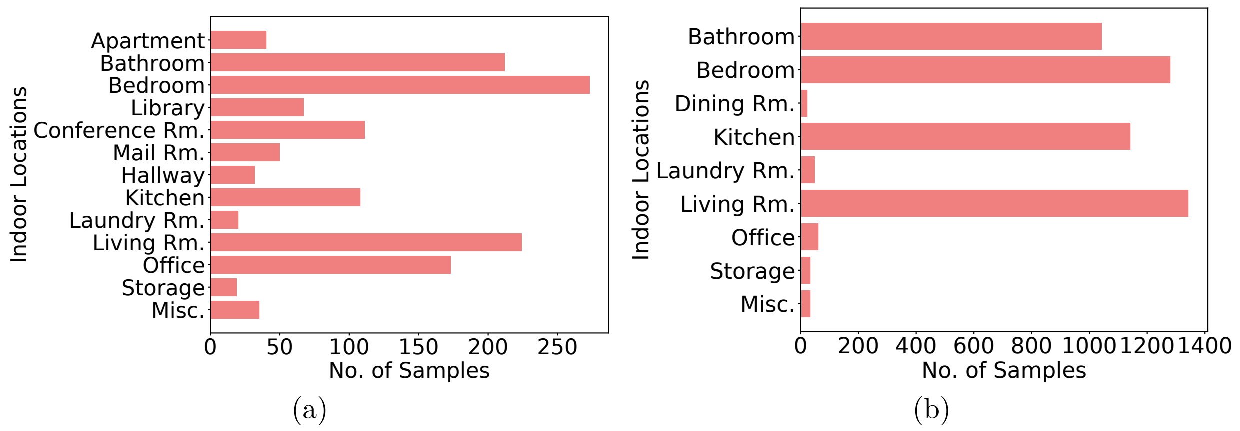


Figure 3.8. Distribution of the indoor location types across (a) ScanNet and (b) ARKitScenes datasets.

ScanNet Dataset. ScanNet [71] is a richly annotated dataset consisting of 1,513 3D spatial maps captured from 707 distinct indoor environments via a depth sensor attached to an iPad Air2. These spatial maps belong to 13 indoor location types and include annotations for 18 object categories.

Figure 3.8a shows the distribution of the number of samples for each location type in the dataset. The "Miscellaneous" location type is assigned to samples that do not distinctively belong to the other location types. We use 1,201 maps from the dataset for training while the remaining for testing. Each spatial map, on average, includes 150K points with a spatial extent of 5.5m Œ 5.1m Œ 2.4m.

ARKitScenes Dataset. ARKitScenes [72] is the first indoor 3D spatial map dataset captured via the Apple LiDAR scanner. It consists of 5,048 spatial maps from 1,661 unique indoor environments in real-world homes. The dataset includes ground truth for the oriented 3D bounding boxes of room-defining objects belonging to 17 different object categories. However, the dataset does not contain the ground truth for location labels and point-wise semantic labels for spatial maps.

We designed a semi-automated approach to generate the ground truth for location and semantic segmentation labels for the dataset (Detailed in Appendix 3.10). Figure 3.8b illustrates the sample distribution for each location type after labeling. We use 4,482 maps for training while the remaining for testing.

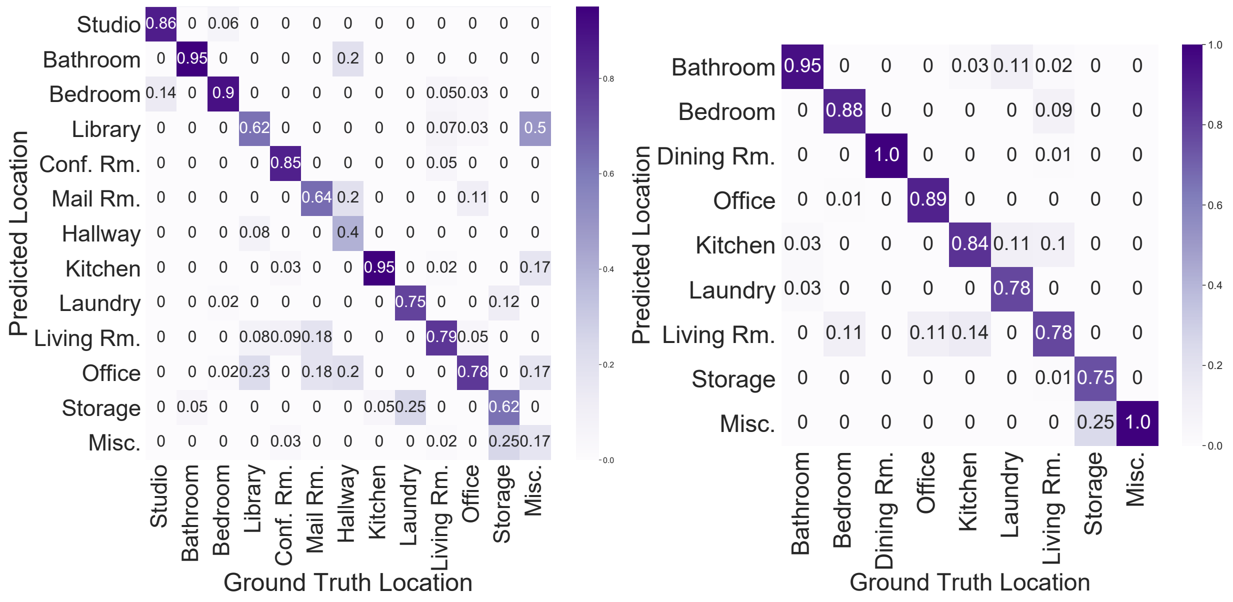
Holo3DMaps Dataset. To evaluate LocIn's effectiveness on various MR devices, we collected our own dataset of spatial maps using HoloLens 2 (there is no publicly available 3D scene understanding dataset captured using HoloLens). We scanned 20 different indoor environments where HoloLens is typically used. These environments are from 5 location types, including bedroom, living room, office, conference room, and kitchen (4 samples per class). We leveraged Microsoft MRTK's Spatial Mapping [79] to extract the 3D spatial maps of these environments. Two researchers manually annotated the spatial maps to generate the ground truth for location, object detection, and semantic segmentation labels.

Ethical Considerations. We collected Holo3DMaps from public places (e.g., labs and common rooms) and a hotel on a university campus. For each environment, we ensure that no human subjects are present during the data collection process and do not collect any personally identifiable information (PII). For the hotel environment, we received permission from the hotel management to visit unoccupied hotel rooms and suites to collect the maps. We contacted our university's IRB office and got advised that IRB approval is not required since our environments do not include any human subjects and we do not collect any sensitive information.

Evaluation Setup. We train the LocIn attack model on the training samples from the ScanNet and ARKitScenes datasets separately. We split each dataset into disjoint training and test sets such that each set's indoor environments are distinct. This splitting ensures that LocIn's reported results are independent of users and prior knowledge about their environments. We subsample each spatial map to 4,096 points through LocIn's preprocessing

Table 3.2. LocIn's overall attack effectiveness.

Dataset	Avg. Accuracy	Avg. Precision	Avg. Recall
ScanNet	81.6%	82%	81.6%
ARKitScenes	85.6%	85.7%	85.6%



(a) (b)

Figure 3.9. LocIn's confusion matrix on (a) ScanNet and (b) ARKitScenes dataset.

step. We perform all our experiments on a PC with 32 GB RAM and dual NVIDIA GTX 1080 Ti SLI GPUs. We provide LocIn's implementation details in Appendix 5.4.

3.7.2 Overall Effectiveness (RQ1)

We measure the effectiveness of LocIn through three evaluation metrics: average accuracy, precision, and recall. We compute the average accuracy as the number of correctly predicted location types in the test set. We calculate the precision for each location type as the average ratio of correctly predicted spatial maps to the total number of spatial maps classified to that type. We report recall for each location type as the average ratio of the number of correctly

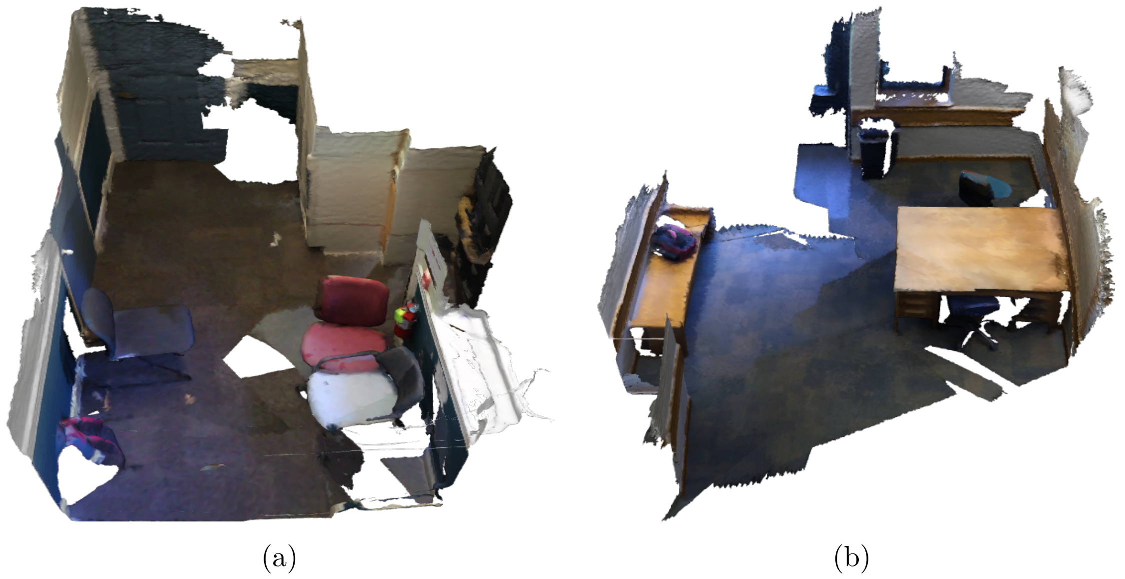


Figure 3.10. Examples of spatial maps of the "hallway" location type misclassified by LocIn.

predicted spatial maps to the total number of spatial maps of the given type. Table 3.2 shows LocIn's results for ScanNet and ARKitScenes.

Evaluation Results with ScanNet. LocIn infers the location from the spatial maps with an average accuracy of 81.6% with 82% and 81.6% average precision and recall rate. Figure 3.9a shows the confusion matrix of the location inference results.

LocIn classifies indoor environments, including distinct object types, correctly with high accuracy. For instance, bedroom, bathroom, and kitchen include unique objects (e.g., bed, sink, and stove) that provide semantic context to LocIn's location decoder and are classified with $> 90\%$ accuracy. In contrast, location types that typically lack such distinct objects have comparatively lower accuracy. For example, spatial maps of "library" type with only chairs and tables are misclassified to "office" since "office" spatial maps include the same object types and share similar geometric structures.

We found that the majority of the spatial maps for location types with low accuracy (e.g., "hallway") included objects commonly found in other location types or no objects, causing LocIn to misclassify them. To illustrate, we present two examples of spatial maps

Table 3.3. Effect of individual decoders in LocIn's multi-task decoder on its performance.

Dataset	**LocIn$_{\textbf{LOC}}$**	**LocIn$_{\textbf{OBJ}}$**	**LocIn$_{\textbf{SEM}}$**	**LocIn**
ScanNet	57%	80.1%	78.4%	81.6%
ARKitScenes	58.6%	83.7%	79.7%	85.6%

from the "hallway" class in Figure 3.10. These spatial maps share similarities with office environments; thus, they are misclassified.

Evaluation Results with ARKitScenes. LocIn achieves an average accuracy of 85.6% in inferring the locations of ARKitScenes spatial maps. It classifies the 9 location types with a precision and recall rate of 85.7% and 85.6%. Figure 3.9b presents the confusion matrix for ARKitScenes results.

The number of spatial maps per class in ARKitScenes is hugely imbalanced (as shown in Figure 3.8). However, because LocIn leverages the class weights in the cross-entropy loss function for its location classifier during training, it accurately infers the location types with few training samples. Similar to ScanNet, LocIn accurately detects distinct indoor locations (e.g., bathroom, bedroom, kitchen) with high accuracy.

LocIn Inference Time. We evaluate LocIn's inference time by measuring the average time taken to predict the user's location type from spatial maps collected through an iPhone 14 equipped with a LiDAR scanner. Overall, given a spatial map subsampled to 4,096 points, LocIn takes 0.89s on average to predict the user's semantic location. Specifically, LocIn's object decoder takes 0.64s on average to generate the bounding boxes for the detected objects and predict their labels. LocIn's semantic decoder predicts the semantic labels for each point in the spatial map within 0.25s on average. The low inference time of LocIn's multi-task framework allows an adversary to infer a user's location in real-world MR apps.

3.7.3 Effectiveness of Decoders (RQ2)

To understand how each component of LocIn contributes to its performance, we perform an ablation study by training three models: (1) location classifier without LocIn's multi-task optimization function, (2) LocIn's object decoder with location classifier and (3) LocIn's semantic decoder with location classifier and comparing their performance with LocIn. LocIn's location classifier in (2) and (3) is solely needed to infer the location labels from the detected objects and semantic features. Table 3.3 shows the average accuracy of LocIn compared to these simplified models.

Location Classification. As discussed in Section 3.5.3, an adversary could infer a user's location by training a DNN classifier directly on the spatial maps. We compare LocIn to a location classifier (LocIn$_{\text{LOC}}$) trained on the spatial maps solely with the cross-entropy loss for classification (See Eq. 3.3).

Table 3.3 shows that LocIn$_{\text{LOC}}$ without LocIn's multi-task learning optimization function can only achieve 57% and 58.6% accuracy on ScanNet and ARKitScenes datasets. This is because without leveraging context (i.e., object and semantic patterns), the classifier extracts only high-level geometric features (i.e., structure and appearance of the environment). These features are unable to distinguish different indoor locations due to similarities in their structure and appearance, resulting in lower classification accuracy compared to LocIn.

Object Detection. We study the impact of object detection on LocIn's performance by building a model, LocIn$_{\text{OBJ}}$, that only leverages the LocIn's object decoder for classification i.e., we only consider L_{loc} and L_{obj} (in Eq. 3.4) for training.

We found that integrating object detection for location classification significantly improves the accuracy of the location classifier. With object decoder, LocIn$_{\text{OBJ}}$ can achieve an average accuracy of 81.9% that is ~24% higher than the model trained without object detection (LocIn$_{\text{LOC}}$). This gain in accuracy shows that objects uniquely characterize indoor location types and enable accurate discrimination between them.

Semantic Segmentation. We study how semantic decoder influences LocIn's performance by training a model, LocIn$_{\text{SEM}}$, that combines only L_{loc} and L_{sem} in Eq. 3.4. It improves accuracy by ~22% compared to the location classifier, LocIn$_{\text{LOC}}$. The semantic decoder helps LocIn extract fine-grained details (e.g., planar surfaces, sparse/small objects) about the environment that help in location classification.

Table 3.4. LocIn's results with varying sparsity of spatial map.

Dataset	512	1024	2048	4096
ScanNet	71.7%	75.5%	78.4%	81.6%
ARKitScenes	75.6%	79.7%	83.4%	85.6%

We note that the accuracy gain with LocIn$_{\text{SEM}}$ is 2% less than that from LocIn$_{\text{OBJ}}$. This is because predicting point-wise semantic object labels is more prone to errors resulting from sparse or missing points on an object's surface. However, combining the object and semantic decoder in LocIn's multi-task decoder provides higher accuracy in inferring locations than using them individually for location classification. This accuracy gain demonstrates the effectiveness of LocIn's unified multi-task network architecture.

3.7.4 Parameter Analysis

We evaluate the impact of the input spatial map's point density and size on LocIn's performance.

Spatial Map Sparsity (RQ3). The results reported in the previous sections are achieved on spatial maps with $N = 4096$ points. We now evaluate the impact of varying the number of points, N, subsampled in LocIn's preprocessing module on its effectiveness. Table 3.4 shows the accuracy of LocIn on spatial maps with varying sparsity.

LocIn achieves more than 70% accuracy on both datasets even when the input spatial maps have fewer points i.e., $N = 512$. We note that even with sparse maps, LocIn's location decoder extracts the environment's structural and semantic properties, sufficient for accurate location classification. This ensures that LocIn infers the user's location even if the map is captured while the user quickly scans their environment.

Spatial Map Size (RQ4). An important factor concerning LocIn's practicality is the input spatial map's size since MR device depth sensors have a limited field of view of the user's environment due to object/surface occlusions or their non-panoramic nature. We define size as the area of the user's environment captured by the MR device.

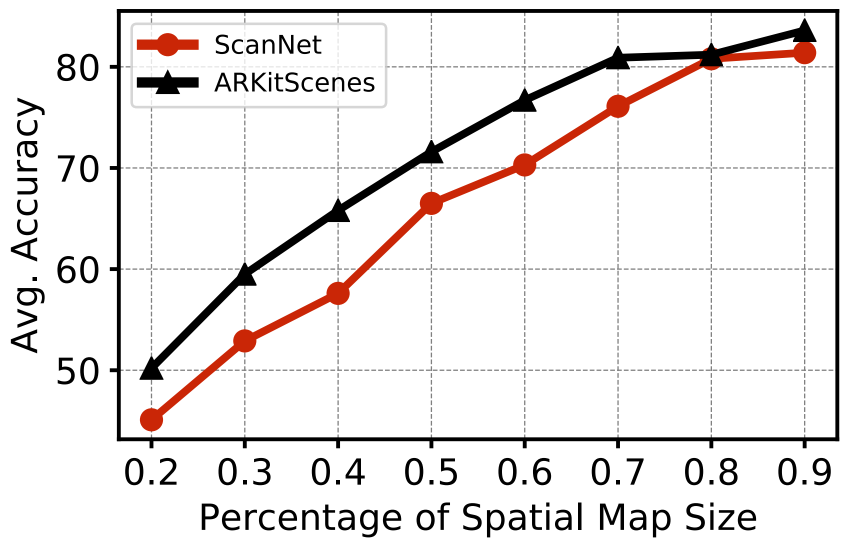


Figure 3.11. LocIn's performance with varying map size.

To this end, we evaluated the ScanNet and ARKitScenes datasets by creating submaps from each spatial map in the test dataset by randomly selecting a point in the map and cropping a bounding box around it. The resulting submaps only include a subset of objects and walls (including the case where no walls are sampled) present in the user's environment. The size of the bounding box is a percentage of the original map's size. We discard submaps with less than $N = 4096$ points.

Figure 3.11 presents the average accuracy of LocIn on submaps generated from the two datasets as we vary the percentage of the indoor environment's size captured by the MR device. LocIn achieves an average accuracy of 69.1% in classifying the indoor location type when the spatial map captures only 50% of the environment. The accuracy improves to 77% as the spatial map size increases to 70%. LocIn effectiveness deteriorates as the size of the map decreases since the cropped submaps lack complete semantic and object details necessary for distinguishing the indoor location (e.g., cropped/missing bed in a bedroom's submap).

We note that we perform this evaluation on LocIn model trained on the complete spatial maps of the indoor environments. Hence, one possible approach to improve LocIn's robustness to map size would be to train on cropped submaps.

Table 3.5. LocIn's performance on Holo3DMaps dataset.

Training Depth Senor	Avg. Accuracy	Avg. Precision	Avg. Recall
Indirect ToF (ScanNet)	85%	85%	86%
LiDAR (ARKitScenes)	45%	45%	60%

3.7.5 Generalizability of LocIn (RQ5)

We perform LocIn's main evaluation on the two publicly available datasets (i.e., ScanNet and ARKitScenes) collected using iPads with two different depth sensors. To evaluate LocIn's generalizability to other MR devices with different depth-sensing technologies, we collected our own dataset (Holo3DMaps) of spatial maps from indoor environments using Microsoft's HoloLens 2 equipped with indirect Time-of-Flight (ToF) depth sensor [77]. We evaluate LocIn's performance on Holo3DMaps through two models: (a) LocIn trained on a dataset collected using indirect Time-of-Flight depth sensor (ScanNet) and (b) LocIn trained on a dataset collected using LiDAR scanner (ARKitScenes).

Table 3.5 shows LocIn's effectiveness on Holo3DMaps on the two models. LocIn achieves 80% accuracy in inferring user location when trained on the ScanNet dataset. Interestingly, LocIn's accuracy deteriorates when evaluated on the model trained with ARKitScenes dataset. This is due to the resolution differences in the spatial maps captured by HoloLens 2 and iPad with LiDAR scanner. HoloLens 2 employs indirect Time-of-Flight (ToF) depth sensor that illuminates the observed scene with infrared light and uses the reflected light to calculate scene depth [104]. In contrast, iPad's LiDAR scanner (direct ToF) uses timed light pulses to measure scene depth, instead of illuminating the whole scene with modulated light like an indirect ToF sensor [105].

Due to the differences in depth calculation methods, the spatial maps of two devices have inherent differences in their 3D point cloud sparsity. Thus, if LocIn is trained on LiDAR spatial maps (ARKitScenes), the features from spatial maps in Holo3DMaps are inconsistent with the features observed during training, causing LocIn to misclassify them.

Contrarily, given that both ScanNet and Holo3DMaps are created using indirect ToF sensors, LocIn trained on ScanNet dataset generalizes well to Holo3DMaps. Therefore, an adversary can leverage LocIn trained on one MR device to attack a different device with the same depth-sensing technology.

3.7.6 Comparison with Baseline (RQ6)

We compare LocIn's effectiveness against three baseline approaches on ScanNet and ARKitScenes datasets. First, we compare LocIn with a naive ML classifier (Base-RF$_{\text{obj}}$) that leverages the objects in the user's environment. For this, we predict the semantic labels for objects using VoteNet [93] and generate a histogram of the objects present in a given location type. We then train a Random Forest classifier on these object class histograms to predict the location. We choose Random Forest classifier as it classifies the point cloud based on statistical features from the object histogram. Second, we compare LocIn against another Random Forest classifier trained on the object class histogram generated from ground-truth object semantic labels (Base-RF$_{\text{obj}}$). This eliminates the impact of errors in object detection and evaluate how object information without spatial information impacts location classification accuracy. Lastly, we compare LocIn with a baseline deep neural network that takes the spatial representation obtained from PointNet++ [70] as input and processes it through a series of fully connected layers followed by a softmax operation to predict the location class (Base-DN).

Table 3.6 shows the accuracy of the baseline models in contrast to LocIn. Base-RF$_{\text{obj}}$ achieves an accuracy of only 29.6% and 38.8% on ScanNet and ARKitScenes, respectively. This is because Base-RF$_{\text{obj}}$ does not consider the spatial and geometric information in the spatial map. We find that the accuracy for location types with distinct objects (e.g., bed in bedroom and toilet in bathroom) is higher than location types with no distinct objects (e.g., office and living room). Moreover, the errors in the object detection model contribute to inaccuracies in the object class histograms and, in turn, the location inference accuracy.

Base-RF$_{\text{ground}}$ achieves a higher location inference accuracy of 41.7% on ScanNet and 44.2% on ARKitScenes. While this model eliminates the impact of errors in object detection,

Table 3.6. LocIn's comparison with baseline approaches.

Dataset	$\textbf{Base-RF}_{\textbf{obj}}$	$\textbf{Base-RF}_{\textbf{ground}}$	**Base-DN**	**LocIn**
ScanNet	29.6%	41.7%	57.1%	81.6%
ARKitScenes	38.8%	44.2%	58.6%	85.6%

it still does not capture the correlation between spatial, geometric, and semantic features of a spatial map. Similarly, Base-DN extracts high-level features, such as structural and geometric properties (e.g., length and width or floor map) of the users environment. However, since indoor environments of the same type vary significantly in their geometric and structural properties (e.g., size of two bedrooms in a house), it is difficult to predict the specific location type, resulting in a low accuracy of 57.1% and 58.6% on ScanNet and ARKitScenes datasets, respectively. In contrast to these models, LocIn's multi-task learning framework performs feature extraction and loss minimization that simultaneously capture both geometric properties and semantic context of the users environment to accurately infer the location.

3.7.7 Comparison with Prior Work (RQ7)

We compare LocIn with a recent work [68], the only prior work that leverages 3D spatial maps from an MR device to infer the user's indoor location. This work extracts 3D spin image features [106] from the input spatial map and employs nearest neighbor distance and deep learning-based 3D place recognizer (PointNetVLAD) [107] to determine if the map's feature vector matches with one of the maps from the user's previously visited locations. However, their attack has limited practicality as it assumes that the attacker has access to a labeled dataset of 3D spatial maps of the target user's indoor locations. Therefore, it only targets a specific user and her previously visited locations.

To evaluate how this work compares against LocIn in a realistic attack scenario (no prior knowledge about a user is available), we implement its DNN place recognizer [107] to extract global features from 3D spatial maps. Since our goal is to infer the location type, we replace its final Euclidean distance-based feature matching step with our location classifier that processes the spatial feature vector through an MLP followed by a softmax operation.

We train this model on the ScanNet dataset with the same training parameters as described in [107]. It achieves an accuracy of 50.1% on ScanNet's test dataset which is much lower compared to LocIn's 81.6% accuracy. This significant difference in accuracy shows that LocIn is more effective in inferring users' location from spatial maps without any prior knowledge about the user's location.

3.8 Limitations and Discussion

Inferring Location of Complex Environments. LocIn can infer users' location from 3D spatial maps in various indoor environments with common objects (Section 5.5.1). However, if a user's environment includes unique objects or very few objects or planar surfaces, LocIn's effectiveness will mainly rely on its location classification decoder. This is because LocIn's object and semantic segmentation decoders cannot extract meaningful patterns from the input map. We conducted an additional experiment to evaluate LocIn's effectiveness on spatial maps where no objects are present in the user's environment. In this case, LocIn infers the users' environment based on its geometric properties with an accuracy of 59.1%.

Generalization to Various MR Devices. We demonstrate LocIn attack on three popular MR devices that leverage depth cameras or LiDAR sensors to capture the spatial map of a user's environment. However, spatial maps from different MR devices have varying sparsity and non-uniform point density because various MR devices employ different sensors and depth calculation techniques to generate the 3D spatial maps [94, 104]. For instance, in contrast to LiDAR scanner-based devices, devices leveraging ToF depth cameras (e.g., smartphones, HoloLens) build spatial maps based on the reflection of light pulses emitted by the device. As a result, they experience difficulty in capturing a complete 3D representation of reflective surfaces and objects, such as mirrors, polished metal, or very dark surfaces, producing more sparse spatial maps. Although we show that LocIn attack is transferrable across devices that leverage the same depth sensor type, this inherent difference in spatial map properties requires building a specialized model for each MR device.

Predicting Unknown Location Types. We demonstrate in Section 3.7 that LocIn can effectively identify 13 indoor location types where MR devices are typically used. LocIn,

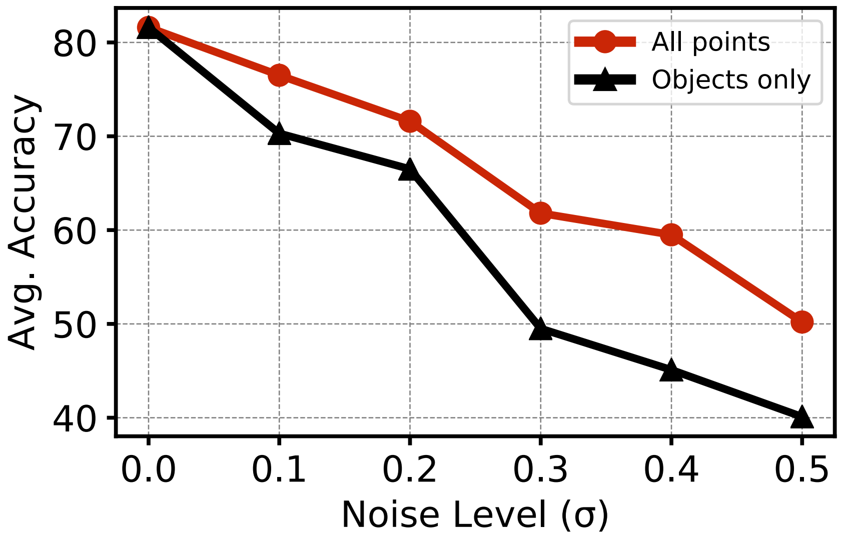


Figure 3.12. LocIn's performance with varying noise levels.

however, is limited to inferring location types observed during its training process. Thus, if the user interacts with their MR device in a location not included in the LocIn's training, LocIn cannot correctly infer the location from the spatial map. An adversary can leverage outlier detection techniques to detect samples out of distribution from the known location classes [108, 109] and collect a more comprehensive dataset with diverse labels for training LocIn.

LocIn Counter Measures. One possible defense against LocIn is to limit the MR app's access to the raw 3D spatial maps and only share privacy-preserving features (e.g., planar surfaces or 3D points of a user-defined region) to enable MR content. Yet, this would affect MR apps' functionality as they rely on detailed spatial understanding to enable MR content, requiring further investigation into its usability for MR apps.

Another possible defense is to inject noise into the 3D spatial maps shared with the MR apps to force LocIn to misclassify the user's location. To test this defense, we conducted two sets of experiments on the ScanNet dataset. First, we applied random perturbations to all points in the spatial maps in our test dataset by adding Gaussian noise to their points' 3D coordinates and predicted their location using LocIn trained on raw spatial maps. Second, we applied random perturbations of varying intensity only to points belonging to objects present within the spatial maps. In both cases, we change the noise intensity by changing the standard deviation (σ) of the added noise. Figure 3.12 presents the change in accuracy with varying noise for both experiments. We observe that LocIn's accuracy drops to 61% when all points are perturbed in the spatial maps by a noise level of $\sigma = 0.3$. Similarly, when similar noise is added to only objects, LocIn's accuracy deteriorates to 49.5%. This decrease in accuracy occurs since noisy points in the spatial map make it difficult to detect the objects and semantic features of the user's environment.

Although noise injection reduces LocIn's effectiveness, its feasibility is limited as the perturbed spatial maps reduce usability for the MR apps. For instance, MR apps leverage the spatial map to localize a user in its environment [110]; thus, a perturbed spatial map may result in erroneous localization results, affecting the apps' functionality. Moreover, implementing this defense requires evaluating different spatial map usage scenarios to ensure app functionality is not affected.

3.9 Related Work

Privacy Leakage in Mobile Mixed Reality. Recent works have exposed attacks that leverage the multi-modal sensors on MR devices, similar to privacy leakages in other mobile devices [2, 16, 111]. For instance, a line of work proposed virtual keystroke detection side-channel attacks on MR devices by exploiting the channel state information (CSI) of WiFi signals [112], headset motion sensors [113, 114] and IMU sensors on MR device hand controllers [115]. Another work [116] proposed an eavesdropping attack through motion sensors on head-mounted MR devices to detect facial movements for inferring human speech. In this chapter, we show a new privacy leakage from MR devices by exploiting the 3D spatial maps to infer the user's location type.

Location Inference Attacks. Several works exploit radio frequency (RF) signals emitted by commodity devices to infer users' location [117, 118]. However, these approaches require a physically proximate attacker to deploy sniffing devices near users. In contrast, LocIn attack operates remotely without requiring any additional devices or information about the user.

Another line of attacks exploits mobile sensors, e.g., microphone [119] and IMU sensors [120], to infer users' location. Yet, these works only localize a user with respect to the mobile device and require an indoor map of the environment.

Previous approaches have also investigated indoor location inference from images and videos through hand-crafted features and deep learning models [62, 63, 65–67]. Yet, these attacks are ineffective in various scenarios. First, these attacks are sensitive to lighting and occlusion, decreasing their location inference accuracy in low-lighting conditions or when objects are partially obscured. Second, the accuracy of these attacks is influenced by different camera orientations/viewpoints while capturing images/videos. Lastly, given users' privacy concerns surrounding apps' access to camera images and videos, several privacy-preserving approaches attempt to eliminate location attacks through visual data [87, 89, 90] and various MR devices limit apps' access to image and video data. In contrast, LocIn exploits spatial maps, shared with apps to enable device localization, to infer locations from different camera orientations/viewpoints regardless of low lighting and occlusion.

A recent work [68] leveraged spatial data on MR devices to recognize users' previously visited locations. While this work, similar to LocIn, exploits the spatial data on MR devices to infer private user information, its practicality is limited as it only recognizes a specific user's previously visited indoor locations. Contrarily, LocIn infers a user's location without any prior information through its semantic aware multi-task network and generalizes well to unseen users (Section 3.7.7).

Multi-task Learning on 3D Data. Several prior works have leveraged multi-task learning for scene understanding tasks (e.g., object detection, semantic segmentation, object classification) on 3D data [93, 121–124]. Previous works [123, 124] simultaneously learned embedded features for 3D object instance segmentation and semantic segmentation through a combined loss function. A recent work [122] learned the shape of 3D objects and their labels simultaneously based on objects' curvature. A line of work [121, 125] used multi-task learning to improve a robot's ability to recognize a scene by combining color and geometric properties of 3D data. In contrast, LocIn leverages 3D objects and semantic context of a user's environment from 3D spatial data to infer its location.

3.10 Labeling ARKitScenes Dataset

We describe our approach for generating ground truth labels for location type and point-wise semantic labels for the ARKitScenes dataset [72].

Table 3.7. Location type to object mapping.

Location Type	Object Type
t_0 : Living Room	sofa, fireplace
t_1 : Bedroom	bed
t_2 : Kitchen	stove, dishwasher, oven, refrigerator
t_3 : Bathroom	bathtub, toilet

Location Type Labels. We adopted a semi-automated annotation process to label each spatial map in ARKitScenes with its location type, Since ARKitScenes includes ground truth for the objects (including bounding boxes and object semantic labels), we leverage the fact that objects uniquely characterize indoor environments to assign an initial location label to each spatial map. Specifically, two researchers developed a mapping (M) between typical indoor environments and the objects that uniquely identify them (Table 3.7). For instance, a bedroom must have a bed, and a kitchen must have a stove. This mapping includes four location types commonly observed in real-world homes where ARKitScenes is collected and a subset of objects from the 17 objects types in the ARKitScenes dataset. We assigned an label to each spatial map through the function $l = \arg\max(t_0, t_1, t_2, t_3)$ where

$$t_\mathrm{i} = \Sigma(u_0, u_1, \ldots, u_m), \text{ where } u_\mathrm{j} = \frac{k}{U} \tag{3.9}$$

Here, k is the number of objects of type u_j in the spatial map, m is the number of object types present in the mapping for location type (t_i), and U is the total number of objects in the input spatial map that belong to location type t_i in M.

Two researchers then manually inspected and verified the initial labels by visualizing the spatial map and its associated images available in the dataset. We found that among 5,048 spatial maps in the dataset, 256 maps did not include any objects in the location-object type mapping in Table 3.7 and hence could not be labeled in the initial labeling process. The two researchers manually labeled these 256 samples individually and assigned them labels following the location types in the ScanNet dataset. The authors then met to discuss and reconcile differences. We assigned the "Miscellaneous" label to samples for which no conclusive location type could be derived from the spatial map and its corresponding images.

Semantic Segmentation Labels. We used the ground truth for 3D objects in the maps to assign the semantic label to each point based on the object the point belonged to. For instance, we assigned the semantic label "bed" to all points within the bounding box of the bed object in a given spatial map. This approach, however, only considers points that belong to one of the 17 object types annotated in the ARKitScenes dataset. These types do not include planar surfaces such as walls, floors, and ceilings. Hence, to annotate points on these planar surfaces, we adopted a semi-automated annotation approach.

We first identified an estimate of all planar surfaces in a given spatial map through Random Sampling Consensus (RANSAC) [126] algorithm for plane detection. For this, we employed an iterative procedure that randomly samples a subset of points from an input spatial map and fits a plane equation on these points. The number of points that satisfy the plane equation (inliers) in each iteration is used to calculate a confidence score for the plane equation. We then marked the plane equation, which achieves the highest confidence score as a planar surface in the spatial map. We used this process to identify the horizontal and vertical planar surfaces (along x and y axes of the spatial maps) only because these planes represent the walls, floors, and ceiling in the spatial maps.

Two researchers manually inspected the plane detection output and corrected errors in the point-wise semantic labels. Through this procedure, we annotated all spatial maps in the dataset and assigned the points to one of 20 object types (17 objects in ARKitScenes and wall, floor and ceiling).

4. FACEREVELIO: A FACE LIVENESS DETECTION SYSTEM FOR SMARTPHONES WITH A SINGLE FRONT CAMERA

4.1 Introduction

Considering the growingly extensive use of smartphones in all aspects of our daily life, reliable user authentication for securing private information and mobile payments is an absolute necessity. Recent years have witnessed a rising usage of face authentication on smartphones as a promising alternative to traditional password-based protection mechanisms. Most of the existing face authentication systems use traditional 2D face recognition technologies, which suffer from vulnerability to *spoofing* attacks where the attacker uses 2D photos/videos or 3D masks to bypass the authentication system.

Recently, some smartphone manufacturers have introduced *liveness detection* features to some of their high-end products, e.g. iPhone X/XR/XS and HUAWEI Mate 20 Pro. These phones are embedded with specialized hardware components on their screens to detect the 3D structure of the user's face. For example, Apple's TrueDepth system [127] employs an infrared dot projector coupled with a dedicated infrared camera beside its traditional front camera.

Although effective, deployment of such specialized hardware components, adding a notch on the screen, is against the bezel-less trend in the smartphones' market. Customers' desire for higher screen-to-body ratio has consequently forced manufacturers to search for alternative methods. For example, Samsung recently launched S10 as its first phone with face authentication and an Infinity-O hole-punch display. However, S10's lack of any specialized hardware for capturing facial depth, made it an easy target for 2D photo or video attacks [128].

Therefore, we ask the following question: How can we enable liveness detection on smartphones only relying on a single front camera?

Prior works on face liveness detection for defense against 2D spoofing attacks have relied on computer vision techniques to detect and analyze facial liveness clues like blinking and eye movements [129], nose and mouth features [130, 131], and skin reflectance [132]. Usually, extracting such characteristics from a face requires ideal lighting conditions, which are hard to guarantee in practice. Another common approach is the use of challenge-response protocols where the user is asked to respond to a random challenge, such as pronouncing a word, blinking or other facial gestures. These techniques, however, are unreliable because facial gestures can be simulated using modern technologies, such as media-based facial forgery [133]. A time-constrained protocol was recently introduced to defend against these attacks, which however still required the users to make specific expressions [9]. The additional time-consuming efforts and their reliance on users' cooperation, make such protocols harder to use in many scenarios, including but not limited to elderly usage and emergency cases.

In this chapter, we introduce a novel face liveness system, *FaceRevelio*, that only uses the front camera on commodity smartphones. Our system reconstructs 3D models of users' faces in order to defend against 2D-spoofing attacks. *FaceRevelio* exploits smartphone screens as light sources to illuminate the human face from different angles. Our main idea is to display combinations of light patterns on the screen and simultaneously record the reflection of those patterns from the users' faces via the front camera. We employ a variant of photometric stereo [134] to reconstruct 3D facial structures from the recorded videos. To this end, we recover four *stereo images* of the face from the recorded video via a least squared method and use these images to build a normal map of the face. Finally, the 3D model of the face is reconstructed from the normal map using a quadratic normal integration approach [135]. From this reconstructed model, we analyze how the depth changes across a human face compared to model reconstructed from a photograph or video and train a deep neural network to detect various spoofing attacks.

Implementing our idea of reconstructing the 3D face structure for liveness detection using a single camera involved a series of challenges. First, displaying simple and easily forgeable light patterns on the screen makes the system susceptible to replay attacks. To secure our system from replay attacks, we designed the novel idea of a *light passcode*, which is a random combination of patterns in which the screen intensity changes during the process of authentication, such that an attacker would be unable to correctly guess the random passcode. Second, in the presence of ambient lighting, the intensity of the reflection of our light passcode was small, hence difficult to separate from ambient lighting. In order to make *FaceRevelio* practical in various realistic lighting conditions, we carefully designed light passcodes to be orthogonal and "zero-mean" to remove the impact of environment lighting. In addition, we had to separate the impact of each pattern from the mixture of captured reflections to accurately recover the stereo images via the least square method. For this purpose, we linearized the camera responses by fixing camera exposure parameters and reversing gamma correction [136]. Finally, the unknown direction of lighting used in the four patterns causes an uncertainty in the surface normals computed from the stereo images which could lead to inaccurate 3D reconstruction. We designed an algorithm to find this uncertainty using a general template for human surface normals. We used landmark-aware mesh warping to fit this general template to users' face structures.

FaceRevelio is implemented as a prototype system on Samsung S10 smartphone. By collecting 3800 videos with a resolution of 1280×960 and a frame rate of $30fps$, we evaluated *FaceRevelio* with 30 volunteers under different lighting conditions. *FaceRevelio* achieves an EER of 1.4% for both dark and day light settings, respectively against 2D printed photograph attacks. It detects the replay video attacks with an EER of 0.0%, and 0.3% for each lighting, respectively.

The contributions of our research are summarized as follows:

1. We design a liveness detection system for commodity smartphones with only a single front camera by reconstructing the 3D surface of the face, without relying on any extra hardware or human cooperation.

2. We introduce the notion of *light passcodes* which combines randomly-generated lighting patterns on four quarters of the screen. Light passcode enables reconstructing 3D structures from stereo images and more importantly, defends against replay attacks.

3. We implement *FaceRevelio* as an application on Android phones and evaluate the system performance on 30 users in different scenarios. Our evaluations show promising results on applicability and effectiveness of *FaceRevelio*.

4.2 Background

In this section, we introduce photometric stereo and explain how it is used for 3D reconstruction under known/unknown lighting conditions.

Photometric stereo is a technique for recovering the 3D surface of an object using multiple images in which the object is fixed and lighting conditions vary [137]. Its key idea is to utilize the fact that the amount of light that a surface reflects depends on the orientation of the surface with respect to the light source and the camera.

Computing Normals under Known Lighting Conditions: Besides the original assumptions under which photometric stereo is normally used [137] (e.g. point light sources, uniform albedo, etc.), we now assume that the illumination is known.

Given three point light sources, the surface normal vectors S can be computed by solving the following linear equation based on the two known variables:

$$I^T = L^T S, \tag{4.1}$$

where $I = [I_1, I_2, I_3]$ is the stacked three stereo images exposed to different illumination, and $L = [L_1, L_2, L_3]$ is the lighting direction for these three images. Note that at least three images under variant lighting conditions are required to solve this equation and to make sure that the surface normals are constrained.

Computing Normals under Unknown Lighting Conditions: Now we consider the case when the lighting conditions are unknown. The matrix of intensity measurements is further denoted as M, which is of size $m \times n$ where m is the number of images. Therefore

$$M = L^T S. \tag{4.2}$$

For solving this approximation, M is factorized using Singular Value Decomposition (SVD) [138]. Using SVD the following is obtained

$$M = U\Sigma V^T. \tag{4.3}$$

This decomposition can be used to recover L and S in the form of $L^T = U\sqrt{\Sigma}A$ and $S = A^{-1}\sqrt{\Sigma}V^T$, where A is an 3×3 linear ambiguity matrix. [134] provides the details about how this equation can be solved with four images under different lighting conditions.

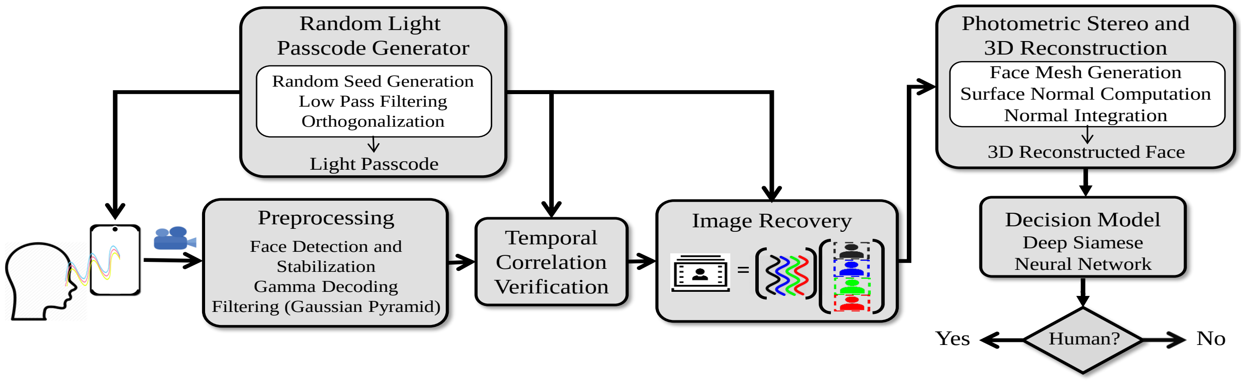


Figure 4.1. System overview

4.3 FaceRevelio System Overview

FaceRevelio is a liveness detection system designed to defend against various spoofing attacks on face authentication systems.

Figure 4.1 shows an overview of *FaceRevelio*'s architecture. It begins its operation by dividing the phone screen into four quarters and using each of them as a light source. *Random Light Passcode Generator* module is used to select a random light passcode which is a collection of four orthogonal light patterns displayed in the four quarters of the phone screen. The front camera records a video clip containing the reflection of these light patterns from the user's face. These light patterns are not only used during video recording, but also help reconstruct 3D structure of the face and detect replay attacks. The recorded video then passes through a preprocessing module where first face region is extracted and aligned in each adjacent video frame. This is followed by an inverse gamma calibration operation applied to each frame to ensure linear camera response. Finally, the video is filtered by constructing its Gaussian Pyramid [139], where each frame is smoothed and subsampled to remove noise. After preprocessing, a temporal correlation between the passcode in the video frames and the one generated by the *Random Light Passcode Generator* is checked. If a high correlation is verified, the filtered video frames along with the random light passcode are fed into an *Image Recovery* module. The goal of this module is to recover the four stereo images corresponding to the four light sources, by utilizing the linearity of the camera response. The recovered stereo images are then used to compute face surface normals under unknown lighting conditions using a variant of photometric stereo technique [134]. A generalized human normal map template and its 2D wired mesh connecting the facial landmarks are used to compute these normals accurately. A 3D face is finally reconstructed from the surface normals by using a quadratic normal integration method [135]. Once the 3D structure is reconstructed, it is passed on to a liveness detection decision model. Here, a Siamese neural network [140] is trained to extract depth features from a known sample human face depth map and the reconstructed candidate 3D face. These feature vectors are then compared via L1 distance and a sigmoid activation function to give a similarity score for the two feature vectors. The decision model declares the 3D face as a real human face if this score is above a threshold and detects a spoofing attack otherwise.

4.4 FaceRevelio Attack Model

Attacks to face authentication techniques can be classified into static and dynamic attacks. In a 2D static attack, a still object such as a photograph or mask is used, such that the face recognition algorithms would not be able to differentiate these presented objects from an actual face. Dynamic attacks aim at spoofing systems where some form of user action is required like making an expression or a gesture. In these attacks, a video of the user is replayed performing the requested action. These videos can easily be forged by merging user's public photos with its facial characteristics. Adversaries can also launch a 3D static attack by using 3D models of the face. However, this requires advanced 3D printing capabilities which requires high cost. Similarly, 3D dynamic attacks involving building a 3D model in virtual settings, are impractical as described in [9].

Our goal is to prevent adversaries from spoofing face authentication systems with 2D static and dynamic attacks. We assume that an attacker has access to high-quality images of the legitimate user's face. We also assume that the adversary can record a video of the user while using *FaceRevelio*. In this case, the recorded video will capture the light patterns' reflections from the user's face. The attacker prepares these videos beforehand and launches an offline attack on our system by displaying them on a laptop screen/monitor. Conducting an online attack is extremely difficult since our system displays a random passcode on the screen each time. This would require the attacker to use a very high-speed camera and computer to determine the random passcode and then generate a forged video response using this passcode, all within the duration of a frame of the passcode; hence impractical.

4.5 FaceRevelio System Design

4.5.1 Light Passcode Generator

To apply photometric stereo, we need to generate four images of the face illuminated under various light sources, from different directions. In order to simulate these light sources using the phone screen, we divide the screen into four quarters where each quarter is assumed a light source. During the video recording, each of these quarters is illuminated alternately in four equal intervals, while the other three quarters are dark. Figure 4.2 shows how the screen changes with different patterns during the four intervals and an example of the 3D reconstruction of the face using these patterns.

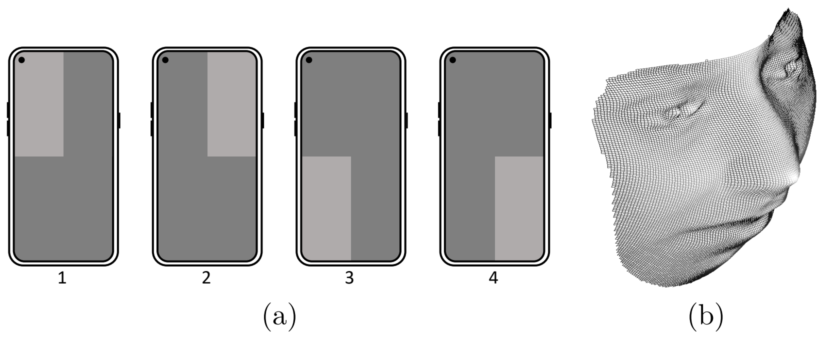


Figure 4.2. An example of 3D reconstruction using four basic light patterns displayed on four quarters of the screen.

Random Passcode Generator

It could be argued that using these basic light patterns, the system would be prone to replay attacks. Keeping this in mind, we consider the idea of illuminating all the four quarters together for a certain period and changing the screen lighting randomly at each time instance and each quarter to a random value between 0 and 1. Now, each quarter of the screen is

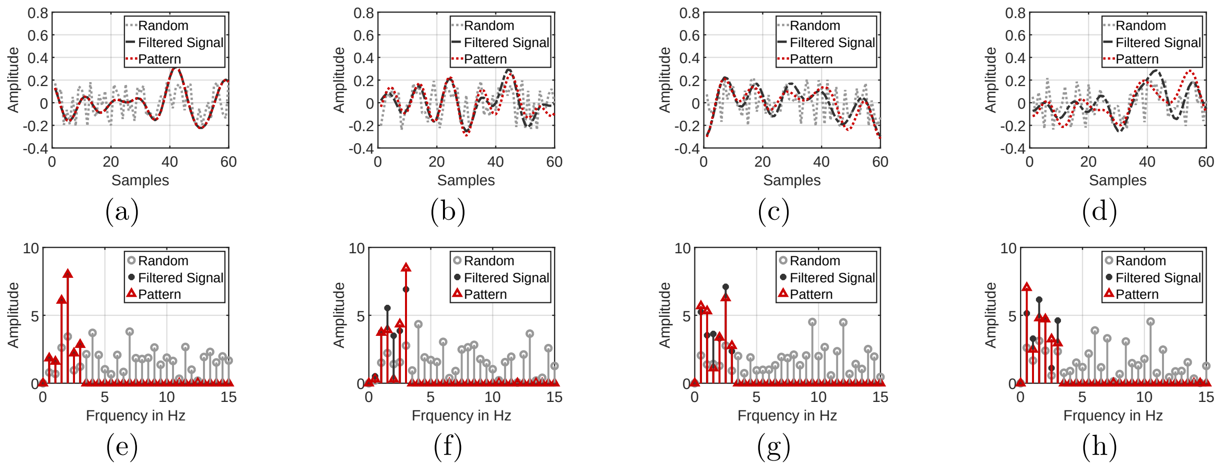


Figure 4.3. An example of a random passcode. The top row shows the four random patterns in the passcode before and after low-pass filtering and the final patterns after applying the Gram-Schmidt process to the filtered pattern. The bottom row shows the FFT of these patterns before and after applying the Gram-Schmidt process. The frequency bound still holds after applying the Gram-Schmidt process.

illuminated simultaneously with a random pixel value, simulating four light sources. Based on this, we define a *light passcode* as a collection of four random light patterns displayed in the four quarters. In the rest of the chapter, we will use *passcode* as a short-term for *light passcode*.

For the passcode, a random light pattern P_{j} is generated for a quarter j. During a time interval t_s, P_{j} is the light pattern represented as a sequence of random numbers, between 0 and 1, of length t_s. The light pattern represents what each pixel of the screen is set to in the quarter j. The screen quarter is white when this value is 1 and black when 0. In order to account for the smartphone screen refreshing rate, we apply an ideal low pass filter with a frequency threshold of $3Hz$ to each of the four light patterns. Although current smartphone screens support a refreshing rate of $60Hz$, there is a delay when the screen is gradually updated from top to bottom. As a result, when the frequency threshold is set to a higher value, the intensity within each quarter may not be consistent. Additionally, setting a higher frequency threshold would result in rapid changes in the screen intensity, making it uncomfortable for users' eyes. These filtered patterns are then normalized such that each pattern is zero-mean.

One problem in illuminating the four quarters together is that the recorded video has a mixture of reflections of the four light patterns from the face. To be able to recover the stereo images from the mixture of reflections, we guarantee independence when combining the light patterns into a passcode. On top of ensuring their independence, we also introduce orthogonality between these four patterns. We apply Gram-Schmidt [141] process to the four light patterns to get their orthogonal basis and use these as patterns. Orthogonality assures a good separation between the impact of the four patterns on the human face and hence helps in the recovery of stereo images. Using induction and the fact that Gram-Schmidt process is linear, we can prove that if each of the original patterns satisfies the frequency threshold of 3Hz, the resulting orthogonal patterns are also within 3Hz. Figure 4.3 shows an example of a passcode with four patterns and the FFT of these patterns before and after the application of Gram-Schmidt process. We can see that the FFT of the patterns generated after applying Gram-Schmidt to the filtered random sequence only has components below $3Hz$. On a side note, the above process is analogous to code-division multiple access (CDMA) [142] used in radio communications. In our case, the face is analogous to the shared media, the camera is the receiver and our orthogonal patterns are like the codes in CDMA. The stereo images generated by each independent quarter are like the data bit sent by each user. The difference is that in our case, we design and use patterns of continuous values that satisfy a frequency bound requirement.

As a result of the above steps, we obtain four orthogonal zero-mean light patterns, forming a passcode. This passcode is then added on top of a constant base intensity value and displayed on the screen. Section 4.5.5 describes how the passcodes are used to defend against replay video attacks.

4.5.2 Video Preprocessing and Filtering

After generating a random passcode, the corresponding light patterns are displayed on the smartphone screen. Meanwhile, a video of their reflections from a user's face is recorded using the front camera. From the recorded video, first, we locate and extract the face in each frame by identifying the facial landmarks (83 landmarks) using Face++ [143]. We then use these landmarks to align the face position in every adjacent frame to neutralize the impact of slight head movements and hand tremors.

Since our following algorithms focus on how the changes in lighting conditions affect the captured face images, we preprocess the recorded video by converting each frame from the color space to the HSV space [144]. Only the V component will be kept and the other two components are discarded since the V component reflects the brightness of an image. Then, each video frame represented by the V component is further processed using Gaussian pyramid [139] for removing noises and optimizing the video size. We use two levels of pyramid and select the peak of the pyramid in the subsequent steps for video analysis, which reduces the system's processing time.

4.5.3 Image Recovery

Recall that in photometric stereo, at least three stereo images with different single light sources are needed for computing the surface normals. However, what we obtained so far is a series of frames, in which the lighting on the face at any given time is a combined effect of all four light patterns on the screen. Therefore, we need to recover these stereo images for each quarter from the preprocessed video frames, which is different from the traditional way of directly collecting stereo images used for photometric stereo.

Based on the theory that the intensities of incoherent lights add linearly [145], we propose to recover the stereo images by directly solving the equation, $G = WX$, where G is a $f \times n$ matrix representing the light intensity values received on each pixel in the recorded video frames, where f is the number of frames and n is the number of pixels in one frame. W represents the $f \times 4$ light patterns $[P_1; P_2; P_3; P_4]$ used while recording the video. X $(= [I_1; I_2; I_3; I_4])$ is a $4 \times n$ matrix representing the four stereo images that we aim to recover. This equation utilizes the fact that under a combined lighting condition, the light intensity received on a certain pixel is a weighted sum of four light intensities with a single light from each quarter.

However, we cannot directly use the above equation unless under the assumption that camera sensors can accurately capture light intensities and reflect the actual values. Problems, e.g. inaccurate image recovery, will arise if we ignore the possible effects of camera parameters and sensitivity. Recently, smartphone camera APIs[1] started supporting manual camera mode which gives the user full control of the exposure parameters, i.e. aperture, shutter speed (exposure time) and sensitivity (ISO). In automatic mode, the camera continuously adjusts its ISO to compensate for lighting changes in the scene. In order to have a smoother camera response with changing light intensity, we use the camera in manual mode where its ISO is set to a fixed value.

Although the camera response curve is smooth after fixing the ISO, we still need to linearize the relationship between the image captured and the light from the screen to be able to use the equation for solving G. For this purpose, we dig deep into the mechanics of the camera sensor and image processing involved in generating the final output images. Cameras typically apply a series of operations on the raw camera sensor data to give us the final output images. These include linearization of sensor data, white balancing, demosaicing [146] and gamma calibration [136]. Gamma calibration is where non-linearity arises between the captured pixel values and the light intensity from the scene. In order to make use of linear relationship between these two, we apply an inverse of the gamma calibration, to the recorded video frames obtained from the camera. As a result, the resulting pixel values in the range between black and saturation level have a linear relationship with the actual light present in the scene. This relationship can be formulated as the linear model, $y = kx + b$, where b is the y-intercept introduced to account for the non-zero black level of the camera sensor. This inverse calibration is applied to each frame in the video preprocessing before face extraction. Now by generalizing the linear model to every frame, containing multiple pixels, we get

$$K = kG + B, \tag{4.4}$$

[1]↑Android supports manual camera mode starting from Android Lollipop 5.1

where K is the video frames that the camera actually captured for the duration of the passcode. By substituting the definition of G into Equation 4.4, we get

$$K = kWX + B. \tag{4.5}$$

Finally, we use the least square method to solve

$$WX = \frac{1}{k}(K - B) \tag{4.6}$$

which can be written as

$$X = (W^T W)^{-1} W^T (\frac{1}{k}(K - B)) \tag{4.7}$$

Here, notice that B is a constant matrix and since each of the four patterns in the passcode W are zero-mean, the term $W^T B$ will be eliminated. Hence Equation 4.7 becomes:

$$X = (W^T W)^{-1} W^T (\frac{1}{k} K) \tag{4.8}$$

Note that this solution X will have an uncertainty of a scale factor. For any $\alpha > 0$, let $X' = \alpha X$, $k' = \frac{1}{\alpha} k$. X', k' will also minimize the above function.

However, this will not have an impact on the reconstructed surface normals. Recall, that surface normals are computed by taking SVD of the stereo images. So, when X and X' are both factorized using SVD, the decompositions are

$$X = U \Sigma V^T, \tag{4.9}$$

$$X' = U(\alpha \Sigma) V^T. \tag{4.10}$$

The surface normal V^T will stay the same in these two cases. From the above observation, we can set $k = 1$ without any impact on the surface normals. Now, we can solve for X' by

$$X' = (W^T W)^{-1} W^T K \tag{4.11}$$

So far, we assumed that the only light present in the scene is due to the passcode displayed on the screen. However, we still need to consider the ambient light present in the scene as well as the base intensity value of the screen on top of which the passcode is added. To account for these other light sources, Equation 4.5 now becomes

$$K = kWX + B + C \tag{4.12}$$

where C is the constant light present in the scene. Again, since C is a constant, because of the orthogonal and zero-mean nature of our passcode, W, W^TC will become 0. As a result, Equation 4.11 will give a solution for X even when ambient light is present.

Due to the inherent delay in the camera hardware, the recorded video may have some extra frames and the timestamps for each video frame captured and the four patterns displayed on the screen at that point may differ. To ensure that we obtain a correct and fine alignment between these two, we first compute the average brightness of each frame and then apply a low pass filter on the average brightness across frames. The peaks and valleys in the average brightness are matched with those of the passcode and finally, DTW [147] is used to align the two series correctly. Once aligned, the result is the video frames which exactly represent the reflection of the passcode from the face. These video frames are then given as input to Equation 4.11 to recover the four stereo images as X. We define the average brightness of these video frames as the recorded passcode for later sections.

An example of the recovered four stereo images corresponding to every single light i.e four patterns displayed in each quarter is shown in Figure 4.4. The top 4 images are the recovered stereo images. The bottom images are the binary representation of these stereo images such that in each image, a pixel value is 1 if the pixel in the corresponding stereo image is larger than the mean value of the same pixel in the other three stereo images. This binary representation is just to visually emphasize how different these stereo images are and how they represent the face illuminated from lighting in four different directions.

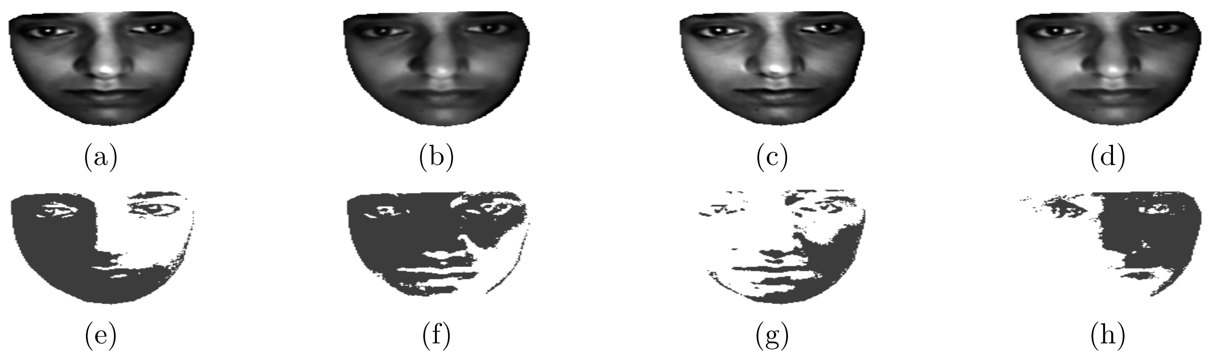


Figure 4.4. The recovered stereo images corresponding to the four patterns in the passcode. The bottom row shows a binary representation to emphasize the differences in these stereo images.

4.5.4 Photometric Stereo and 3D Reconstruction

The stereo images recovered from the least squared method approximate the facial images taken with four different point lights. Now, we can use these stereo images to compute the surface normals of the face as described in Section 4.2.

However, as mentioned earlier, these surface normals have an ambiguity of matrix A. We design an algorithm illustrated in Algorithm 2 to compute the normals without this uncertainty. We use a generalized template, N_t, for the surface normals of a human face and use this to solve for A. This template can be the surface normals of any human face recovered without any ambiguity like surface normals computed when the lighting is known. Note that obtaining this template is a one-time effort and the same normal template is used for all users. Along with the normal map, we also have a $2D$ wired triangulated mesh, M_t, connecting the facial landmarks (vertices), for this template. Now, when computing the normals of a user subject, we use the facial landmarks detected from an RGB image of the face to build a triangulated mesh of the face, M, using M_t as a reference for connecting the vertices and triangles. A representation of this mesh can be seen in Figure 4.5 (left). An affine transformation from the template mesh, M_t to M is then found independently for each corresponding pair of triangles in the two meshes and applied to the matching piece in N_t. As a result, the transformed normal map template, N_t' now fits the face structure of the user.

Algorithm 2 Surface Normal Computation

```
1: procedure SurfaceNormalComputation
2:     Input: normal map template $N_t$, template mesh $M_t$, stacked four stereo images $I$,
   and face RGB image $R$
3:     Output: surface normals $S$
4:
5:     $V \leftarrow$ getFaceLandmarks($R$)
6:     $M \leftarrow$ buildMesh($V, M_t$)
7:     $\hat{S}, \hat{L}^T \leftarrow$ SVD($I$)
8:     $N_t' \leftarrow$ transform($N_t, M_t, M$)
9:     Solve $N_t' = A\hat{S}$ for $A$
10:    $S' \leftarrow A\hat{S}$
11:    $M_s \leftarrow$ symmetrizeMesh($M$)
12:    $S \leftarrow$ transform($S, M, M_s$)
13:    $S \leftarrow$ adjustNormalValues($S$)
14:
15:    function transform($Z, T_1, T_2$)
16:        for each pair of triangles $< t_1, t_2 > \in T_1, T_2$ do
17:            $a \leftarrow$ affineTransformation($t_1, t_2$)
18:            $Z_{\text{out}} \leftarrow$ warp($Z(t_1), a$)
19:            $Z(t_2) \leftarrow Z_{\text{out}}$
20:        end for
21:    end function
22: end procedure
```

This transformed template can finally be used to find the unknown A, by solving $N_t' = A\hat{S}$ where $\hat{S}$ are the approximate normals recovered from SVD, and obtain the surface normals, S'. The last step in normal map computation is to make the normal map symmetric. This is needed to reduce noise in the recovered stereo images and hence the surface normals. We first find the center axis of the $2D$ face mesh using landmarks on the face contour, nose tip and mouth. Once the center is found, each pairing landmarks like eyes, eyebrow corner etc. are adjusted such that they have equal distance to the center to get a symmetric mesh. After symmetrizing the mesh, we fit S into this symmetrized mesh. Now, we can easily apply inverse symmetry to the x component of S and symmetrize the values in y and z components of S. Note that by introducing symmetry, we might loose some tiny details of the facial features as all human faces are not symmetrical. However, since our goal is to distinguish the human face from their spoofing counterpart and not another human, the information retained in the surface normals is more than sufficient. Figure 4.5 (right) shows an example of the x, y and z components of a normal map generated from our algorithm.

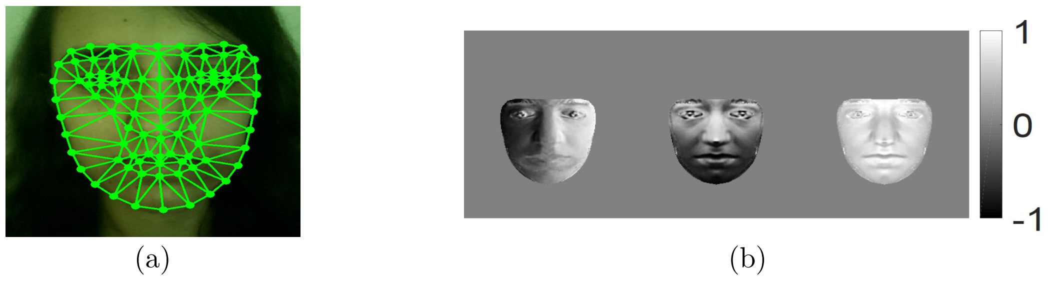


Figure 4.5. Normal map calculation (left) shows $2D$ triangulated face mesh generated by using facial landmarks. (right) shows the X, Y, and Z components of the normal map generated from Algorithm 2.

After we have successfully recovered the surface normals, we can reconstruct the 3D surface of the face from them. For 3D reconstruction, we follow the quadratic normal integration approach described in [135]. The results of 3D reconstruction are shown in Figure 4.6. Side and top view are shown for each reconstructed model.

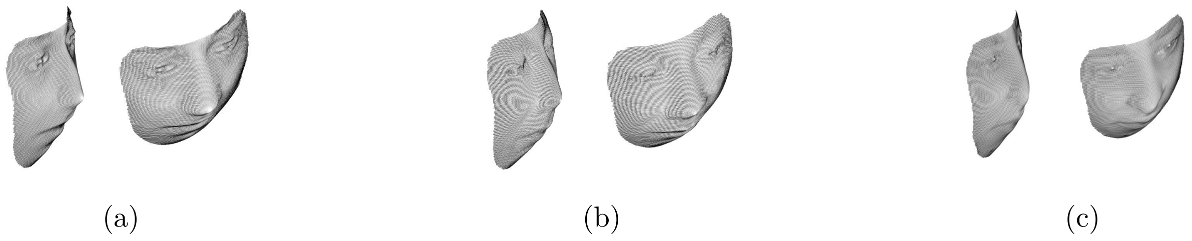


Figure 4.6. Examples of 3D reconstruction from human faces. Side and top views are shown.

4.5.5 Liveness Detection

FaceRevelio aims to provide security against two broad categories of spoofing attacks: 2D printed photograph attack and video replay attack.

2D Printed Photograph Attack: To defend against the 2D printed photograph attacks, we need to determine whether the reconstructed 3D face belongs to a real/live person or a printed photograph. Figure 4.7 shows examples of 3D reconstruction from a printed

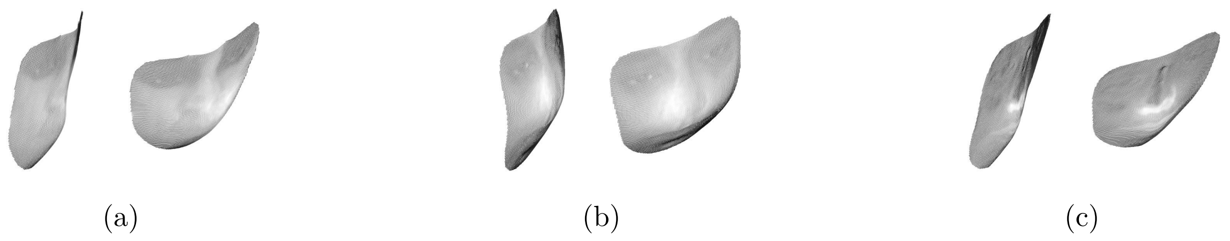


Figure 4.7. Examples of 3D reconstruction from 2D printed photographs. Side and top views are shown.

photograph using the approach described in the previous section. It is interesting to note here that the same general human face normal map template is used for computing the surface normals of a photograph. As a result, the overall structure of the reconstructed model looks similar to a human face. However, even when using this human normal map template, the freedom provided by solving for A is only upto 9 dimensions. Therefore, despite having a similar structure, the reconstruction from the 2D photograph lacks depth details in facial features, e.g. nose, mouth and eyes, as is clear in the examples in Figure 4.7.

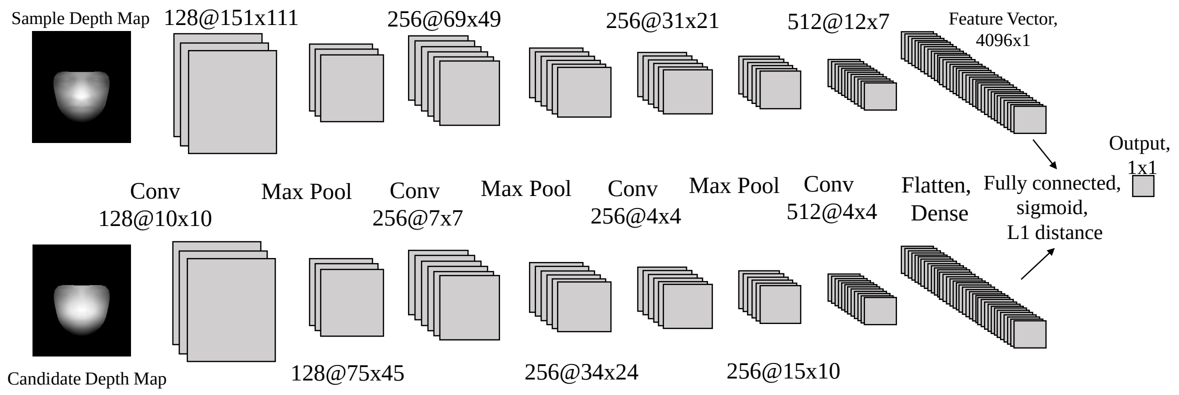
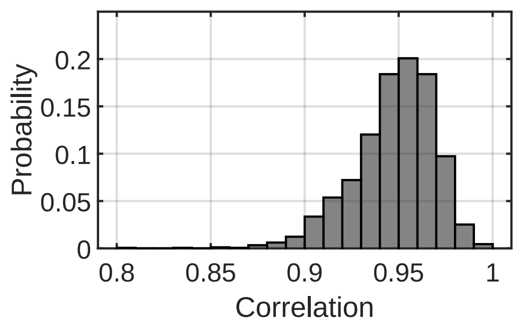


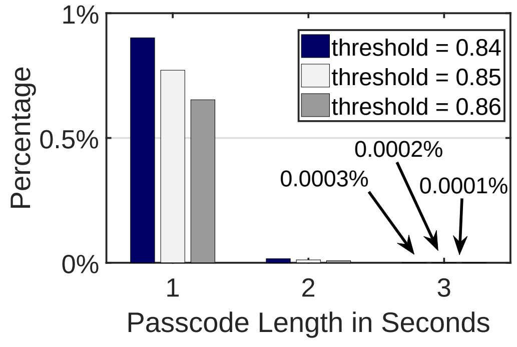
Figure 4.8. Architecture of the Siamese neural network. One of the twin neural networks takes a known human depth map as input while the other is passed the candidate 3D reconstruction.

Based on these observations, we employ a deep neural network to extract facial depth features from the 3D reconstruction and classify it as a human face or a spoofing attempt. We train a Siamese neural network adapted from [140] for this purpose. The Siamese network consists of two parallel neural networks whose architecture is the same, however, their inputs are different. One of these networks takes in a known depth map of a human face while the other is given the candidate depth map obtained after the 3D reconstruction. Both these networks output a feature vector for their inputs. These feature vectors are then compared using L1 distance and a sigmoid activation function. The final output of the Siamese network is the probability of the candidate depth map being that of a real human face. Figure 4.8 shows the architecture of the Siamese network. Every time a subject tries to authenticate using *FaceRevelio*, the reconstructed $3D$ model along with a sample human depth map is fed as input to the Siamese network. If the output of the Siamese network is above a threshold τ_s, the system detects a real face. Otherwise, a spoofing attempt is identified.

Since Siamese network uses the concept of one-shot learning [148] and takes pairs as input for training, the amount of data required for training is much smaller than traditional convolutional neural networks. Here, one may argue that why not train the model with the raw images/videos captured by the front camera, for the duration of passcode, instead of the depth map? Training a model with these raw images would significantly increase the storage and computation costs of our system and would require huge amounts of training data for the different ambient environments.



(a)



(b)

Figure 4.9. Video Replay Attack: (left) shows the distribution of correlation between recorded passcodes from human face and the original passcode. (right) shows the percentage of passcodes which have a correlation with another random passcode higher than a threshold for different thresholds.

Video Replay Attacks: *FaceRevelio* has a two-fold approach for defending against video replay attacks. The first line of defense is to utilize the randomness of the passcode. When a human subject tries to authenticate via *FaceRevelio*, the passcode displayed on the screen is reflected from the face and captured by the camera. As a result, the average brightness of the video frames across time has a high correlation with the light incident upon the face i.e. the sum of the four patterns in the passcode displayed on the screen. Figure 4.9 (left) shows a distribution of the correlation between recorded passcodes and the original passcode for experiments conducted with humans. The correlation between the two passcodes is higher than 0.85 for more than 99.9% of the cases. An adversary may try to spoof our system by recording a video of a genuine user while using *FaceRevelio* and replay this video on a laptop screen or monitor in front of the phone later. In this case, the video frames captured by the camera will have the reflections of the passcode on the phone screen as well as the passcode present in the replay video. Since *FaceRevelio* chooses a totally random passcode each time as described in 4.5.1, the probability that the passcode displayed on the screen and the passcode in the video has a high correlation is extremely low. To give an idea, for a passcode duration of 3s, if we compare 300 million pairs of random passcodes, only 0.0003% of the pairs will have a correlation greater than 0.84. Figure 4.9 (right) shows the percentage of passcode with a correlation higher than threshold values 0.84, 0.85 and 0.86 for passcode lengths of 1, 2 and 3s. Hence, just by computing and setting a threshold on the correlation between the recorded passcode and the sum of passcode from the screen, the chances of detecting a replay attack are very high.

For the rare cases when the correlation is higher than the predefined threshold, our second line of defense comes into play. Similar to 2D photograph attack, video replay attacks can also be detected using the reconstructed 3D model. The reconstruction from the replayed video suffers from two main problems. First, it is hard for the adversary to accurately synchronize playing the attack video with the start of the passcode display on the smartphones. Second, even if the correlation passes the threshold, there will be some differences in the replayed passcode and *FaceRevelio*'s passcode. Because of this, the DTW matching will not match the recorded video frames with the displayed passcode very well, resulting in wrong stereo image recovery. The incorrect 3D reconstruction from these wrong stereo images is sufficient to identify a spoofing attempt.

4.6 Evaluation

We describe the implementation and evaluation of our system in this section. We first describe the experiment settings and the data collection details and then the performance of our system in different settings.

4.6.1 Implementation and Data Collection

We implemented a prototype for *FaceRevelio* on Samsung S10 which runs Android 9, with 10 MP front camera that supports Camera2 API. The videos collected for our authentication system have a resolution of 1280x960 and a frame rate of 30fps. For each experiment setting, we display the passcode patterns on the smartphone screen and record a video of the reflections from the user's face via the front camera. We use Face++ [143] for landmark detection and OpenCV in the image recovery and reconstruction modules of our system. Python libraries for TensorFlow [149] and Keras were used to train the neural network for liveness detection while TensorFlow Lite was used for inference on Android.

We evaluated *FaceRevelio* with 30 volunteers using our system for liveness detection. The volunteers included 19 males and 11 females with ages ranging from 18 to 60. These volunteers belonged to different ethnic backgrounds including Americans, Asians, Europeans and Africans. During the experiments, the volunteers were asked to hold the phone in front of their faces and press a button on the screen to start the liveness detection process. Once the button was clicked, the front camera started recording a video for the duration of the passcode. During all experiments, we collected a high-quality image of the user to test the performance of our system against photo attacks. For the video replay attack, we used the videos collected from real users and replayed them to the system.

We collected a total of 3800 videos from the 30 volunteers over a duration of 2 weeks. We evaluated the performance of our system in natural daylight as well as in dark. For the daylight setting, all experiments were conducted during daytime however the lighting varied based on the weather conditions on the day and time of the experiment. Each volunteer performed 10 trials of liveness detection using our system for each of the two light settings. A random passcode of 1s duration was added on top of a gray background (grayscale intensity value of 128) for these trials. We also tested *FaceRevelio* with passcode durations of 2 and 3s in the two light settings. We also evaluated the impact of indoor lighting and the use of a background image on the performance of our system. For these scenarios, we collected data from 10 volunteers with a passcode duration of 1s. These volunteers also used the system 10 times for each scenario.

We used the Siamese neural network described in section 4.5.5 to test a user by using the depth maps generated from data collected from the remaining users for training.

4.6.2 Performance Results

For evaluating *FaceRevelio* system performance, we answer the following questions:

(1) What is the overall performance of *FaceRevelio*?

To determine the overall performance of our system, we evaluated our system's ability to defend against 2D printed photographs and video replay attacks. We report the accuracy of our system as the true and false accept rate for the two light settings. We also determine the equal error rate (EER) for the attacks.

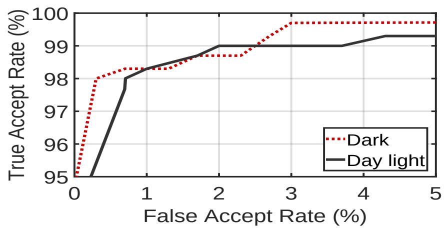


Figure 4.10. ROC curve for detecting photo attack in dark and daylight setting with a passcode of 1s. The detection rate is 99.7 and 99.3% when true accept rate is 98% and 97.7% for the two settings respectively.

First, we describe our system's performance against printed photograph attack. Figure 4.10 shows the ROC curve for *FaceRevelio*'s defense against photo attack in the dark and daylight setting with a passcode duration of 1s. For dark setting, with a true accept rate of 98%, the false accept rate is only 0.33%. This means that a photo attack is detected with an accuracy of 99.7% when the real user is rejected in 2% of the trials. The EER for the dark setting is 1.4%. In daylight, the photo attack is detected with an accuracy of 99.3% when the true accept rate is 97.7%. The EER in this case is also 1.4%. *FaceRevelio* performs better in dark setting because the impact of our light passcode is stronger when the ambient lighting is weaker. Hence, the signal-to-noise ratio in the recorded reflections from the face is higher, resulting in a better 3D reconstruction.

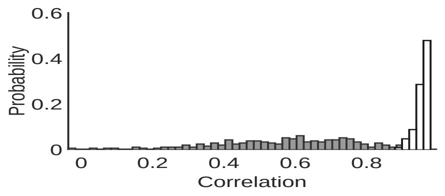


Figure 4.11. Distribution of the correlation between the passcode on the phone and the camera response from real human and video attack combined for dark and dayllight setting.

We also evaluated our system against video replay attacks by using videos collected from the volunteers during the experiments. Each video was played on a Lenovo Thinkpad laptop, with a screen resolution of 1920 x 1080, in front of a Samsung S10 with *FaceRevelio* installed. Our system detected these video replay attacks with an EER of 0% in dark and 0.3% in daylight settings. Figure 4.11 shows a histogram of the correlation between the passcode displayed on the phone and the camera response for all experiments with 1s long passcode. The correlation for all the attack videos is less than 0.9. In contrast, 99.8% of the videos from real human users have a correlation higher than 0.9.

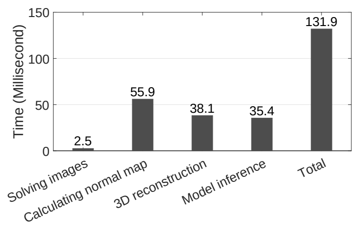


Figure 4.12. Processing time of the different modules of the system for a passcode of 1s duration.

Another performance metric is the total time it takes to detect liveness with *FaceRevelio*. Figure 4.12 shows the processing time of the different modules of our system. On top of the signal duration of the passcode, the liveness detection process only takes 0.13s in total. The stereo images recovery only takes 3.6ms. The most expensive computation step is the normal map computation, taking 56ms, since it involves two 2D warping transformations.

3D reconstruction and feature extraction and comparison via the Siamese network take 38.1 and 35.4 ms respectively.

(2) What is the effect of the duration of the light passcode?

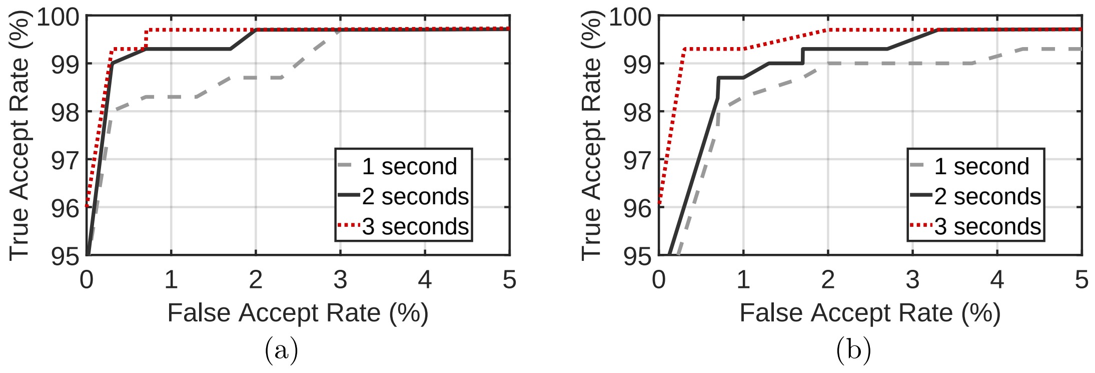
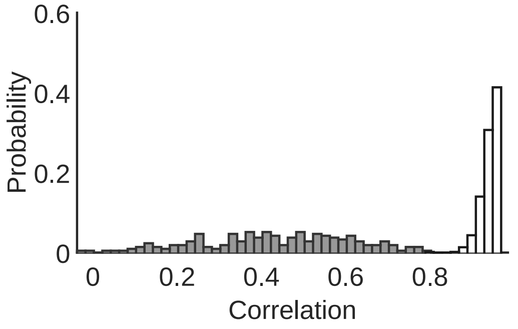


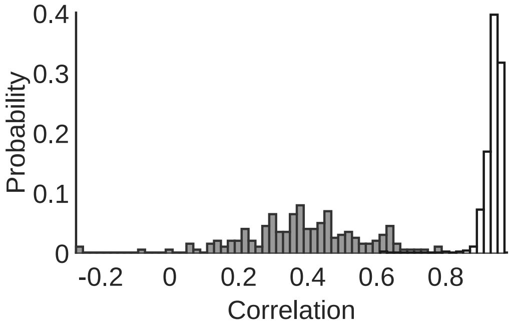
Figure 4.13. ROC curve for passcode durations of 1, 2, and 3 seconds in dark (left) and daylight (right) settings.

To answer this question, we tested the performance using passcodes of time durations 1, 2 and 3s. Figure 4.13 shows the ROC curve for photo attack with different passcode duration in dark (left) and daylight (right) settings. In dark, the attacks are detected with an accuracy of 99.7% for passcodes of length 1, 2 and 3 seconds each. These accuracies are achieved when the true accept rate is 98%, 99% and 99.3% for the three time durations respectively. The EER is 1.44% for 1s and 0.7% for 2 and 3 each. For daylight, the detection accuracy is 99.3% for 1s and 2s. For 3s, the photograph attack is detected with an accuracy of 99.7%. These accuracies are achieved when the true accept rate is 97.7%, 98.3% and 99.3% for 1, 2 and 3s respectively. We observe that the performance of *FaceRevelio* improves as we increase the duration of the passcode. Although the true accept rate deteriorates when a passcode of 1s is used, achieving a higher attack detection accuracy within a short duration is the priority of an effective liveness detection system.

We also evaluated the effect of passcode duration on detecting video attacks. Figure 4.14 shows the correlation distribution for human and video attack combined for passcode duration of 2 (Figure 4.14 left) and 3 (Figure 4.14 right) seconds in the two light settings. For 2s, all the video attacks have a correlation less than 0.84 while 99.8% of the human data have



(a)



(b)

Figure 4.14. Distribution of the correlation between passcode on the phone and the camera response from real human and video attack for $2s$ (left) and $3s$ (right) long passcodes.

a correlation higher than 0.86. In case of $3s$, 99.8% of the real human experiments have a correlation higher than 0.8. In comparison, all attack videos have correlation of less than 0.8.

We also determine the effect of the passcode duration on the processing time in the authentication phase. The duration of the passcode only affects the time taken to determine the least squared solution for recovering the four stereo images as that depends on the number of frames in the recorded video. The computation time for the other components of the system stays consistent across different passcode duration. The total processing time remains below $0.15s$ for all three passcode durations.

(3) How well does *FaceRevelio* perform in indoor lighting?

To evaluate the effect of indoor lighting, we conducted experiments with 10 volunteers in a room with multiple lights on. The goal was to determine if this extra light had any impact on the efficacy of our light passcode. In these experiments, we used $1s$ long passcodes. For a true accept rate of 98%, *FaceRevelio*'s accuracy against 2D attacks is 99.7%. It achieves an EER of 1.4% which is comparable to the dark setting. Hence, we conclude that *FaceRevelio* performs well even when artificial light is present in the scene.

(4) What is the effect of displaying the signal on a background image?

So far, we used gray image as a base for the light passcode displayed on the screen to evaluate our system. Here we want to determine how the system performance change if we used an RGB image for the passcode instead of the gray background. For this purpose, we selected a total of 5 background images (shown in Figure 4.15(top)). Figure 4.15 also shows an example of what the passcode frames look like with an image background across time. We performed experiments with 10 users where each user performed 10 trials in daylight setting using the 5 background images. Our system achieves an EER of 1.15% against the spoofing attacks. A photo attack is detected with an accuracy of 99.4% when the true accept rate for humans is 97%. These results show that *FaceRevelio*'s process can be made more user friendly by using images of the user's choice as a base for the passcode.

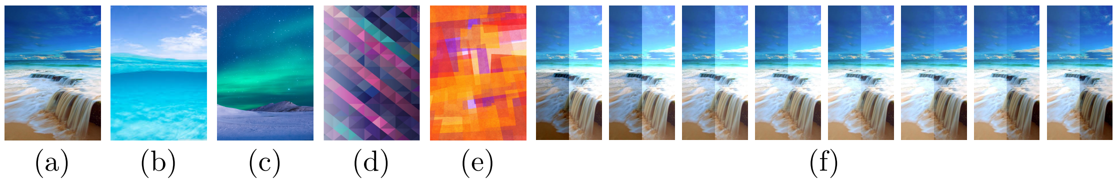


Figure 4.15. Top row shows images chosen as background for the light passcode. Bottom row shows what the passcode looks like with an image as background

Table 4.1. Summary of existing face liveness detection methods

Algorithm	Attack Resistance	Special Hardware?	User Interaction Required?	Limitation	Accuracy
FaceID [127]	2D & 3D	TrueDepth	No	3D head mask attack	$> 99.9\%$
Samsung FR [150]	None	No	No	Photo Attack	-
EchoFace [151]	2D photo	No	No	Audible sound	96%
FaceCloseup [152]	2D photo/video	No	Requires moving the phone	Slow response	99.48%
EchoPrint [153]	2D photo/video	No	No	Audible sound, low accuracy in low illumination	93.75%
Face Flashing [9]	2D photo/video	No	Requires expression	Slow response	98.8%
FaceHeart [154]	2D photo/video	No	Place fingertip on back camera	Low accuracy in low illumination	EER 5.98%
FaceLive [133]	2D photo/video	No	Requires moving the phone	Slow, low accuracy in low illumination	EER 4.7%
Patel et al. [155]	2D photo/video	No	No	Device dependent, low accuracy in low illumination	96%
Chen et al. [130]	2D photo/video	No	Requires moving the phone	Slow response	97%

(5) Where does *FaceRevelio* stand compared to existing face liveness detection methods?

Table 4.1 gives an overview of the existing methods for face liveness detection on smartphones. It shows the type of attacks these methods can defend against and if they require any extra hardware or user interaction for doing so. Among the commercial solutions, Samsung's face recognition is vulnerable to simple 2D photo attacks and needs to be combined with other authentication methods for security [150]. Apple's *FaceID* [127] is the most secure method against 2D and 3D spoofing attacks, owing to the *TrueDepth* camera [127] used for 3D reconstruction. However, among methods that do not rely on any extra specialized hardware [9, 133, 152, 154], *FaceRevelio* achieves the highest accuracy in detecting 2D photo and video attacks with the fastest response time of $1s$. Tang et al. [9] use a challenge-response protocol to achieve a high detection accuracy, however, their approach relies on the user to make facial expressions as instructed and takes $6s$ or more (depending on the number of video frames collected) to perform well. In contrast, *FaceRevelio* detects the spoofing attempts in $1s$, without requiring any user interaction, increasing its overall usability. Another important comparison metric is the performance variation in different lighting conditions. For methods like [153–155], the performance mentioned in table 4.1 is achieved under controlled lighting conditions and deteriorates in dark environments. EchoFace [151] achieves a good accuracy by using an acoustic sensor based approach however their sound frequency is within human audible range, (owing to smartphones' speaker limitation [156]) making it less user friendly.

4.7 Related Work

Several software-based face liveness detection techniques have been proposed in the literature. These depend on features and information extracted from face images captured without additional hardware. Texture-based methods detect the difference in texture between real face and photographs/screens. In [132], local binary patterns were used to detect the difference in local information of a real face and a 2D image using binary classification. Another technique, [157], measures the diffusion speed of the environmental light which helps distinguish a real face. [154] operates by comparing photoplethysmograms independently extracted from the face and fingertip videos captured by front and back cameras. Similarly [158] uses a combination of rPPG and texture features for spoof detection. These works do not perform well in poor lighting conditions and are affected by the phone camera limitations. Some works [155, 159], make use of the degraded image quality of attack photos or videos. However, with modern cameras and editing softwares, an adversary can easily obtain high quality images and videos to conduct an attack. In contrast to these approaches, *FaceRevelio* works in different lighting conditions and is not dependent on the quality of the videos captured.

Other techniques use the involuntary human actions such as eye blinking [129] or lips movement [160] to detect spoofing, but these techniques fail against video replay attacks. Challenge-response protocols require the user to respond to a random challenge, such as blinking, face expression, head gesture, etc [161]. These systems are limited by their unconstrained response time and are still prone to replay attacks. Another work, [9], used a time constrained challenge-response technique that shows different colors on the screen and detects the difference in the time of reflection between a real face and an attack. This work differs from *FaceRevelio* as they utilize the random challenge on the screen to perform a timing verification whereas we use the screen lighting to reconstruct the 3D surface of the face. Also, [9] requires the user to make a face expression to defend against static attacks. Some works like [130, 133] require the user to move the phone in front of their face and analyze the consistency between the motion sensors' data and the recorded video to detect liveness. These approaches require some form of user interaction unlike our system which operates independently of the user.

Some hardware-based techniques require extra hardware or different sensors to detect more features of the human face structure. FaceID was introduced by Apple in the iPhone X to provide secure 3D face authentication using a depth sensor [127]. However, the extra hardware consumes screen space and requires additional cost. [153] developed an authentication system that uses the microphone with the front camera to capture the 3D features of the face. However, this technique does not work well in poor lighting and depends on deep learning which requires large training datasets. Similarly, [151] uses acoustic sensors to detect the 3D facial structure. Both these techniques play audible sound for detection, which makes their system less user friendly. Some other techniques use thermal camera [162], 3D-camera or multiple 2D-cameras [163]. Again, these techniques suffer from the setup cost for these extra devices.

4.8 Discussion

FaceRevelio depends on light emitted from the screen, therefore it is sensitive to rapid changes in the ambient lighting like when a user is in a moving car. The accuracy of our system would be affected in such scenarios. This requires investigating other camera features to recognize the small light changes produced by our passcode in the presence of strong, changing ambient light.

Recently, some advanced machine learning based attacks [164, 165] have been successful in spoofing state-of-the-art face recognition systems. However, *FaceRevelio* can defend against these attacks because the random light passcode changes with every use of our system and does not have any relation to the passcodes used previously. Hence, learning a machine learning model to guess the password on the fly and replaying it to the system is not possible. Here, we admit that *FaceRevelio* can be spoofed by using a 3D printed mask of the subject, however, such attacks are costly and much more difficult to execute than existing attacks.

In our system, we divided the phone screen into four quarters for displaying four random patterns in the passcode. These passcodes helped us achieve a good accuracy in detecting replay attacks. However, we can further push the randomness involved in our passcodes by dividing the screen into smaller regions or using combination of different shapes to display the light patterns. We also plan to increase the system usability by using more sophisticated light patterns, such as a picture of blinking stars or animated waterfall.

FaceRevelio provides a promising solid idea for secure liveness detection without any extra hardware. Our technique can be integrated with existing 2D face recognition technologies on smartphones. Detecting the 3D surface of the face through our system before face recognition would help them in identifying spoofing attacks at an early stage. This will improve the overall accuracy of these state-of-the-art technologies. Apart from this, our system also has the potential to be used for 3D face authentication directly.

5. ONE KEY TO RULE THEM ALL: SECURE GROUP PAIRING FOR HETEROGENEOUS IOT DEVICES

5.1 Introduction

Internet of Things (IoT) devices need secure wireless communication channels to protect the confidentiality and integrity of the data they exchange (e.g., sensor measurements, actuator states). This protection is critical to prevent various attacks (e.g., man-in-the-middle (MitM) [13, 166] and protocol manipulation [167, 168]), provide user privacy and ensure the trustworthiness of IoT systems [10, 15]. Therefore, IoT devices require a *pairing* mechanism, which establishes shared cryptographic keys between devices to enable secure wireless communication among them.

Traditional pairing methods employ a centralized approach where a user pairs each device with a trusted IoT gateway/hub through an external helper device (e.g., user typing a password on their smartphone to pair a smart light with the IoT hub). Yet, central gateways/hubs (a) are prone to temporary or permanent failures due to operational malfunctions [169], and (b) can be compromised due to their vulnerabilities [168, 170]. In such cases, devices need a secure mechanism for directly communicating with each other.

To illustrate, consider a smart home that turns on the lights and unlocks the door when smoke is detected for fire safety. If the hub is not present or experiences a failure, these devices cannot communicate with each other, resulting in severe consequences, e.g., residents being trapped in a fire.

Consequently, IoT platforms have recently been pushing towards decentralized IoT networking protocols (e.g., OpenThread [171]). Such decentralized protocols have applications in smart homes and industrial automation due to their reliability (always-on), scalability (easy device addition), and adaptability (supporting devices from different vendors). Past efforts at decentralized secure IoT device pairing have explored two primary approaches: (1) human-in-the-loop-based and (2) context-based pairing.

In human-in-the-loop-based approaches, a user needs to be physically involved to facilitate the pairing process. A line of work requires users to contact devices by exploiting the fact that the movement pattern is correlated in multiple devices [172, 173]. For instance, a user with a wristband touches a device [12] or shakes two devices at the same time for pairing [11]. Another line of work relies on the user to enter passwords, read QR codes, or press buttons [174]. For example, OpenThread requires a user to scan the QR code of each device with a mobile phone [171]. These approaches, however, require human involvement that affects usability and scalability with an increasing number of devices.

To address these limitations, there is an increasing interest in context-based pairing schemes [13]. In this, co-located sensors establish shared keys based on the entropy extracted when they observe common events. Yet, these approaches are limited to pairing devices only equipped with homogeneous sensors that sense identical sensing modalities [14, 15, 175, 176]. For instance, to pair a power meter with a microphone, another microphone must be embedded into the power meter so that both devices capture a common audio context.

Recent work has proposed capturing event timings among heterogeneous devices as evidence for device co-presence to derive secure keys [10]. While this approach supports heterogeneous sensing types without human intervention, it suffers from four fundamental limitations: (1) takes several hours or a few days for pairing, (2) is limited to sensors that only sense instant physical quantities (e.g., cannot pair widely deployed temperature and humidity sensors), (3) entire pairing process is impaired by concurrent events (e.g., when the sound from door lock and coffee machine overlaps), and (4) can pair solely two sensors at a time. These limit its usability, make it infeasible for pairing diverse device types and ultimately fail in offering promise for adoption to a wide-variety of IoT deployments in practice.

In this chapter, we design and develop IoTCupid, a secure group pairing system for IoT deployments with heterogeneous sensing modalities. IoTCupid complements trusted gateways in IoT deployments when they experience operation failures or are compromised and enables secure communication among devices. It operates both on instantly and continuously influenced sensors, supports concurrent events for context extraction, and establishes a secure shared group key among devices that sense the same events.

IoTCupid first obtains the sensor measurements corresponding to events sensed by each device. It implements a feature processing technique through a window-based derivation algorithm to support sensors that measure instant (e.g., sound) and continuous physical quantities (e.g., temperature and humidity). It then derives temporal sensor features from detected events, extends a fuzzy clustering algorithm to group concurrent and independent events into different types, and obtains inter-event timings (time interval between consecutive events) for each event type. Lastly, it uses the inter-event timings as evidence to authenticate the devices and establishes a shared group key among all devices that sense the same events. IoTCupid's group key establishment protocol enables dynamic group generation and is resilient to MitM, offline brute-force and denial of key exchange attacks. In contrast to previous approaches, IoTCupid provides a fast, practical, and secure group pairing scheme that automates pairing diverse device types with no active user involvement and a minimal computational and storage cost.

We present two studies evaluating IoTCupid's effectiveness. In a first study, we conducted experiments in a smart home with 4 sensors and 4 event sources while two residents were conducting their routine activities. We also deployed devices outside the home to evaluate IoTCupid's resilience against attacker devices that aim to pair with legitimate devices. To demonstrate IoTCupid's practicality in different environmental conditions, in a second study, we evaluated IoTCupid on a dataset [177, 178] collected in an office environment with multiple devices and event sources while four people were conducting their everyday work. IoTCupid correctly detected events with an average precision and recall rate of 95.8% and 83%, and successfully paired all devices with only four equivalent inter-event timings. We show that the attacker devices cannot pair with legitimate devices using IoTCupid. These studies demonstrate IoTCupid establishes group keys without a significant overhead. It takes 39 milliseconds on average to derive group keys among 20 devices with 4 event sources, and the overhead increases linearly with an increasing number of event sources and devices. In summary, we make the following contributions:

- We introduce IoTCupid, a secure and practical group pairing system for heterogeneous sensors. IoTCupid leverages inter-event timings of commonly sensed events to pair devices with diverse sensing modalities in a short duration with minimal computation and storage cost.

- We design a dynamic group key establishment protocol that extends a partitioned group password-authenticated key exchange scheme with provable security.

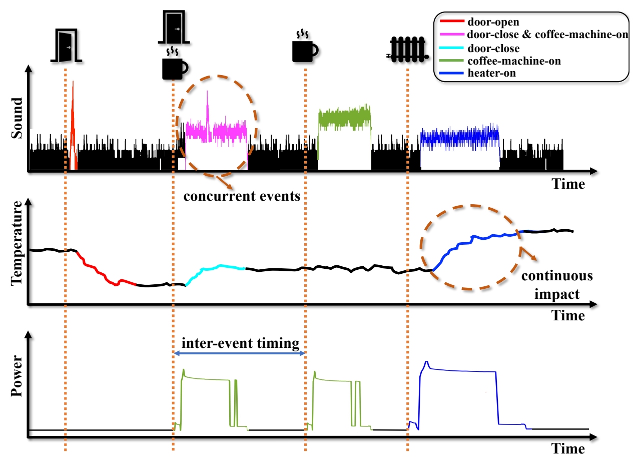


Figure 5.1. An illustration of how common events sensed by different sensors can be used for device pairing.

- We perform two studies in a smart home and smart office to show IoTCupid's effectiveness and performance in pairing multiple devices with diverse modalities.

5.2 Problem Statement

Trusted gateways may not always be available in IoT deployments. In such cases, secure communication channels are constructed through decentralized pairing systems that authenticate the devices and establish cryptographic keys for device-to-device communication. Most of these systems require human involvement (e.g., simultaneously shaking two devices) to pair the devices. However, such solutions have limited usability and are infeasible for IoT deployments with the increasing number of devices. Therefore, recent efforts have explored context-based pairing to provide secure communication channels [14, 15, 175, 176]. These approaches, however, can only pair devices equipped with homogeneous sensors that sense identical sensing modalities.

Since IoT devices are equipped with sensors with different capabilities and sensing modalities, our goal is to design a secure, and practical pairing system for these heterogeneous devices based on their shared context, without user involvement. To illustrate, we consider an IoT deployment with three devices. Each device is equipped with one of the following sensors: a

Table 5.1. Commonly occurring events in IoT environments and the sensors impacted by these events.

Event	Sensors Impacted
`door-open/close`	air pressure, humidity, illuminance, microphone, motion, temperature
`coffee-machine-on/off`	microphone, power
`window-open/close`	air pressure, humidity, illuminance, motion, temperature
`oven-on/off`	humidity, power, temperature
`light-on/off`	illuminance, power
`AC-on/off`	air pressure, humidity, microphone, power, temperature
`heater-on/off`	humidity, microphone, power, temperature
`TV-on/off`	illuminance, microphone, power
`dryer-on/off`	humidity, microphone, power, temperature

microphone, a power meter, and a temperature sensor. In the morning, `user-A` opens the door (`door-open`) to go out. Meanwhile, `user-B` turns on the coffee machine (`coffee-machine-on`). While the coffee machine is on, `user-A` returns and closes the door (`door-close`). `User-A` prepares a cup of coffee for herself (`coffee-machine-on`) and turns on the heater (`heater-on`).

Figure 5.1 presents the physical influences of the described event sequence on the sensors, where all devices perceive common events. For instance, the `door-open/close` events influence the microphone and temperature sensor while the microphone and power meter sense the `coffee-machine-on` event. Similarly, `heater-on` influences all three sensors.

In general, commonly occurring events in typical IoT environments are sensed by multiple devices equipped with heterogeneous sensors [10, 177, 179–181]. Table 5.1 summarizes commonly occurring events and the group of sensors impacted by these events in a typical smart home setting. Although the devices' measurements are not directly comparable due to different signal characteristics and their times may not be synchronized, these devices can leverage the inter-event timings to verify they observed the same event. Particularly, two or more devices can use the time interval between the subsequent occurrences of a commonly observed event type (e.g., `coffee-machine-on` events sensed by the microphone and power meter) as a proof of co-presence and use them as an evidence to establish a symmetric key.

Definitions. In this chapter, we use the term *events* to refer to instances of changes in the device states (e.g., `door-open`, `coffee-machine-on` and `heater-on`). Events influence a set of physical channels that are measured by sensors. We use the term *signals* to refer to the processed sensor readings that represent the influence of an event separated from the background noise. Common devices in IoT environments are influenced by multiple *event types*. For instance, a device equipped by a temperature sensor may be influenced by events of types `door-open`, `heater-on` and `AC-on`.

5.2.1 Design Requirements and Challenges

Several schemes leveraged shared homogeneous context to securely pair IoT devices [14, 15, 175, 176], and only a single prior work [10] explored using inter-event timings for the secure pairing of heterogeneous sensing devices. However, these schemes suffer from four primary problems, which makes them impractical in many IoT environments. We detail them below and address each with IoTCupid.

Short Pairing Time. Existing works use cryptographic primitives that are vulnerable to offline brute-force attacks. In these attacks, the attacker enumerates all possible inter-event timings to derive the cryptographic keys. Thus, they require many inter-event timings to provide sufficient entropy for the shared keys to be cryptographically secure. The time needed to derive secure keys is further increased due to concurrent events, which cannot be used for pairing as they create inter-event timing mismatches (Detailed below).

Pairing Continuously Influenced Sensors. Prior pairing systems only consider sensors that are instantly influenced by an event. Yet, in real environments, many sensors measure continuous physical quantities (e.g., temperature and humidity). We refer to such sensors as *continuously influenced* sensors. For instance, in Figure 5.1, the `heater-on` event's sound instantly influences the microphone. However, the temperature continuously increases over time. Unless event detection considers gradual changes in the temperature, the `heater-on` event cannot be detected and used to pair the microphone and temperature sensor. Without considering continuously influenced sensors, the number and types of devices which can be paired are very limited.

Concurrent Events. Existing pairing schemes mainly evaluate scenarios where a device only senses a single event per time period. However, in practical scenarios, multiple events occur simultaneously and produce an overlapping influence on the sensors. In Figure 5.1, `coffee-machine-on` and `door-close` happen concurrently, and the microphone measures their aggregated influence. Prior works group concurrent events into a new separate event type. This leads to longer pairing times since the inter-event timings for a device sensing concurrent events will not match with another device sensing only one of those event types. To address this, a method to separate independent and concurrent events into groups is needed for scalable and practical pairing.

Deriving Group Keys. In centralized IoT deployments, a trusted hub monitors and controls the devices. However, in decentralized IoT protocols, devices need to broadcast their states to invoke their event-triggered automations. For instance, a smoke-detector broadcasts the `smoke-detected` event; upon receiving, the lights are turned on and the door is unlocked for user safety. Previous decentralized pairing schemes establish individual keys among pairs of devices. When a device needs to broadcast a message, it individually encrypts it with all shared keys and then sends the ciphertexts, causing linear computation, communication and energy consumption overhead. Since IoT devices typically have limited resources and energy constraints, this overhead negatively impacts their performance and battery.

5.2.2 Threat Model

The attacker, $\mathcal{A}$, aims to eavesdrop on the communication between IoT devices and learn private information about users. We assume that the devices are deployed within an indoor closed physical space (e.g., smart home, smart office, industrial control room, etc.), and controlled by a common trusted entity (e.g., smart home owner, office occupants, etc.). We assume that the attacker is not present within the physical boundary of the indoor IoT environment and cannot access, add devices or control the devices inside. We also assume that the attacker has complete knowledge of the pairing protocol and has access to the communication channels.

We consider $\mathcal{A}$ can launch the following attacks: (1) Eavesdropping attack, $\mathcal{A}$ places malicious devices $D_{\mathcal{A}}$ outside the IoT environment's physical boundary to pair them with legitimate devices. $D_{\mathcal{A}}$ may be (a) embedded with off-the-shelf sensors with similar capabilities as the legitimate devices, or (b) equipped with higher-end sensors that are more powerful and expensive compared to the legitimate devices. (2) Man-in-the-middle attack, $\mathcal{A}$ intercepts the messages between legitimate devices and attempts to establish keys with them. (3) Brute-force attacks, $\mathcal{A}$ tries every possible evidence to derive the cryptographic keys used by legitimate devices. $\mathcal{A}$ can conduct this attack in two ways. For online attacks, $\mathcal{A}$ joins the key establishment process and guesses the evidence. For offline attacks, $\mathcal{A}$ eavesdrops on the communication between the legitimate devices and attempts to crack the established key after pairing. (4) Denial of key exchange, $\mathcal{A}$ participates in group key establishment with random evidences to prevent legitimate devices from establishing a key.

5.3 IoTCupid

5.3.1 System Overview

Figure 5.2 illustrates the three stages of IoTCupid. IoTCupid first processes the raw time-series data collected in real-time by both instant and continuously influenced sensors and performs a threshold-based signal detection to separate the sensor data corresponding to events (signals) from background noise (❶). For instance, it determines that Device A detects five signals (a_1-a_5), and Device B and Device C detect four signals (b_1-b_4, c_1-c_4).

Second, IoTCupid extracts distinctive time-series features from the signals each sensor has detected. It then extends a fuzzy clustering algorithm to group independent and concurrent signals into different events (❷). For example, Device A creates two event clusters, E_1 and E_2. It then groups each detected signal into one or both of these clusters (a_5 is grouped into both clusters as E_1 and E_2 occur concurrently). The clustered events are used to obtain the sequence of time intervals between consecutive events of a given type, which serve as evidence of the devices' shared context.

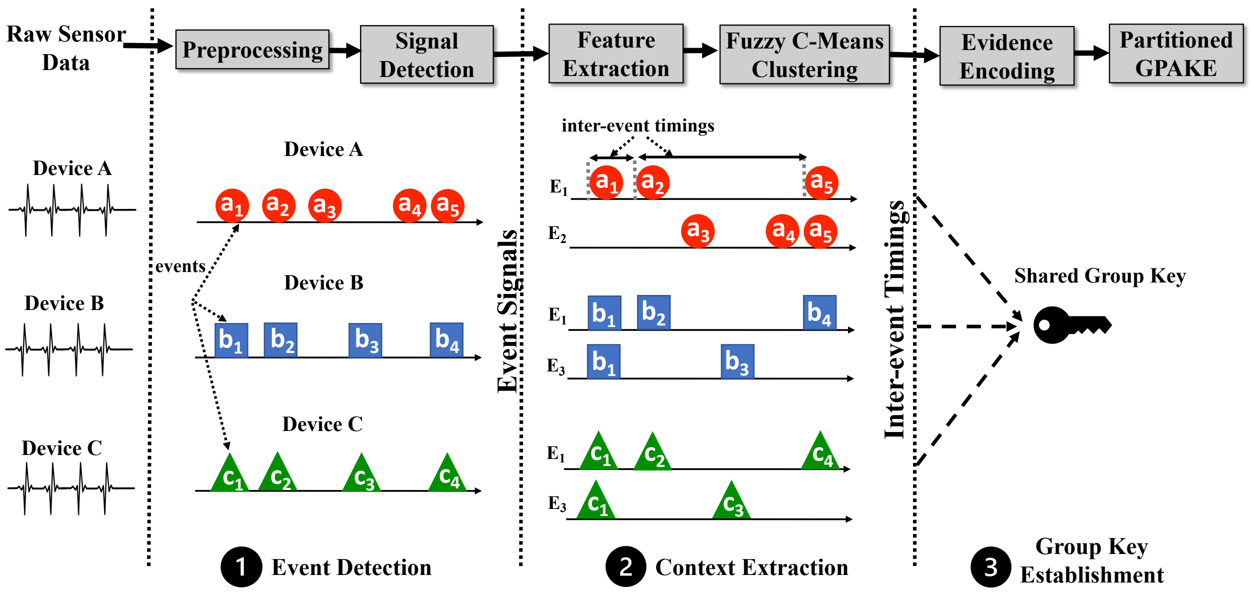


Figure 5.2. Overview of IoTCupid's architecture.

Lastly, IoT devices use the inter-event timings to authenticate each other and establish a shared group key (❸). IoTCupid encodes the inter-event timings into passwords and extends a partitioned group password-based authenticated key exchange scheme for a group key establishment protocol. We consider the devices that sense the same event as a *group*. Through this, each subset of devices that have the same inter-event timings establishes a group key. For instance, Devices A, B, and C derive a shared group key since they extract matching inter-event timings through $\mathtt{E}_1$.

Deployment. IoTCupid presents a secure system for ad-hoc connectivity among heterogeneous IoT devices without a central gateway or IoT hub. It does not require specific user actions to initiate the protocol or trigger events in the IoT environment. It solely depends on the entropy extracted from events resulting from users' routine activities. In IoTCupid's key establishment protocol initiation, all devices broadcast their public keys encrypted with the extracted inter-event timings for establishing group keys. Given the ad-hoc nature of the network, any device broadcasting its encrypted public key can initiate and participate in the protocol. Thus, IoTCupid operates without knowing the number of devices present in the IoT deployment.

5.3.2 Event Detection

IoTCupid's event detection operates on each sensor's raw time-series data. We first pre-process the sensor data for signal smoothing and noise reduction. We then perform threshold-based signal detection to separate the events' impact on sensor data from noise. We adapt our approach for both instantly and continuously influenced sensors.

Sensor Data Extraction and Pre-processing

To extract signals corresponding to events, we first segment the sensor data into multiple samples with window size, $\mathtt{w_s}$. Many sensor values heavily fluctuate throughout the day (e.g., temperature sensor readings depend on ambient temperature) [10, 16]. To address this, we first normalize the sensor readings to eliminate these fluctuations' impact and capture the transient changes caused by events. We then apply a smoothing filter by computing sensor data's exponentially weighted moving average (EWMA) to reduce noise. We compute the sensor data's EWMA as $\mathtt{S_w} = \alpha * \mathtt{Y_w} + (1 - \alpha) * \mathtt{S_{w-1}}$, where α is the weight, $\mathtt{Y_w}$ is the sensor data in window $\mathtt{w}$, and $\mathtt{S_{w-1}}$ is the EWMA of the preceding window. Appendix 5.8 shows an example of the sensor data before and after pre-processing.

Event Signal Detection

We design a threshold-based approach to distinguish events' influence on sensor readings from background noise. We leverage a lower threshold, $\mathtt{T_L}$, to identify peaks in the sensor readings that distinguish events' impact from background noise, and an upper threshold, $\mathtt{T_U}$, to remove high amplitude noise signals. We consider the consecutive timestamps at which the sensor values exceed $\mathtt{T_L}$ but are below $\mathtt{T_U}$ as a signal representing a single event.

Figure 5.3 (highlighted region) shows how the thresholds determine the microphone signal corresponding to an event. We disregard short discontinuities between the signals (Figure 5.3a) to ensure small fluctuations do not segment a signal representing a single event into multiple

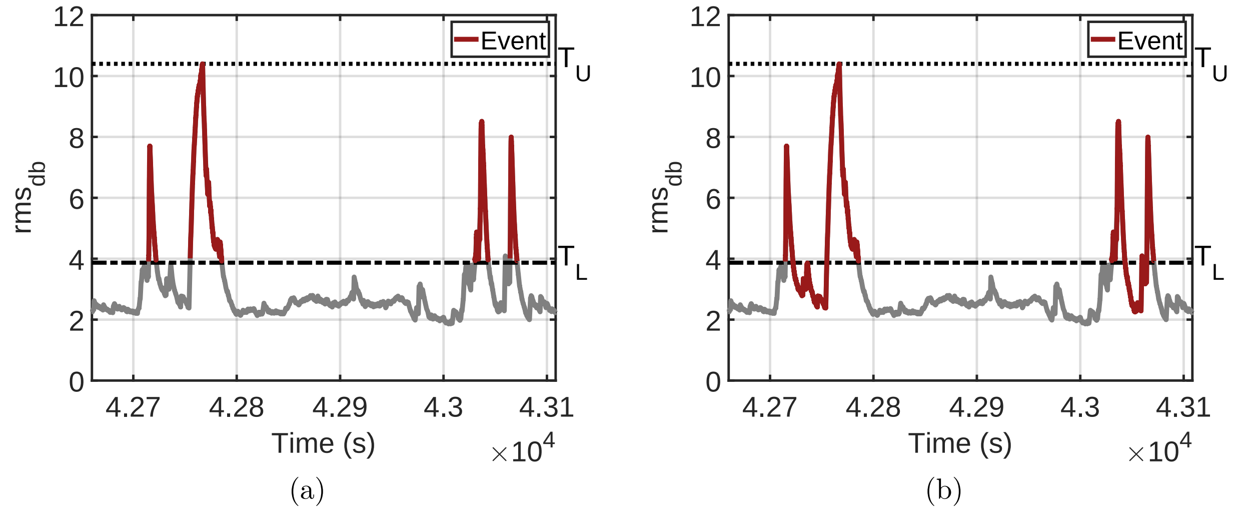
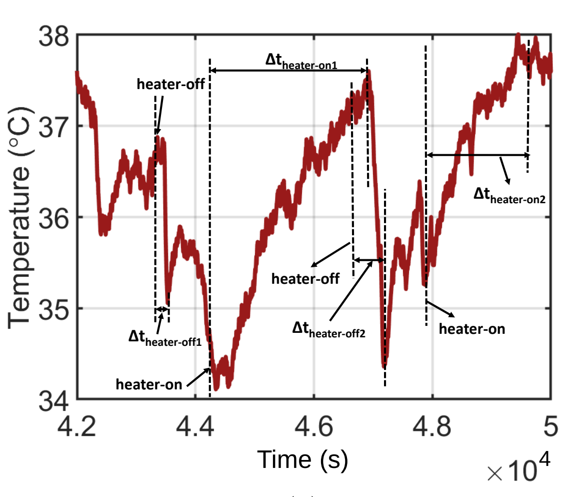


Figure 5.3. Signal detection using lower ($\mathtt{T_L}$) and upper ($\mathtt{T_U}$) thresholds. (a) shows short discontinuities between detected signals aggregated by our approach in (b).

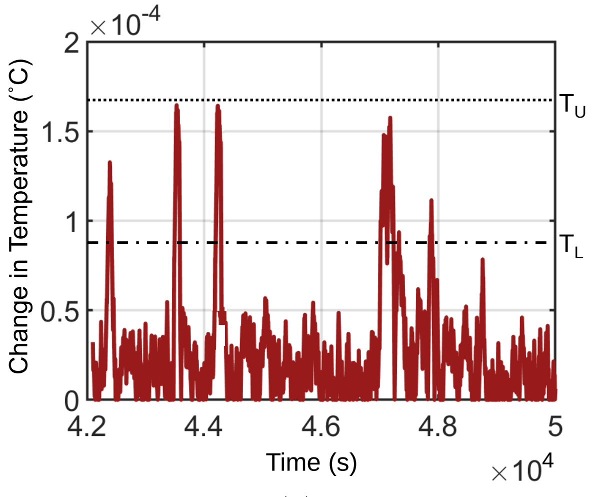
signals. To detail, we aggregate consecutive events into a single signal if the duration between these signals is less than the aggregation threshold, $\mathtt{t_A}$, (Figure 5.3b).

Detecting Continuous Physical Quantities. The event detection approach described above is suitable for sensors instantly impacted by an event since the raw sensor readings can be directly compared to the thresholds. Yet, many sensors measure continuous physical quantities, such as temperature, and humidity, that may not change instantly in response to an event. For instance, a `heater-on` event occurring at time `t` causes a gradual increase in temperature sensor values after a delay ($\Delta\mathtt{t}$), which may vary depending on environmental factors such as the sensor's distance from the heater.

To illustrate, consider the raw sensor data from a temperature sensor shown in Figure 5.4a. The sensor values gradually increase or decrease in response to `heater-on`/`off` events. However, the maximum (or minimum) temperature values are sensed after a delay from the original event timestamp. The length of this delay is different even for two events of the same type. For example, for the two `heater-on` events, $\Delta\mathtt{t}_{\mathtt{heater\text{-}on2}}$ is larger than $\Delta\mathtt{t}_{\mathtt{heater\text{-}on1}}$. Additionally, the maximum (or minimum) values recorded in response to the two events also vary. Due to



(a)



(b)

Figure 5.4. (a) Raw sensor data collected from a temperature sensor. (b) Absolute of the first derivative of the temperature data with upper and lower thresholds.

these variations, using thresholds determined from the raw sensor data results in inaccurate event detection and incorrect inter-event timings.

To account for the gradual changes and the varying delay in the impact on sensor readings, IoTCupid leverages the rate of change in the sensor readings to detect signals corresponding to events for continuously influenced sensors. IoTCupid first computes the derivative of the pre-processed sensor values in each window ($\mathtt{w}$) as $\mathtt{S}'_{\mathtt{w}} = (\mathtt{S}_{\mathtt{w}_{\mathtt{w}_\mathtt{s}}} - \mathtt{S}_{\mathtt{w}_0})/\mathtt{w}_\mathtt{s}$, where $\mathtt{w}_\mathtt{s}$ is the window size and $\mathtt{w}_0$ and $\mathtt{w}_{\mathtt{w}_\mathtt{s}}$ are the first and last sensor values in the window. IoTCupid then applies the lower and upper thresholds ($\mathtt{T}_\mathtt{L}$ and $\mathtt{T}_\mathtt{U}$) determined based on the average derivative values for each sensor. We extract the timestamps where the absolute value of the sensor readings' derivatives lie within the predetermined $\mathtt{T}_\mathtt{L}$ and $\mathtt{T}_\mathtt{U}$ for the sensor. Compared to the raw sensor data, the sensor data's derivative has clear peaks that align well with the events' timestamps, as shown in Figure 5.4b.

5.3.3 Context Extraction

IoTCupid leverages inter-event timings as evidence of a shared context among devices. To compute inter-event timings, it first derives the temporal features of the detected signals, performs dimensionality reduction, and clusters them into events. It then computes the sequence of time intervals between the start times of events of a given type.

Event Clustering

We cluster the detected signals into different events to extract their inter-event timings. This is because each device can sense multiple event types, and the devices do not know the type of detected signals. We use temporal features extracted from the detected signals to cluster similar events via a fuzzy clustering algorithm, without any prior information about the event types.

Deriving Temporal Sensor Features. IoTCupid implements a feature extraction and selection algorithm to extract features representing different events. It extracts time-domain features (`F`) such as min, max, mean, and median from the signals corresponding to each event. We perform feature selection to identify the features that enable the correct characterization of different events. We normalize the selected features and perform dimensionality reduction through principal component analysis (PCA) [182] to select a subset of features, $\mathtt{F}_{\mathtt{min}}$, which is used to differentiate event types.

Fuzzy C-Means Event Clustering. In real IoT deployments, multiple events may occur simultaneously and produce an overlapping signal on the devices. For instance, consider the event sequence shown in Figure 5.5, where `door-open` and `coffee-machine-on` events occur concurrently. The signal features corresponding to such simultaneous events may significantly differ from the same event's occurrence in isolation. Thus, traditional hard clustering methods such as K-Means may cluster concurrent events into a separate type instead of the existing types (as shown by the K-Means output in Figure 5.5). This approach results in a higher mismatch in the inter-event timings generated by two devices for a given event type, leading to longer pairing times.

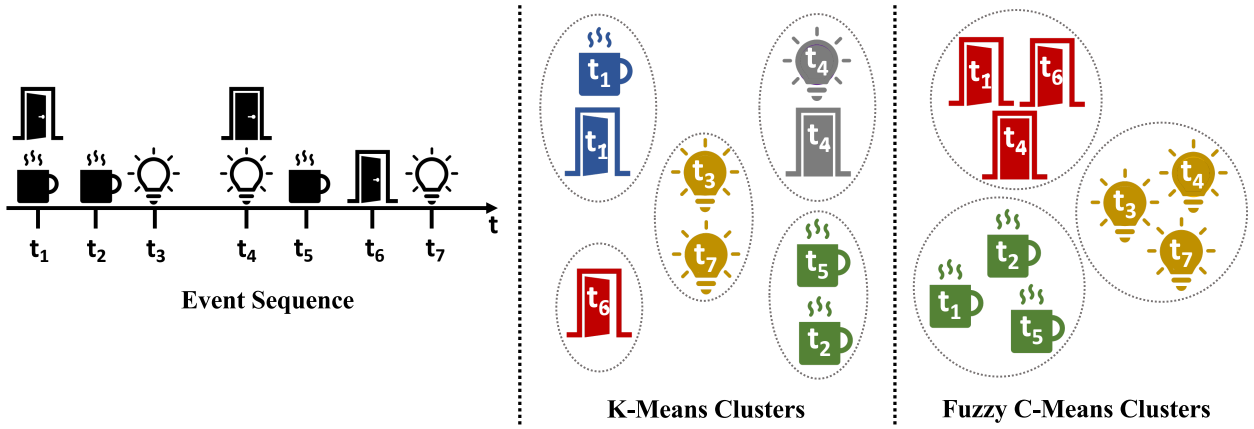


Figure 5.5. Comparison of event clustering through K-Means and Fuzzy C-Means algorithms.

To address this, we extend fuzzy C-Means clustering [183] to assign the detected signals into one or more appropriate event clusters based on the extracted features. Fuzzy C-Means partitions signal into $\mathtt{c}$ fuzzy event clusters, where $\mathtt{c}$ is an input. Each signal corresponding to an event is assigned a degree of membership to each of the $\mathtt{c}$ clusters. For a given signal ($\mathtt{e}$), the degree of its membership ($\mathtt{u_{ej}}$) in cluster $\mathtt{j}$ is determined by $\mathtt{u_{ej}} = 1/\sum_{\mathtt{k}=1}^{\mathtt{c}}(\mathtt{d_{ej}}/\mathtt{d_{ek}})^{2/\mathtt{m}-1}$, where $\mathtt{d_{ej}}$ represents the Euclidean distance between the cluster $\mathtt{j}$'s centroid and signal $\mathtt{e}$'s feature vector. Since a device does not know the possible event types that may occur, IoTCupid employs the elbow method to identify the optimal value of $\mathtt{c}$ [33]. The fuzziness index, $\mathtt{m}$, is the hyper-parameter controlling the tendency of an element to belong to multiple clusters. We detail how $\mathtt{c}$ and $\mathtt{m}$ are determined in Section 5.4. This process allows two events occurring simultaneously to belong to their appropriate clusters (Figure 5.5 (Right)).

Context Evidence Generation

IoTCupid generates inter-event timings, $\mathtt{I}$, for each device after clustering the detected signals into different event types. Each device $\mathtt{d}$ extracts and concatenates the time intervals between the start times of event occurrences $\langle \mathtt{e_1}, \ldots, \mathtt{e_n} \rangle$ in cluster $\mathtt{k}$ to generate inter-event timings, $\mathtt{i_k}$. IoTCupid uses the inter-event timings for all events sensed by a device ($\mathtt{I}$) as evidence in our group key establishment protocol (Section 5.3.4).

Table 5.2. Our group key establishment protocol.

	Device 1 ($\mathtt{d}_1$)	**Device 2** ($\mathtt{d}_2$)	...	**Device N** ($\mathtt{d}_N$)
	Step 1: Evidence Extraction			
❶	$\{\mathtt{i}_{1,d_1} \dots \mathtt{i}_{c_1,d_1}\} = \text{CONTEXT\_EXTRACTION}(\mathtt{d}_1)$	$\{\mathtt{i}_{1,d_2} \dots \mathtt{i}_{c_2,d_2}\} = \text{CONTEXT\_EXTRACTION}(\mathtt{d}_2)$		$\{\mathtt{i}_{1,d_N} \dots \mathtt{i}_{c_N,d_N}\} = \text{CONTEXT\_EXTRACTION}(\mathtt{d}_N)$
	Step 2: Encoding			
❷	$\{\mathtt{pw}_{1,d_1} \dots \mathtt{pw}_{c_1,d_1}\} = \lfloor\{\mathtt{i}_{1,d_1} \dots \mathtt{i}_{c_1,d_1}\}/\mathtt{W}\rfloor$	$\{\mathtt{pw}_{1,d_2} \dots \mathtt{pw}_{c_2,d_2}\} = \lfloor\{\mathtt{i}_{1,d_2} \dots \mathtt{i}_{c_2,d_2}\}/\mathtt{W}\rfloor$		$\{\mathtt{pw}_{1,d_N} \dots \mathtt{pw}_{c_N,d_N}\} = \lfloor\{\mathtt{i}_{1,d_N} \dots \mathtt{i}_{c_N,d_N}\}/\mathtt{W}\rfloor$
	Step 3: Partitioned GPAKE			
❸	Determine the public parameters, two primes $\mathtt{p}$ and $\mathtt{q}$, a finite field $\mathbb{F}_q$ and a group $\mathbb{Z}_p$. $\mathtt{E}(\mathbb{F}_q)$ is an elliptic curve, and $\mathtt{P} \in \mathtt{E}(\mathbb{F}_q)$ is its generator.			
❹	Choose random $\{\mathtt{x}_{1,d_1} \dots \mathtt{x}_{c_1,d_1}\} \overset{\$}{\leftarrow} \mathbb{Z}_p$	Choose random $\{\mathtt{x}_{1,d_2} \dots \mathtt{x}_{c_2,d_2}\} \overset{\$}{\leftarrow} \mathbb{Z}_p$		Choose random $\{\mathtt{x}_{1,d_N} \dots \mathtt{x}_{c_N,d_N}\} \overset{\$}{\leftarrow} \mathbb{Z}_p$
❺	$\mathtt{X}_{i,d_1} \leftarrow \mathtt{x}_{i,d_1} \cdot \mathtt{P} \bmod \mathtt{q}$, $\mathtt{Y}_{i,d_1} \leftarrow \mathtt{Enc}_{\mathtt{pw}_{i,d_1}}(\mathtt{X}_{i,d_1})$	$\mathtt{X}_{i,d_2} \leftarrow \mathtt{x}_{i,d_2} \cdot \mathtt{P} \bmod \mathtt{q}$, $\mathtt{Y}_{i,d_2} \leftarrow \mathtt{Enc}_{\mathtt{pw}_{i,d_2}}(\mathtt{X}_{i,d_2})$		$\mathtt{X}_{i,d_N} \leftarrow \mathtt{x}_{i,d_N} \cdot \mathtt{P} \bmod \mathtt{q}$, $\mathtt{Y}_{i,d_N} \leftarrow \mathtt{Enc}_{\mathtt{pw}_{i,d_N}}(\mathtt{X}_{i,d_N})$
❻	Broadcast $(\mathtt{d}_1, \mathtt{Y}_{i,d_1})$, where $\mathtt{i} \in \{1, \dots, \mathtt{c}_1\}$	Broadcast $(\mathtt{d}_2, \mathtt{Y}_{i,d_2})$, where $\mathtt{i} \in \{1, \dots, \mathtt{c}_2\}$		Broadcast $(\mathtt{d}_N, \mathtt{Y}_{i,d_N})$, where $\mathtt{i} \in \{1, \dots, \mathtt{c}_N\}$
❼	For every received message $(\mathtt{d}_j, \mathtt{Y}_{j,d_j})$, where $\mathtt{j} \in \{1, \dots, \mathtt{N}\}$:			
❽	$\mathtt{X}_{j,d_j} \leftarrow \mathtt{Dec}_{\mathtt{pw}_{i,d_1}}(\mathtt{Y}_{j,d_j})$, $\mathtt{if}(\mathtt{X}_{j,d_j} \in \mathtt{E}(\mathbb{F}_q))$:	$\mathtt{X}_{j,d_j} \leftarrow \mathtt{Dec}_{\mathtt{pw}_{i,d_1}}(\mathtt{Y}_{j,d_j})$, $\mathtt{if}(\mathtt{X}_{j,d_j} \in \mathtt{E}(\mathbb{F}_q))$:		$\mathtt{X}_{j,d_j} \leftarrow \mathtt{Dec}_{\mathtt{pw}_{i,d_1}}(\mathtt{Y}_{j,d_j})$, $\mathtt{if}(\mathtt{X}_{j,d_j} \in \mathtt{E}(\mathbb{F}_q))$:
❾	$\mathtt{sid}_{i,j,d_1} = \{\mathtt{d}_1, \mathtt{Y}_{i,d_1}, \mathtt{d}_j, \mathtt{Y}_{j,d_j}\}$	$\mathtt{sid}_{i,j,d_2} = \{\mathtt{d}_2, \mathtt{Y}_{i,d_2}, \mathtt{d}_j, \mathtt{Y}_{j,d_j}\}$		$\mathtt{sid}_{i,j,d_N} = \{\mathtt{d}_N, \mathtt{Y}_{i,d_N}, \mathtt{d}_j, \mathtt{Y}_{j,d_j}\}$
❿	$\mathtt{sk}_{i,j,d_1} \leftarrow \mathtt{H}(\mathtt{d}_1, \mathtt{d}_j, \mathtt{X}_{i,d_1}, \mathtt{X}_{j,d_j}, \mathtt{x}_{i,d_1} \cdot \mathtt{X}_{j,d_j} \bmod \mathtt{q})$	$\mathtt{sk}_{i,j,d_2} \leftarrow \mathtt{H}(\mathtt{d}_2, \mathtt{d}_j, \mathtt{X}_{i,d_2}, \mathtt{X}_{j,d_j}, \mathtt{x}_{i,d_2} \cdot \mathtt{X}_{j,d_j} \bmod \mathtt{q})$		$\mathtt{sk}_{i,j,d_N} \leftarrow \mathtt{H}(\mathtt{d}_N, \mathtt{d}_j, \mathtt{X}_{i,d_N}, \mathtt{X}_{j,d_j}, \mathtt{x}_{i,d_N} \cdot \mathtt{X}_{j,d_j} \bmod \mathtt{q})$
⓫	$\mathtt{r}_{i,d_1} \overset{\$}{\leftarrow} \mathbb{Z}_p$, $\alpha_{i,j,d_1} \leftarrow \mathtt{Enc}_{\mathtt{sk}_{i,j,d_1}}(\mathtt{r}_{i,d_1})$	$\mathtt{r}_{i,d_2} \overset{\$}{\leftarrow} \mathbb{Z}_p$, $\alpha_{i,j,d_2} \leftarrow \mathtt{Enc}_{\mathtt{sk}_{i,j,d_2}}(\mathtt{r}_{i,d_2})$		$\mathtt{r}_{i,d_N} \overset{\$}{\leftarrow} \mathbb{Z}_p$, $\alpha_{i,j,d_N} \leftarrow \mathtt{Enc}_{\mathtt{sk}_{i,j,d_N}}(\mathtt{r}_{i,d_N})$
⓬	Broadcast $(\mathtt{d}_1, \mathtt{sid}_{i,j,d_1}, \alpha_{i,j,d_1})$	Broadcast $(\mathtt{d}_2, \mathtt{sid}_{i,j,d_2}, \alpha_{i,j,d_2})$		Broadcast $(\mathtt{d}_N, \mathtt{sid}_{i,j,d_N}, \alpha_{i,j,d_N})$
⓭	For every received message $(\mathtt{d}_j, \mathtt{sid}_{i,j,d_j}, \alpha_{i,j,d_j})$, where $\mathtt{i} \in \{1, \dots, \mathtt{c}\}$ and $\mathtt{j} \in \{1, \dots, \mathtt{N}\}$:			
⓮	$\mathtt{if}(\mathtt{Y}_{i,d_1} \in \mathtt{sid}_{i,j,d_j}) : \mathtt{r}_{i,j,d_j} \leftarrow \mathtt{Dec}_{\mathtt{sk}_{i,j,d_1}}(\alpha_{i,j,d_j})$	$\mathtt{if}(\mathtt{Y}_{i,d_2} \in \mathtt{sid}_{i,j,d_j}) : \mathtt{r}_{i,j,d_j} \leftarrow \mathtt{Dec}_{\mathtt{sk}_{i,j,d_2}}(\alpha_{i,j,d_j})$		$\mathtt{if}(\mathtt{Y}_{i,d_N} \in \mathtt{sid}_{i,j,d_j}) : \mathtt{r}_{i,j,d_j} \leftarrow \mathtt{Dec}_{\mathtt{sk}_{i,j,d_N}}(\alpha_{i,j,d_j})$
⓯	$\mathtt{key}_i \leftarrow \sum_{j=1}^{t}(\mathtt{r}_{i,j,d_j})$	$\mathtt{key}_i \leftarrow \sum_{j=1}^{t}(\mathtt{r}_{i,j,d_j})$		$\mathtt{key}_i \leftarrow \sum_{j=1}^{t}(\mathtt{r}_{i,j,d_j})$

Using inter-event timings as evidence to bootstrap key establishment has significant advantages for pairing heterogeneous devices. First, two devices detecting the same event roughly record the same inter-event timings even if their raw signals have different characteristics (e.g., `door-close` events sensed by a microphone and gyroscope). Second, inter-event timings eliminate the need for a global clock or time synchronization. Even if an event's timestamps observed by two devices are not synchronized (e.g., `heater-on` is instantly sensed by a sound sensor but gradually sensed by a temperature sensor with a delay), the time intervals between consecutive events of the same type are still similar. Third, inter-event timings eliminate the impact of changes in the duration of detected events, e.g., a `coffee-machine-on` event's duration depends on the number of cups.

5.3.4 Establishing Group Keys from Evidences

After devices extract inter-event timings, they use the timings as evidence to authenticate each other and establish group keys. A group key is shared between the devices that sense the same event type, and all devices in the group can securely communicate using a single key. Group keys have unique advantages over deriving individual keys among each device pair. First, devices must store an individual key for each device they pair with. Second, when a device communicates with multiple devices (e.g., broadcasts a message), it must encrypt and authenticate the message with all individual keys. Due to the storage, computation, and energy constraints of IoT devices, linear storage, communication, and computation overhead from individual keys significantly deteriorates the devices' performance and battery.

Design Space Exploration

Deriving group keys using inter-event timings from multiple events in a dynamic IoT deployment introduces several challenges. First, the groups must be generated dynamically based on the devices that sense the same event. Second, the group key establishment protocol must support device addition and removal. When a device is added, it must pair with the existing devices for secure communication, and when a device is removed, its keys must be revoked since an adversary can capture it (e.g., through reselling or returning [168, 184]) and physically extract the keys. Unfortunately, prior works cannot be easily extended to address these challenges.

Group Diffie-Hellman. Group keys can be generated using the secure communication channels from the individual keys derived through a standard pairing protocol. This means the devices run an additional Group Diffie-Hellman (GDH) [185] protocol to derive group keys, increasing the pairing time. Here, the group key establishment protocol must be run over the secure communication channels to prevent MitM attacks. Since group key establishment protocols require multiple broadcasting rounds, using the secure channels between each pair of devices further increases the communication and computation overhead of the group key establishment. Additionally, when a device is added to the IoT environment, it must first individually pair with other devices to be authenticated and then participate in the group key establishment.

Fuzzy Commitment. Several approaches use fuzzy commitment schemes [186, 187] to generate individual keys. These schemes are built on error-correcting codes and enable verifying two evidences even when they have small differences (e.g., Hamming distance less than a threshold). These schemes can be extended to derive group keys where each device broadcasts its commitment, and the ones with similar evidences derive the same keys. Yet,

Table 5.3. Comparison of group key exchange approaches.

Group Key Exchange Protocol	**Dynamic Group Generation**	**Efficiency**	**Resilience to Offline Attacks**	**Resilience to Denial of Key Exchange**
Group DH	✗	✗	✓	✓
Fuzzy Commitment-based	✓	✗	✗	✓
Group PAKE	✗	✓	✓	✗
Our Protocol	✓	✓	✓	✓

approaches only built on fuzzy commitment are vulnerable to offline brute-force key guessing attacks [12]. In this attack, the adversary collects the network traffic and tries all evidences until they find the one that can decrypt the network traffic.

There are two approaches to protect against these attacks. First, a large number of evidences (i.e., inter-event timings) can be used to derive the keys. Yet, this approach sacrifices efficiency since it may take a long time to derive a large number of evidences. Second, Password-Authenticated Key Exchange (PAKE) schemes have been proposed to prevent offline brute-force attacks. However, extending PAKE into group settings is non-trivial, as discussed below.

Group PAKE (GPAKE). GPAKE enables multiple devices sharing the same evidence to derive group keys [188, 189]. However, the passwords of all devices that participate in the key agreement must be the same because these schemes abort without establishing a shared key even if a single password is different. The adversary can leverage this limitation by joining the key agreement protocol with arbitrary evidences to deny legitimate devices from deriving shared keys. We refer to this attack as *Denial of Key Exchange.*

Our Group Key Establishment Protocol

Table 5.2 shows our group key establishment protocol, which offers dynamic group generation with computational efficiency, device addition/removal, and resilience to offline brute-force and denial of key exchange attacks. We extend a partitioned GPAKE scheme [190] to build our protocol. Particularly, we include evidence extraction and encoding steps to first derive inter-event timings and then encode them into passwords to address their deviations. The devices then use the passwords to run the partitioned GPAKE scheme such that each subset of devices sensing the same events derives a group key. Here, we implement the partitioned GPAKE scheme over an elliptic curve to offer compact key sizes and fast computations. Lastly, we introduce a re-initiation-based key management scheme to support device additions and removals. Table 5.3 shows our protocol's advantages over the alternatives.

Evidence Extraction. Each device first extracts inter-event timings as evidence of co-location (❶). We represent the evidences as $\{\mathtt{i}_1, \ldots, \mathtt{i}_\mathtt{c}\}$, where $\mathtt{c}$ is the number of event types the device senses. For each event type, the devices can concatenate multiple inter-event timings to increase the entropy of the evidence. Our protocol requires 32-bit entropy in its evidences, whereas the protocols based only on fuzzy commitment (e.g., Perceptio's key establishment [10]) need 128-bit entropy that requires a larger number of inter-event timings. This is because our protocol is resilient to offline brute-force attacks, where an adversary who guesses the password correctly after the group keys are established cannot extract the keys. We further elaborate on each inter-event timing's entropy and required number of timings in Section 5.5.

Encoding. Slight deviations in the inter-event timings may occur since devices may have different sampling rates for their measurements. For instance, a device with a 10 Hz rate collects a sensor measurement every 0.1 seconds, whereas a device with 1 Hz rate collects every second. This leads the first device to compute an inter-event timing of 24.2 whereas the second device computes it as 24. Yet, the passwords used in the partitioned GPAKE must be identical for the devices to pair. To address this, we use a quantization window ($\mathtt{W}$) to round down the inter-event timings while deriving the passwords (❷). Here, the quantization window introduces a trade-off between efficiency and the password's entropy. With a larger $\mathtt{W}$, more devices have matching passwords, and with a smaller $\mathtt{W}$, the passwords have higher entropy.

Partitioned GPAKE. We extend a partitioned GPAKE scheme with provable key secrecy and password privacy [190]. In this, a probabilistic polynomial time adversary who does not know the passwords cannot extract keys or passwords by eavesdropping or MitM attacks (See Appendix 5.9 for proof).

We implement the partitioned GPAKE scheme on an elliptic curve (EC). The devices first determine the public EC parameters (❸). Each device then generates a unique private and public key pair for each event type it senses and encrypts the public keys with its passwords (❹-❺). The devices broadcast the encrypted public keys with their device identifiers (❻). Only the devices with the same password can decrypt the messages, preventing an adversary from conducting a MitM attack.

After a device receives the encrypted public keys (❼), it tries to decrypt them using all of its passwords. If one matches with the password the public key was encrypted with, the device derives a valid public key (❽). The device then generates session IDs using the received IDs and public keys (❾), and derives intermediate two-party elliptic curve Diffie-Hellman (ECDH) keys (❿). It generates random values for each event type, encrypts them using the intermediate keys, and broadcasts along with its ID and the session ID (⓫-⓬).

Upon receiving a message, the device checks whether its ID is in the session ID of the received message. If it is, the device decrypts the message with its intermediate key to derive the random value of the other device (⓭-⓮). After collecting all such random values, the device adds them to derive the group keys (⓯). Since random values are uniformly sampled from $\mathbb{Z}_p$, the group keys are random and secure. Yet, if the devices do not have a reliable source of randomness, they can use a key derivation function instead of addition to generate group keys. At the end of the protocol, each device derives a separate shared group key with the other devices that can sense the same event type.

Key Management. In IoT environments, device additions and removals are common. Yet, an added device cannot securely communicate with other devices without establishing keys with them, and an adversary may leverage the removed devices to physically extract keys. Therefore, a key management scheme is required to support added and removed devices. Prior schemes for sensor networks, however, require a trusted entity to assign keys to devices [191–193], which is infeasible for IoT devices from different vendors.

To address this problem, we integrate a re-initiation-based key management strategy. First, the added devices re-initiate IoTCupid and extract inter-event timings to prove their legitimacy to the existing devices. Thus, when IoTCupid is re-initiated, the existing devices also start extracting inter-event timings to pair with the newly added devices. Yet, an adversary can abuse this and attempt a denial of service attack by initiating the protocol unnecessarily. Such attacks are easy to detect since the adversary cannot prove its legitimacy and pair with the existing devices. When an adversarial pairing attempt is detected, the devices notify the user and we provide a waiting period to re-initiate the protocol. This period offers a trade-off between time to re-initiate the protocol for benign devices and the security against denial of service attacks.

Second, the removed devices can no longer derive correct inter-event timings. We integrate a periodic liveness check to detect if a device is removed and initiate IoTCupid to derive new keys after a device fails the liveness check.

Group-to-Group Communication. Devices may need to communicate with other devices in the same environment that they do not share a common event type. In such cases, they can leverage intermediary devices from their groups for communication. For instance, we consider a case where devices $\mathtt{d_a}, \mathtt{d_b}, \mathtt{d_c}$ share a key, and devices $\mathtt{d_b}, \mathtt{d_c}, \mathtt{d_d}$ share a key. When $\mathtt{d_a}$ needs to communicate with $\mathtt{d_d}$, it can leverage $\mathtt{d_b}$ or $\mathtt{d_c}$ as an intermediary device for secure communication.

Attack Resiliency. Our protocol offers resiliency to MitM attacks, offline brute-force password enumeration, and denial of key exchange. First, the adversary can try to conduct a MitM attack by intercepting a legitimate device's messages and establishing keys with other devices. Yet, as legitimate devices encrypt their public keys with passwords (❺), the adversary cannot decrypt them to join the protocol. Thus, the passwords (i.e., inter-event timings) provide the authentication necessary to protect against MitM attacks.

Second, the adversary can enumerate all possible inter-event timings to find the password used. However, the adversary still cannot extract the group keys due to the online ECDH session in the partitioned GPAKE (❿). Particularly, since the probability that the adversary can guess the password is very low (e.g., $1/2^{32}$), the adversary's attack can be successful only after the key agreement is completed. If an adversary recovers the password after keys are established, the adversary can decrypt the devices' broadcasted public keys. Yet, the adversary cannot use them to derive the private keys or group keys due to the hardness of the computational elliptic curve Diffie-Hellmann problem [194].

Lastly, an adversary can enter the protocol with random passwords to disrupt pairing. Here, the legitimate devices cannot decrypt the adversary's public keys and thus would ignore the adversary's keys while deriving their group keys.

5.4 Implementation

Event Detection. We implement IoTCupid's event detection in Python 3.9.12. IoTCupid's event detection module uses lower and upper thresholds for sensors to detect events. Prior pairing schemes relying on event detection for pairing assume that these thresholds are built-in to the sensors by device manufacturers or manually configured [10]. However, in the case that such thresholds are not available (as for the devices in our experiments), we design an approach for threshold identification, which requires minimal sensor data without any information about the event types.

Our approach begins by separating the ambient noise from the event signals from the sensor data. We first extract the signal data in the interval $[\mathtt{t_s} - \Delta\mathtt{t_n}, \mathtt{t_e} + \Delta\mathtt{t_n}]$ for all events, where $\mathtt{t_s}$ and $\mathtt{t_e}$ are event start and end timestamps, and consider the rest of sensor data as noise. From the sensor signal values corresponding to an event, we consider the values between intervals $[\mathtt{t_s}, \mathtt{t_s} + \Delta\mathtt{t_v}]$ and $[\mathtt{t_e} - \Delta\mathtt{t_v}, \mathtt{t_e}]$ as event start and end signals. Here, we empirically determine $\Delta\mathtt{t_n}$ and $\Delta\mathtt{t_v}$ from a specific event type's average duration. We determine the frequency distribution of the signal values for noise and event samples separately by assigning them to equally sized bins. We then select the bin with the highest and lowest frequency for the event and noise samples, respectively. We set the selected bin's maximum and minimum values as the upper ($\mathtt{T_U}$) and lower ($\mathtt{T_L}$) thresholds. This approach allows IoTCupid to determine event detection thresholds for both instantly and continuously influenced sensors.

To determine the optimal value for the event detection window size ($\mathtt{w_s}$) and aggregation threshold ($\mathtt{t_a}$), we perform a grid search between 1 sec to 5 mins and choose the ones that give the highest event detection accuracy.

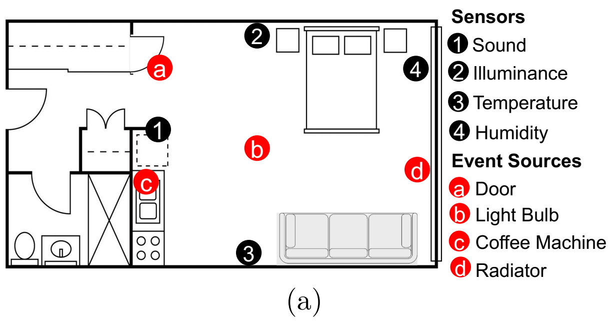
Context Extraction. We use the `MinimalFCParameters` class of `tsfresh` [195] package in Python to extract common time-domain features. We extend the Scikit-Fuzzy [196] package to implement the fuzzy C-Means clustering. To determine the optimal number of clusters ($\mathtt{c}$), we employ elbow method [33], average Silhouette coefficient [197], and gap statistic [198]. We observe the optimal number of clusters is similar using these methods and select the elbow method. Since IoTCupid does not assume any prior information about the type of events occurring in the environment, we need to identify the fuzziness index ($\mathtt{m}$) to accurately separate independent and concurrent events. For this, we perform a grid search to select the optimal value for the fuzziness index ($\mathtt{m}$) based on the variance in the distances between cluster centroids and event feature vectors.

Group Key Establishment. We implement partitioned GPAKE on the FourQ elliptic curve [199], which offers 128-bit security with fast computations. We use blake2 [200] as the hash function and ChaCha-Poly [201] as the authenticated encryption due to their efficiency. We leverage the portable libraries of FourQ [202], blake2 [203] and ChaCha-Poly [204] to implement our protocol in C. We implement the communication rounds with the zeromq library [205].

5.5 Evaluation

We perform two studies to evaluate IoTCupid in two different IoT environments, a smart home and smart office.

In the first study, we conduct experiments with 4 sensors and 4 event sources to evaluate IoTCupid's effectiveness in pairing devices inside a home and its security against attacks launched from outside the environment. In the second study, we evaluate IoTCupid on a dataset [177, 178] collected in an office with multiple sensors and event sources to demonstrate IoTCupid's practicality in different environmental conditions. Our studies show IoTCupid effectively pairs all devices in the smart home and office via shared group keys using only



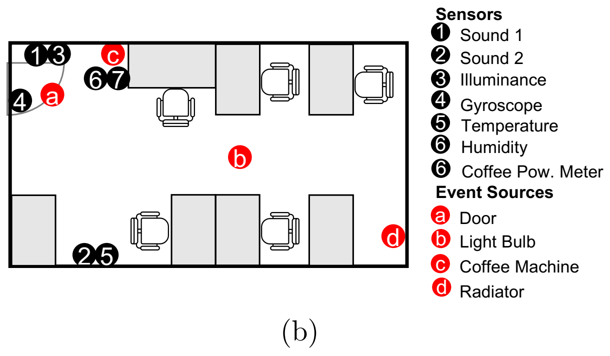


Figure 5.6. IoT deployments in (a) a smart home and (b) office.

four equivalent inter-event timings extracted from 13 or fewer events detected by each device. We present our IoTCupid analysis results by focusing on several research questions:

RQ1 What is the accuracy of IoTCupid in event detection?

RQ2 What is the impact of sensors' locations on IoTCupid's event detection accuracy?

RQ3 What is the time required to achieve sufficient entropy for our group key establishment protocol?

RQ4 How effective is IoTCupid in establishing group keys?

RQ5 How resilient is IoTCupid against adversarial sensors?

RQ6 What is IoTCupid's performance overhead?

RQ7 How does IoTCupid perform against other schemes?

Evaluation Setup. Figure 5.6a shows the placement of the 4 event sources and 4 sensors in the smart home. The deployed sensor types are similar to commonly found sensors in smart homes. The event sources include the smart home's interior door, a ceiling light, an electric drip coffee machine, and a portable heater. We collect sensor traces over three days during which events are sporadically triggered by the two occupants (authors) while conducting their daily activities (e.g., walking, cooking, cleaning, etc.). We contacted our universitys IRB office and got advised that IRB approval is not required as we do not collect any sensitive information.

Table 5.4. Events detected by sensors in IoT environments.

Events[†]	Sound	Illum.	Gyroscope	Temp.	Humidity	Pow. meter
`Door-open/close`	✓	✓	✓	✓	✓	✗
`Coffee-machine-on`	✓	✗	✗	✗	✗	✓
`Light-on/off`	✗	✓	✗	✗	✗	✗
`Radiator-on/off`	✗	✗	✗	✓	✓	✗

† The smart home includes a BMP180 temperature sensor, DHT11 humidity sensor, photoresistive illuminance sensor and an iPhone XR microphone. The smart office includes two USB microphones, a TSL2560 illuminance sensor, an ST Micro LSM9DS1 gyroscope, a BMP280 temperature sensor and a BME680 humidity sensor. There are two microphones in the office and they detect the same events.

For the second environment, we use 4 event sources and 7 sensors, as shown in Figure 5.6b. The sensor data is measured over three days while four people are using the office for their everyday work. The door, light, and coffee events are triggered during the office occupants' routine activities, while the radiator events are triggered automatically.

We run IoTCupid on a Raspberry Pi 4 with ARM Cortex-A72 processor and 2 GB RAM.

5.5.1 Event Detection Performance

We evaluate each sensor's performance in distinguishing the events' physical influences from background noise. With a higher event detection accuracy, more sensors can derive similar inter-event timings in a shorter duration and establish group keys. Table 5.4 presents the events detected in the two environments. For instance, `door-open/close` events are detected by all devices except the power meter, and `light-on/off` events are detected by the illuminance sensor.

We present the event detection accuracy for each sensor type in terms of precision and recall. The precision represents the ratio of the number of detected events whose start times accurately match the ground truth event start times. The recall is the ratio of the number of correctly detected events to the number of events occurred. We only use the events' ground truth timestamps in the first hour of data collection from the smart office for identifying event

Table 5.5. Smart home event detection results.

Event Sources	Sensors	Precision	Recall
ⓐ `door-open/close`	❶ Sound	1.0	1.0
	❷ Illuminance	1.0	1.0
	❸ Temperature	1.0	0.97
	❹ Humidity	0.9	1.0
ⓑ `light-on/off`	❷ Illuminance	1.0	1.0
ⓒ `coffee-machine-on`	❶ Sound	1.0	1.0
ⓓ `radiator-on/off`	❸ Temperature	1.0	1.0
	❹ Humidity	1.0	1.0

thresholds and exclude these events from our evaluation. We use the same thresholds for event detection in both smart home and office. IoTCupid correctly detects events with an average precision and recall of 95.8% and 83%, and uses these detected events to extract matching inter-event timings for group key establishment. We note that IoTCupid's event detection may not capture some events with very small influence on the sensor data, resulting in a lower recall rate. Yet, the high precision detection of high impact events ensures IoTCupid extracts a sufficient number of inter-event timings for group key establishment.

Smart Home Deployment Results. Table 5.5 presents the event detection results of the 4 sensors in the smart home. All sensors detect events with a high precision and recall. For instance, all sensors detect the `door-open/close` events with a precision and recall greater than 0.9. Thus, these sensors can use the door events to successfully establish a group key. Additionally, we found that the events' influence on the sensor data is relatively large compared to the ambient noise such as disturbances from the home occupant's activities, resulting in a high event detection accuracy.

Smart Office Deployment Results. Table 5.6 shows the precision and recall rate for the 7 sensors in the smart office. All sensors detect events with high precision. The recall rate is also high for gyroscope, coffee power meter, and sound sensors for all events. Yet, the temperature and humidity sensors yield a lower recall rate for the `radiator-on/off` and `door-open/close` events. This low recall is attributed to the relatively small influence of these

Table 5.6. Smart office event detection results.

Event Sources	Sensors	Precision	Recall
ⓐ `door-open/close`	❶ Sound 1	1.0	0.97
	❷ Sound 2	0.91	0.86
	❸ Illuminance	0.98	0.64
	❹ Gyroscope	1.0	1.0
	❺ Temperature	0.84	0.42
	❻ Humidity	0.98	0.44
ⓑ `light-on/off`	❸ Illuminance	1.0	1.0
ⓒ `coffee-machine-on`	❶ Sound 1	1.0	0.71
	❷ Sound 2	1.0	0.71
	❼ Power Meter	1.0	1.0
ⓓ `radiator-on/off`	❺ Temperature	0.8	0.55
	❻ Humidity	0.75	0.33

events on sensor measurements, compared to the ambient changes in the temperature and humidity levels, particularly due to the uncontrolled disturbances from four office occupants' activities. Similarly, although the illuminance sensor accurately detects `light-on/off`, its recall is lower for the `door-open/close` events. This is because the change in illuminance caused by some events occurring during the day is insignificant compared to the ambient lighting in the office. Despite the low recall rate, these sensors' high precision event detection ensures that they can still correctly extract sufficient inter-event timings for participating in group pairing.

Impact of Distance on Event Detection. We conduct additional experiments in the smart home to demonstrate varying sensors' locations impact on event detection. We vary the devices' distance from the event sources between 1 to 5 m. Figure 5.7 shows the precision and recall for the detected events at different distances. IoTCupid's threshold-based signal detection lets most sensors detect all events correctly even when their distance from the event source increases. We find that the influence of door events on temperature and humidity sensors slightly deteriorates with an increased distance. This is because the short duration of the door events results in a small change in temperature and humidity, making it difficult to detect the events at a longer distance.

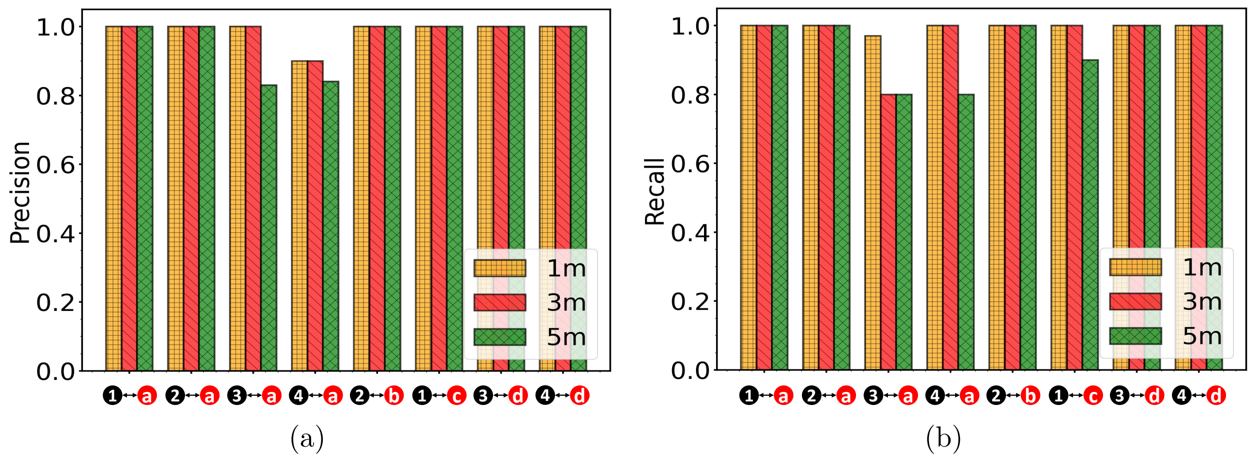


Figure 5.7. (a) Precision and (b) recall with varying distance between sensors and the event sources.

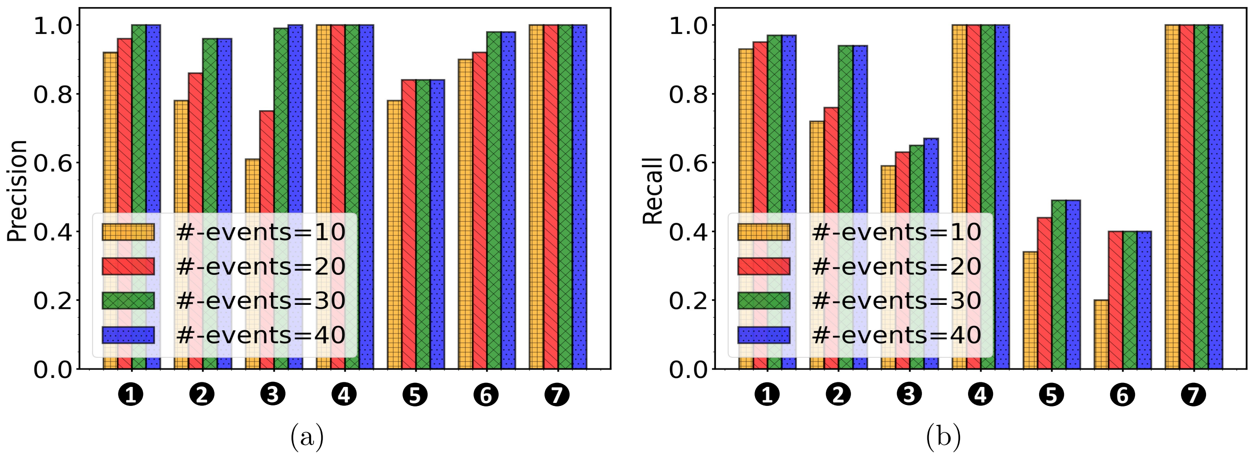


Figure 5.8. (a) Precision and (b) recall with varying number of events used for sensor calibration.

Impact of Number of Events on Sensor Calibration. We analyze how the number of events used for sensor calibration (i.e., determining sensor thresholds and parameters) impacts IoTCupid's event detection accuracy. We perform this analysis on the smart office dataset since it is collected in a noisy environment with a higher variance in the events' influence on the sensor data. Figure 5.8 shows the precision and recall rates for the detected events with an increasing number of events used for calibration. Both the precision and recall increase with more event calibration data as it improves the generalization of determined signal thresholds in detecting different event types. All sensors detect events with a precision higher than 0.8. Sensors that only detect a single event type require fewer events to achieve accurate detection. For instance, the coffee power meter and gyroscope need less than 10 events for high precision and recall.

5.5.2 Context Extraction and Key Agreement

We evaluate IoTCupid's efficacy in context extraction and key establishment by analyzing events' entropy and then present the groups of securely paired sensors. IoTCupid's fuzzy clustering enables deriving four matching inter-event timings among six sensors when only 13 or fewer events are detected by each device.

Entropy Analysis. We analyze the events' entropy to determine how many bits of security each inter-event timing provides with different quantization windows ($\mathtt{W}$). This window is used in IoTCupid's encoding to address slight deviations in inter-event timings extracted by different devices. This analysis is required to determine the number of inter-event timings sufficient to securely pair the devices. We found inter-event timings provide enough entropy, even with larger windows, that can be used in IoTCupid's key establishment.

Previous works model event arrivals as a Poisson process, and thus, inter-event timings' probability density function follows a gamma distribution [10, 206, 207]. Hence, we fit a gamma distribution on the inter-event timings to compute their entropies. Figure 5.9 shows the cumulative distribution function (CDF) of the inter-event timings of the events sensed by multiple devices (door events in the smart home and door and coffee events in the smart office). From this, an adversary can uniformly sample inter-event timings from the intervals ($[\mathtt{0}, \mathtt{T}]$) that contain 95% of the timings to establish keys with legitimate devices. In this case, the probability that the adversary successfully guesses the inter-event timing is $\mathtt{W}/\mathtt{T}$. Thus, an inter-event timing's bit security is $\mathtt{log_2(T/W)}$. For instance, the coffee machine events in the smart office provide $\mathtt{log_2(100500/30)} = \mathtt{11.71}$ bits of security when the quantization window is $\mathtt{W} = \mathtt{30}$ secs, and 95% of the inter-event timings fall into an interval of $[\mathtt{0}, \mathtt{T} = \mathtt{100500}]$.

Inter-event Timings Analysis. We present the similarity of inter-event timings computed by IoTCupid from the detected events after event clustering. Our analysis shows that sensors

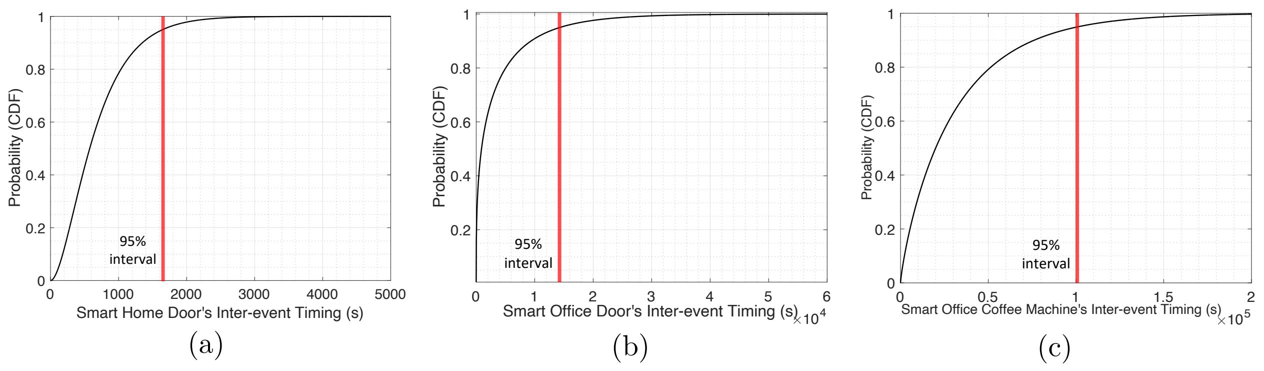


Figure 5.9. CDF of inter-event timings for (a) smart home's door events, (b) smart office's door, and (c) coffee events.

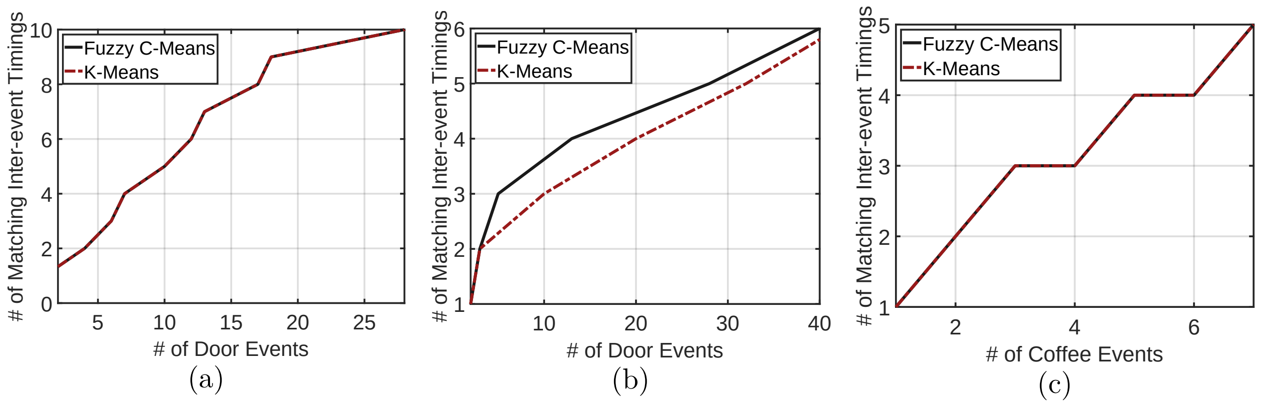


Figure 5.10. # of matching inter-event timings for (a) smart home's door events, (b) smart office's door, and (c) coffee machine events with K-Means and fuzzy C-Means clustering.

influenced by the same event types extract a sufficient number of matching inter-event timings. We determine the number of equivalent inter-event timings from the events sensed by multiple devices (door events in the smart home and door and coffee machine events for the smart office) as they can derive group keys with these timings.

Figure 5.10 shows the number of matching inter-event timings extracted by each sensor with an increasing number of events. We compute the inter-event timings for smart home door events with a quantization window $\mathtt{W} = 8$ seconds to account for the differences in sensor sampling rates. Based on our entropy analysis, for the smart home sensors, only four matching inter-event timings are sufficient to achieve 32-bit password security. This enables four sensors influenced by door events to derive a secure group key with matching inter-event timings from less than 10 detected events.

For the smart office events, we use a larger quantization window $\texttt{W} = \texttt{60}$ seconds due to the larger number of devices and higher ambient noise. Since the door and coffee events in the smart office offer a higher entropy (See Figure 5.9), four similar inter-event timings are sufficient to securely pair the devices despite the larger quantization window. The six smart office sensors sensing the door events establish a group key with only 13 door events. Even though the recall rate of temperature and humidity sensors for door events is relatively low, the correctly detected events are sufficient for these devices to derive the group key. The number of coffee events required to pair the sound sensors and the coffee power meter is even smaller, as shown in Figure 5.10c.

Comparison of K-means with Fuzzy C-Means. Figure 5.10 shows the effect of using K-Means instead of fuzzy clustering in distinguishing event types. K-Means takes longer to extract the same number of matching inter-event timings for door events in the smart office as it clusters concurrent events into separate event types. For instance, generating four matching inter-event timings among the six sensors needs 13 detected door events with fuzzy C-Means clustering while it takes 20 events with K-Means. The inter-event timings generated for the door events in the home and coffee events in the office are similar for the two methods since their timings do not overlap with the other events in the datasets.

Group-to-group Communication. IoTCupid supports group-to-group communication where devices that do not share a key can securely communicate over a common device. In the smart office, sensors are paired into two groups, i.e., the six sensors influenced by the door events and the sound sensors and power meter influenced by the coffee events. Although the power meter is not directly paired with the illuminance, temperature, humidity, and gyroscope sensors, it can still securely communicate with these devices via the sound sensors. With group-to-group communication, multiple devices that belong to various groups can broker communication between devices in different groups. Thus, unlike centralized IoT systems, IoTCupid does not rely on a single device for secure communication.

Table 5.7. Event detection results for malicious devices.

Event Sources	Sensors	Precision	Recall
ⓐ `door-open/close`	Sound	0.0	0.0
	Illuminance	0.0	0.0
	Temperature	0.0	0.0
	Humidity	0.0	0.0
ⓑ `light-on/off`	Illuminance	1.0	0.2
ⓒ `coffee-machine-on`	Sound	0.0	0.0
ⓓ `radiator-on/off`	Temperature	0.0	0.0
	Humidity	0.0	0.0

Advanced attacker devices include a Snowball Black Ice USB microphone, a TSL2560 illuminance sensor and a BME 680 temperature and humidity sensor.

5.5.3 Security Analysis

We evaluate IoTCupid against eavesdropping attacks where an attacker tries to sense the events from outside. We conduct experiments in the smart home by placing sound, illuminance, temperature and humidity sensors outside the front door to simulate attacker devices ($\mathcal{D}_{\mathcal{A}}$). We deploy two types of $\mathcal{D}_{\mathcal{A}}$ equipped with: (a) off-the-shelf sensors with the same capabilities as the smart home devices, and (b) advanced sensors that are more powerful and expensive compared to the inside sensors. Our experiments show that $\mathcal{D}_{\mathcal{A}}$ can only detect events with a precision and recall rate of 0.125 and 0.025 on average. Thus, an attacker is unable to extract sufficient inter-event timings for pairing.

Table 5.7 demonstrates the average event detection results for the two types of $\mathcal{D}_{\mathcal{A}}$. We found that both the normal and advanced $\mathcal{D}_{\mathcal{A}}$'s sound, temperature and humidity sensors cannot correctly detect any of the events that occur inside the home; hence they cannot participate in the pairing protocol. This is because the home walls induce a significant distortion and attenuation in the event signals. We observe that the illuminance sensor senses a limited `light-on/off` events with a precision of 1.0. The light penetrates through the window when the light inside the home is turned on, especially at night when the outside ambient light is too low. Yet, the recall rate of the illuminance sensor is 0.2, lower than the sensors inside the home. This prevents it from extracting sufficient inter-event timings required for pairing with inside sensors.

We note that IoTCupid, by design, offers resiliency to MitM, offline brute-force, denial of key exchange attacks, and key extraction from removed devices. We show in Section 5.3.4 that IoTCupid's partitioned GPAKE scheme offers provable password and group key security against a probabilistic polynomial-time adversary conducting a MitM or brute-force attacks [190]. To launch a denial of key exchange attack, the adversary uses a malicious device to enter the key establishment with random passwords. Yet, legitimate devices still derive their group keys despite this attack because they discard the public keys of malicious devices as they do not share a password. Lastly, a removed device's keys cannot be used to communicate with legitimate devices since the legitimate devices derive new group keys with new inter-event timings in each key update.

5.5.4 Performance Evaluation

Event Detection and Context Extraction Overhead. We evaluate IoTCupid's event detection overhead by measuring the average time for processing and detecting events in each sensor data window. IoTCupid takes, on average, 20.08 secs to pre-process and extract event signals when the window size ($\mathtt{w_s}$) is 2 mins. From the extracted event signals, IoTCupid performs feature extraction in 5.66 ms and inter-event timing extraction in 73.9 ms on average. Since the computation overhead for signal detection and feature extraction is negligible compared to the sensor data window size, it does not impact the overall performance of IoTCupid.

Group Key Establishment Overhead. We evaluate IoTCupid's key establishment overhead, the time between context extraction to deriving group keys, with an increasing number of devices and event types. Figure 5.11 shows the computation time where we run our protocol 100 times and compute the average and standard deviation of the timings. IoTCupid can efficiently pair a large number of devices with inter-event timings extracted from multiple event types. For instance, with 4 event types and 20 devices, the computation overhead is 39.04 ± 13.78 ms ($\approx$ 39M CPU cycles), and with 10 event types and 5 devices, the overhead is

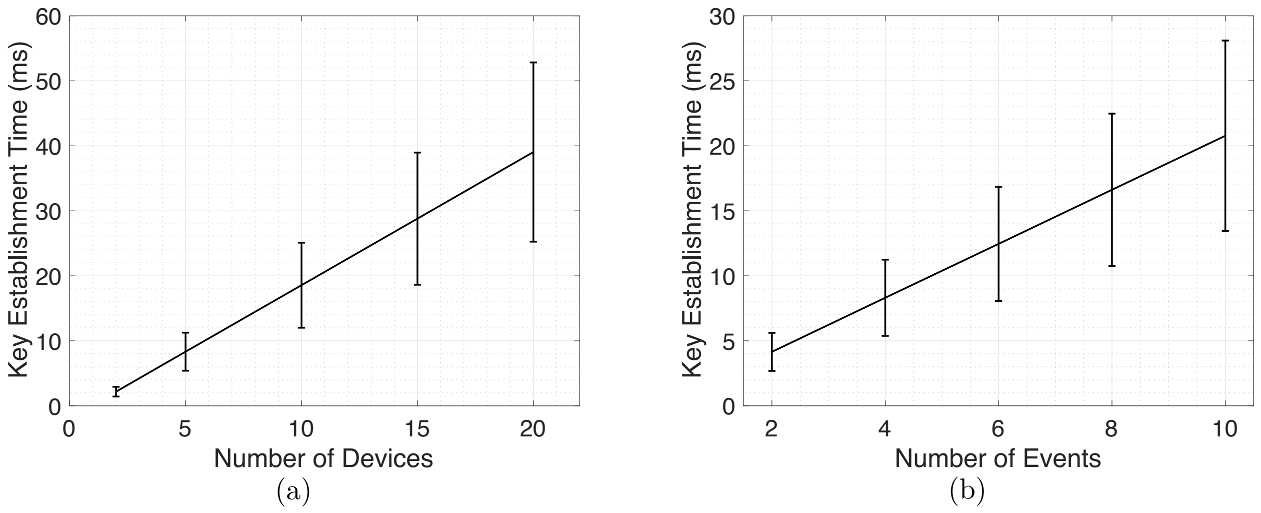


Figure 5.11. Key establishment time overhead with (a) varying number of devices when number of event types is 4, (b) varying number of event types when number of devices is 5.

20.77 ± 7.32 ms ($\approx 20.8\mathtt{M}$ CPU cycles). The overhead increases linearly with the number of devices (See Figure 5.11a) and event types (See Figure 5.11b). This is because the communication rounds and elliptic curve scalar multiplications dominate the time overhead, and their number increases linearly.

Memory Usage. IoTCupid requires each device to temporarily store the data received from other devices during the group key establishment protocol's communication rounds. This corresponds to $96 * X * Y$ Bytes, where X is the number of event types and Y is the number of devices. For instance, when there are 100 devices that sense 10 event types, the memory usage of each device is 93.75 KB, which is acceptable even for low-end IoT devices.

Encryption and Communication Cost. We evaluate IoTCupid's encryption and communication cost with a varying number of devices (See Section 5.5.5 for comparison with individual keys). We run the encryption scheme for a 32-byte message 100 times. We observe IoTCupid's secure channels incur 0.21 ± 0.005 ms ($\approx 0.2\mathtt{M}$ CPU cycles). Figure 5.12 shows that IoTCupid's computation overhead and communication cost are constant with an increasing number of devices as the devices communicate with a group key.

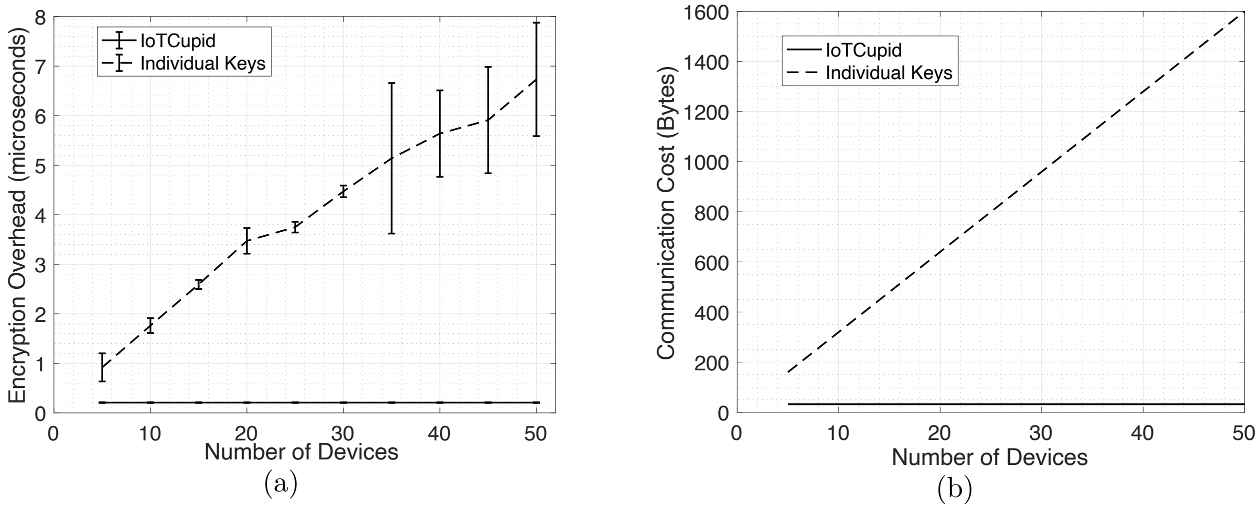


Figure 5.12. (a) Encryption and (b) communication cost with IoTCupid's group keys and Perceptio's individual keys.

We also evaluate IoTCupid's group-to-group communication cost when devices that do not share a key securely communicate over a common device that they share a key with. This requires one additional decryption and encryption operation, which takes 0.42 ± 0.007 ms.

5.5.5 Comparison with Prior Work

Among existing pairing approaches, only two works, T2Pair [12] and Perceptio [10], can pair heterogeneous IoT devices. T2Pair requires users to swipe screens or press buttons on devices and uses the timings of these actions as evidence for pairing. Given the need for active user involvement, quantitatively comparing T2Pair with IoTCupid is infeasible. Therefore, we discuss its strengths and weaknesses in comparison to IoTCupid in Section 5.7.

Perceptio [10], similar to IoTCupid, leverages inter-event timings from various sensing modalities as evidence of co-presence to pair devices. It detects events for only instantly influenced sensors and leverages a fuzzy commitment scheme to establish individual keys among devices with similar inter-event timings. Below, we quantitatively compare IoTCupid with Perceptio.

Paired Devices. We compare the number of devices paired using IoTCupid and Perceptio in our evaluation setup. Perceptio can only pair 7 out of the 11 devices (63%) in the two IoT environments while IoTCupid pairs all of them. Perceptio can only pair instant sensors but cannot detect events for continuously influenced sensors since it cannot capture the gradual changes in sensor values (as described in Section 5.3.2). For instance, Perceptio cannot pair the temperature and humidity sensors with the other devices even though they commonly sense the door events. In contrast, IoTCupid detects the events for both instantaneously and continuously influenced sensors and pairs all six devices in the smart office influenced by the door events.

Pairing Time. To compare IoTCupid and Perceptio's pairing time, we assume the time required to extract inter-event timings is same for both systems and compare their entropy bits required to defend against offline attacks. Since Perceptio relies on fuzzy commitment, it requires 128-bit entropy whereas IoTCupid requires 32-bit entropy due to its resilience to offline brute-force attacks. Thus, IoTCupid requires 4× less matching inter-event timings, resulting in 4× faster pairing compared to fuzzy commitment-based pairing protocols. Moreover, IoTCupid extracts correct inter-event timings using a fewer number of events. Particularly, for smart office's door events, IoTCupid extracts four matching inter-event timings from 13 occurred events whereas Perceptio requires 20. This translates into 54% faster pairing with IoTCupid.

Secure Communication Cost. We compare IoTCupid's computation and communication cost with Perceptio's when a device aims to broadcast a single 32-Byte message. IoTCupid incurs a constant overhead, whereas Perceptio's overhead linearly increases due to its pairwise individual keys [10] (Figure 5.12). Particularly, a device needs to encrypt the message one by one with all the individual keys and then send it to the other devices individually. For instance, broadcasting a message to 50 devices takes on average 0.21 ms and requires transmitting 32-Byte with IoTCupid's group key, while it takes on average 6.73 ms and requires transmitting 1600-Byte with Perceptio's individual keys. We note that linear overhead is inherent in any pairing protocol that establishes pairwise individual keys. On the contrary, with IoTCupid's group keys, the device encrypts the message only once with the group key and broadcasts it.

5.6 Limitations and Discussion

Handling Mismatches in Inter-event Timings. IoTCupid requires concatenating four inter-event timings in a password to provide enough entropy for group key establishment. However, inter-event timings of sensors that sense the same event may not always match (e.g., due to a sensor missing an event). This would create discrepancies in the passwords and prevent them from establishing a key.

To address such mismatches, we initially considered integrating a private set intersection (PSI) protocol [208] into IoTCupid for devices to determine which inter-event timings they should use for their passwords. However, since the universal set of possible inter-event timings is small (e.g., 2^8-2^{12}), an adversary can enter the PSI protocol with many inter-event timings to learn the benign devices' timings. Thus, we let devices enter our group key establishment protocol with all combinations of their inter-event timings, allowing them to use the matching ones to derive keys. We set an upper limit ($\mathtt{n_p}$) on the number of inter-event timings the device should extract before entering key establishment to ensure the number of combinations does not hurt the protocol's scalability. For instance, when $\mathtt{n_p} = \mathtt{10}$, the number of devices is 20, and the number of event types is 4, it takes, on average, $\approx$ 8 secs to run our key establishment protocol.

Deployment Considerations. IoTCupid uses window-based pre-processing and sensor thresholds for event detection and identifies the optimal parameters for clustering via statistical methods [33]. In practice, IoTCupid's calibration for determining these parameters can be performed in two ways: (a) offline by device manufacturers or (b) online by IoT service provider at the time of device installation. For the offline calibration, device manufacturers may calibrate sensors by (1) generating commonly occurring events in IoT deployments or (2) using publicly available smart home datasets [177, 178, 209–211] that include different sensors' measurements corresponding to common events.

We show in Section 5.5.1 that parameters extracted from a publicly available dataset [177] are transferable across IoT deployments. Yet, some IoT environments may include unique event types or may be exposed to environmental disturbances, distinct from typical IoT environments. In such cases, the manufacturer determined parameters may require fine-tuning for the specific deployment. For this, calibration can be initiated by IoT service providers by recording timestamps of various events occurring in the IoT deployment for a given amount of time (a few hours is sufficient, detailed in Section 5.5.1). Both of these calibration methods do not require any information about the event types. Each device only needs the sensor data and the timestamps at which events occurred to determine its parameters.

Resourceful Attackers. IoTCupid is resilient against normal and advanced eavesdropping attackers (Section 5.5.3). Yet, an attacker may have access to devices with asymmetric capabilities (e.g., x-ray vision). We do not consider such attackers as they could already visualize and reveal users' private activities, independent of our pairing protocol. Moreover, an outside attacker may attempt to inject signals to the inside sensors to pair with them or disrupt the pairing process. For this, the attacker may use electromagnetic interference (EMI), acoustic injection, and inaudible voice attacks [212–214]. Yet, such attacks require a high amplitude signal, which is difficult to achieve since outside signals experience a high attenuation from the walls (as shown in Section 5.5.3). Besides, these attacks can be identified by anomaly detection and prevented by shielding techniques [215–218].

Pairing in Large Spaces. We demonstrate in Section 5.5.1 that IoTCupid can successfully pair devices at a distance of up to 5m. In large indoor spaces, some devices may be located far away from event sources and may not be able to sense the same events as other devices and establish the same group keys. However, such devices may share group keys with common devices (e.g., a nearby device paired with far away devices) or there may exist transient devices that have a view of different areas of the room (e.g., an illuminance sensor in a dining area may view both the kitchen and living room) and can sense the events occurring in each. These devices can then use the inter-event timings of the commonly observed events as evidence and act as a bridge between the groups in two distant areas for secure communication.

Rarely/Regularly Occurring Events. Although a few frequent events (e.g., `door-open`) commonly sensed by sensors are enough to establish group keys, some sensors may only detect rarely occurring events (e.g., a laundry washer's power meter). Such rarely occurring events would cause longer pairing times. Devices equipped with such sensors would need additional sensors (e.g., a microphone) that measure diverse events to timely pair with other devices.

Contrarily, some events may regularly occur in an IoT environment (e.g., smart home door opening at 9 am every day) and may be predictable by attackers. However, it is extremely difficult to predict successive timestamps of such events at a fine granularity. Thus, given that IoTCupid's group key establishment concatenates multiple inter-event timings of a given event type as evidence of co-presence and event detection accuracy is very low for attacker devices (Section 5.5.3), an attacker cannot extract evidences matching with the legitimate devices.

Impact of Environmental Noise. We develop a signal threshold-based event detection approach where we subtract the sensor readings' mean value in the preceding window for each window and compute the absolute difference before applying a smoothing filter for noise removal (Detailed in Section 5.3.2). This allows us to accurately detect events even with varying environmental noise. In rare cases, environmental noise's impact might be significantly higher than an event's physical influence, eliminating the event's impact on the sensor readings. Yet, this limitation is present in all existing systems that rely on sensing physical processes.

5.7 Related Work

Human-in-the-loop-based Pairing. Initial methods leverage mobile phone cameras and 2D barcodes to establish keys [219]. Mayrhofer et al. propose pairing the devices with a user simultaneously shaking them [11]. Move2Auth requires users to perform hand gestures by holding their smartphones in front of the devices and uses the variations in received signal strength for pairing [172]. Tap2Pair pairs devices through a user synchronously tapping on a device following the patterns displayed on the other device [173]. T2Pair needs users to apply operations such as pressing a button and swiping a touchscreen, and uses timestamps as a source of entropy [12]. SenCS pairs devices with mobile phones carried by users using the entropy from their actions (e.g., walking) [220]. These schemes need human involvement which only allows pairwise device pairing, incurring a huge manual effort from users with an increasing number of devices. Automating these user actions would require specialized equipment (e.g., robotic arms); thus, impractical for typical IoT environments.

Table 5.8. Comparison of IoTCupid with context-aware pairing schemes for IoT devices.

Pairing Scheme	Sensing Modality	Concurrent Events	Group Pairing	Continuous Sensors
Schurmann et al. [176]	Ambient sound	✗	✗	✗
Mathur et al. [221]	Wireless signal	✗	✗	✗
Miettinen et al. [14]	Ambient sound or light	✗	✗	✗
Rostami et al. [175]	Heart beat	✗	✗	✗
FastZIP [15]	Accelerometer, Gyroscope, Barometer	✗	✗	✗
Perceptio [10]	Heterogeneous sensors	✗	✗	✗
IoTCupid	Heterogeneous sensors	✓	✓	✓

Context-aware Pairing. To address the limitations of above approaches, context-aware pairing schemes have been proposed. Table 5.8 compares IoTCupid with several of them.

Schurrman et al. leverage audio context [176] and Miettinen et al. [14] use fingerprints from sound and luminosity to pair sensors. Mathur et al. [221] use similarities in the temporal variations in wireless channels between two nearby wireless devices as evidence. Rostami et al. [175] use the entropy extracted from heart beat signals of patients to pair implantable medical devices with their controllers. FastZIP [15] uses sensor fusion to construct fingerprints of shared context for intra-car device pairing with a Fuzzy PAKE scheme. Yet, these schemes do not support heterogeneous sensor modalities and therefore, their use cases are limited.

Similar to IoTCupid, Perceptio [10] uses inter-event timings from heterogeneous sensing modalities as evidence for co-presence to pair devices. Unfortunately, it does not support pairing continuously influenced sensors (e.g., temperature and humidity), does not support concurrent events, and only establishes individual keys. This results in longer pairing times to establish the keys, and linear storage, computation, and communication overhead after the keys are established. IoTCupid addresses these limitations, providing a secure and practical group pairing solution.

5.8 Sensor Data Pre-processing

Figure 5.13 shows an example of the temperature sensor signal before and after IoTCupid's pre-processing during event detection. As a result, the signal becomes smoother and amenable for event detection.

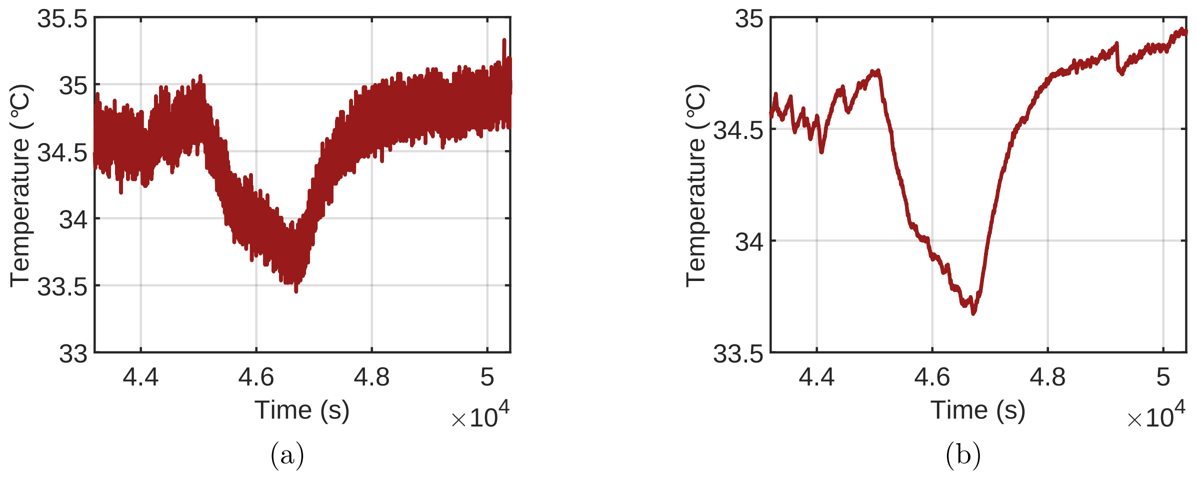


Figure 5.13. Sensor data (a) before and (b) after pre-processing.

5.9 Partitioned GPAKE Security Analysis

Following [190], we provide a formal security analysis of the partitioned GPAKE protocol. We first define two security properties, key secrecy and password-privacy. *Key secrecy* means that assuming the passwords (evidences) are distributed uniformly at random and only a constant number of passwords can be checked by the adversary on each online attack, the probability that the adversary can derive a group key with legitimate devices is negligible. *Password-privacy* ensures that an adversary that conducts an online attack cannot gain any information on the passwords used by legitimate devices, including which devices actually share the same password. We provide a proof sketch below, and refer to [190] for the full proof. We note that we reduce our protocol's security to computational elliptic curve Diffie-Hellmann (ECDH) instead of traditional computational DH in [190], although the reductions remain the same.

Theorem 5.9.1. *Let the encryption scheme in Table 5.2 be both unforgeable and chosen plaintext semantically-secure. Then, the protocol in Table 5.2 is a correct partitioned group password-authenticated key exchange scheme that achieves key secrecy and password-privacy under the elliptic curve computational Diffie-Hellman assumption in the random oracle and ideal cipher model.*

Proof Sketch. Correctness. In an honest execution of the partitioned GPAKE protocol where no adversaries are involved, it is straightforward to verify that all devices that share the same password derive the same session ids and a shared group key. This follows from the fact that the devices that share the same password first derive session keys with each other, and then use these session keys to broadcast a random value. The devices with the same session key can decrypt the random values and add them to derive the group key. All the devices that share the same password conduct these steps, and thus, they all add the same random values shared among them, deriving a correct group key.

Key Secrecy. An adversary can target different stages of the protocol to gain information about the group keys.

First, we consider an adversary ($\mathcal{A}$) who has a valid tuple (d_{i}, d_{j}, $X_{\text{i},d_{\text{i}}}$, $X_{\text{j},d_{\text{j}}}$, $x_{\text{i},d_{\text{i}}} \cdot X_{\text{j},d_{\text{j}}}$ $mod\ q$) in the group key establishment protocol. Such an adversary can derive the same group key with legitimate devices as it has a valid session key. However, we show that if such an adversary exists, we can construct another adversary ($\mathcal{B}$) that can break the computational elliptic curve Diffie-Hellman (ECDH) assumption. Particularly, given the input of $x \cdot P$ and $y \cdot P$, $\mathcal{B}$'s goal is to derive $x \cdot (y \cdot P)$. For this, $\mathcal{B}$ picks two random user indices (i and j) and a random execution number. It next sets the i^{th} device's $X_{\text{i},d_{\text{i}}}$ as $x \cdot P$ and j^{th} device's $X_{\text{j},d_{\text{j}}}$ as $y \cdot P$. $\mathcal{B}$ then uses $\mathcal{A}$ as a subroutine and returns $x_{\text{i},d_{\text{i}}} \cdot X_{\text{j},d_{\text{j}}}$ $mod\ q$. If $\mathcal{B}$ guesses the random user indices and the execution number correctly, it correctly breaks the computational ECDH problem and derives $x \cdot (y \cdot P)$. Therefore, the security of the session key of the partitioned GPAKE protocol reduces to the hardness of the computational ECDH problem.

We next consider an adversary who guesses a password correctly. Such an adversary can participate in the protocol and derive the same keys with legitimate devices by following an honest execution of the protocol. However, assuming the password's are uniformly distributed, the probability of such an adversary existing is $\frac{q}{2^{\|\text{pw}\|}}$ where q is the number of times the adversary can guess a password in an online attack and *pw* is the bit-length of the password. Therefore, if the password has a sufficient bit-length, the probability of such an attack is negligible.

Lastly, we consider an adversary who modifies the messages in the broadcast stages of the protocol such that it crafts messages that can decrypt correctly by legitimate devices to derive shared keys with them. Yet, existence of such an adversary reduces to the unforgeability of the encryption scheme used.

Password-privacy. The proof for password-privacy is very similar to the one for key secrecy. This is because after the devices encrypt their public keys using their passwords with an unforgeable and CCA-secure encryption scheme, the protocol messages become independent of the passwords. Therefore, a polynomial-time adversary who has no prior knowledge about the passwords cannot learn any information about them from the protocol under the hardness of the computational ECDH problem and the unforgeability of the encryption scheme.

□

Following this proof, the security of IoTCupid relies on (1) the randomness of the passwords, (2) the hardness of the computational elliptic curve Diffie-Hellman problem, and (3) the security of the encryption scheme used in GPAKE. We show in our evaluation that the inter-event timings provide enough entropy to be used as passwords. The security of the computational ECDH problem and used symmetric encryption scheme are already well-established.

6. CONCLUSIONS AND FUTURE WORK

In this thesis, I presented my studies on how mobile sensors pose privacy threats to users and then how we can leverage these sensors to improve the security of these devices through usable user and device authentication methods.

Chapter 2 introduced a side-channel attack through motion sensors by exploiting the vulnerability introduced due to the introduction of embedded magnets in stylus pencils. We presented $\mathtt{S}^3$, a novel system that infers what a user is writing from motion sensors' data on an iPad Pro using the latest version of the Apple Pencil. To track the Pencil movement, we developed a novel multi-dimensional particle filtering algorithm using a 3D magnetic map of the Pencil to identify the magnetic impact of different locations and orientations of the Pencil. We evaluated $\mathtt{S}^3$ with 10 subjects and demonstrated that an attacker could identify 93.9%, 96%, 97.9%, and 93.33% of the letters, numbers, shapes, and words correctly by only using the motion sensors' data. In future work, we will expand our analysis to support more devices that use stylus pencils with embedded magnets to find potential privacy leaks.

In chapter 3, we present `LocIn`, a new location inference attack on mixed reality (MR) devices via 3D spatial data. `LocIn` exploits the 3D spatial maps accessible to MR apps to extract contextual patterns from a user's environment and infer their location. It leverages a multi-task learning approach to train an end-to-end encoder-decoder architecture that integrates these contextual patterns into a classification network for predicting users' location. Our evaluation on spatial maps collected from three MR devices demonstrates that `LocIn` can effectively infer an MR user's location type. Future work will explore `LocIn` attack's transferability across various MR devices that leverage different depth sensors and develop countermeasures that optimize the spatial maps' utility and privacy.

In chapter 4, we presented a secure liveness detection system, `FaceRevelio`, that uses a single smartphone camera with no extra hardware. `FaceRevelio` uses the smartphone screen to illuminate the human face from various directions via a random light passcode. The reflections of these light patterns from the face are recorded to construct the 3D surface of the face. This is used to detect if the authentication subject is a human or not. `FaceRevelio` achieves a mean EER 1.4% and 0.15% against photo and video replaying attacks, respectively.

While `FaceRevelio` focuses on spoofing attacks via photos and videos, future work will investigate liveness detection methods against generative spoofing attempts (e.g., adversarial examples simulating 3D human face and its depth features) and 3D masks.

In chapter 5, we introduced `IoTCupid`, a secure group pairing system for heterogeneous devices, without requiring active user involvement. `IoTCupid` exploits the fact that multiple co-located sensors sense the same events and the time between subsequent event occurrences sensed by different sensors is similar. `IoTCupid` pairs both instantly and continuously influenced sensors, supports distinguishing concurrent events for context extraction and establishes group keys from inter-event timings with partitioned GPAKE. We evaluated `IoTCupid` on two IoT environments, a smart home and a smart office, and showed that it can pair devices with diverse sensing modalities with minimal overhead. `IoTCupid` is an important step forward in automated and practical secure device pairing in IoT deployments. Future efforts will focus on developing automated pairing approaches for public IoT environments with no physical security guarantees.

6.1 Future Research Directions

6.1.1 Privacy Preserving Sensing Systems

With new computing devices emerging and frequent software updates, privacy leakages through sensors continue to be a serious threat to users' security and privacy. This thesis has highlighted a few of these privacy leakages through sensors on existing mobile devices. However, users' awareness and understanding of sensor-based privacy threats still remain largely unexplored. In the short term, I plan to conduct large-scale studies to gauge users' awareness of the private data collected by sensors and their perception of the associated threats. For example, I seek to answer questions such as do mobile users understand which of their activities impact motion sensors and whether they think existing mobile permissions control apps' access to this data.

Based on the findings about users' understanding and perception, in the long term, I aim to develop sensing systems that leverage sensor data to enable useful applications while maintaining users' privacy. Existing solutions for sensor-based privacy leakages focus on limiting apps' access to sensor data or data modification techniques (e.g., noise injection) without considering their impact on this data's utility for useful applications. Developing privacy and utility-preserving sensing systems requires a careful in-depth analysis of desired applications of mobile sensor data and the design of machine learning techniques that transform sensor data while optimizing its utility.

6.1.2 Cross-Device Security and Privacy

My prior research has focused on improving the security of interactions between mobile and IoT devices. I plan to extend my insights on secure and usable authentication to emerging computing platforms. For instance, traditional user authentication and pairing mechanisms are not feasible for emerging mixed reality or intermittent computing devices due to their unique interfaces and energy constraints. In the short term, I seek to investigate the unique challenges introduced by these devices and address them to improve the usability and security of authentication and communication mechanisms for AR/VR and energy-constrained devices.

While my current research has focused on individual devices or single components of mobile systems, in the long term, I strive to evaluate the security and privacy of multi-device systems. We now live in a world where users simultaneously interact with smartphones, wearables, IoT devices, and even autonomous vehicles while these devices interact with each other. This interconnectivity of various systems creates unique security and privacy challenges that must be addressed presently. For example, without unified access control, an app on a smartphone and smartwatch connected to a car may access a user's location even if the location permission is not granted on the phone and watch. I plan to leverage multi-modal sensing and my skills in computer vision and machine learning to design context-aware privacy solutions for multi-device systems and verify their usability for end-users.

6.1.3 Digital Safety for Diverse Populations

I also aim to expand this thesis to a broad investigation of the societal impacts of emerging computing platforms. As access to technology is becoming widespread, understanding the privacy needs and challenges of users with diverse social and cultural backgrounds is imperative for developing privacy solutions for all. My work on understanding the impact of technology on the security and privacy of refugees is the first step in this direction. I plan to expand my work by investigating the security and privacy of marginalized and vulnerable populations from a cross-culture perspective. In the future, I aim to conduct research that brings more diverse perspectives into security and privacy scholarship. For instance, most of the existing security research relies on the assumption that each device has a single user. This assumption is invalid for many developing countries where an entire family may share a single device. To this end, I will develop privacy mechanisms that are responsive to local and global constraints and explore methods to improve privacy legislation to account for potential privacy issues across regions and cultures.

REFERENCES

[1] A. K. Sikder, G. Petracca, H. Aksu, T. Jaeger, and A. S. Uluagac, "A survey on sensor-based threats and attacks to smart devices and applications," *IEEE Communications Surveys & Tutorials*, 2021.

[2] P. Delgado-Santos, G. Stragapede, R. Tolosana, R. Guest, F. Deravi, and R. Vera-Rodriguez, "A survey of privacy vulnerabilities of mobile device sensors," *ACM Computing Surveys (CSUR)*, 2022.

[3] H. Wang, T. T.-T. Lai, and R. Roy Choudhury, "Mole: Motion leaks through smart-watch sensors," in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, ACM, 2015, pp. 155–166.

[4] M. Mehrnezhad, E. Toreini, S. F. Shahandashti, and F. Hao, "Touchsignatures: Identification of user touch actions and pins based on mobile sensor data via javascript," *CoRR*, vol. abs/1602.04115, 2016. arXiv: 1602.04115. [Online]. Available: http://arxiv.org/abs/1602.04115.

[5] A. Maiti, O. Armbruster, M. Jadliwala, and J. He, "Smartwatch-based keystroke inference attacks and context-aware protection mechanisms," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS '16, Xi'an, China: ACM, 2016, pp. 795–806, ISBN: 978-1-4503-4233-9. DOI: 10.1145/2897845.2897905. [Online]. Available: http://doi.acm.org/10.1145/2897845.2897905.

[6] E. Owusu, J. Han, S. Das, A. Perrig, and J. Zhang, "Accessory: Password inference using accelerometers on smartphones," in *Proceedings of the Twelfth Workshop on Mobile Computing Systems & Applications*, ACM, 2012, p. 9.

[7] R. Ning, C. Wang, C. Xin, J. Li, and H. Wu, "Deepmag: Sniffing mobile apps in magnetic field through deep convolutional neural networks," in *2018 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, Mar. 2018, pp. 1–10.

[8] N. Matyunin, Y. Wang, T. Arul, J. Szefer, and S. Katzenbeisser, "Magneticspy: Exploiting magnetometer in mobile devices for website and application fingerprinting," *arXiv preprint arXiv:1906.11117*, 2019.

[9] D. Tang, Z. Zhou, Y. Zhang, and K. Zhang, "Face flashing: A secure liveness detection protocol based on light reflections," *arXiv preprint arXiv:1801.01949*, 2018.

[10] J. Han *et al.*, "Do you feel what i hear? enabling autonomous IoT device pairing using different sensor types," in *IEEE Symposium on Security and Privacy (S&P)*, 2018.

[11] R. Mayrhofer and H. Gellersen, "Shake well before use: Intuitive and secure pairing of mobile devices," *IEEE Transactions on Mobile Computing*, 2009.

[12] X. Li, Q. Zeng, L. Luo, and T. Luo, "T2pair: Secure and usable pairing for heterogeneous IoT devices," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2020.

[13] M. Fomichev, F. Álvarez, D. Steinmetzer, P. Gardner-Stephen, and M. Hollick, "Survey and systematization of secure device pairing," *IEEE Communications Surveys & Tutorials*, 2017.

[14] M. Miettinen, N. Asokan, T. D. Nguyen, A.-R. Sadeghi, and M. Sobhani, "Context-based zero-interaction pairing and key evolution for advanced personal devices," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2014.

[15] M. Fomichev, J. Hesse, L. Almon, T. Lippert, J. Han, and M. Hollick, "Fastzip: Faster and more secure zero-interaction pairing," in *International Conference on Mobile Systems, Applications, and Services*, 2021.

[16] H. Farrukh, T. Yang, H. Xu, Y. Yin, H. Wang, and Z. B. Celik, "S3: Side-channel attack on stylus pencil through sensors," *Interactive, Mobile, Wearable and Ubiquitous Technologies (UbiComp)*, 2021.

[17] H. Farrukh, R. M. Aburas, A. Nare, A. Bianchi, and Z. B. Celik, "Inferring semantic location from spatial maps in mixed reality," in *USENIX Security Symposium*, 2023.

[18] H. Farrukh, R. M. Aburas, S. Cao, and H. Wang, "Facerevelio: A face liveness detection system for smartphones with a single front camera," in *Proceedings of the International Conference on Mobile Computing and Networking (MobiCom)*, 2020.

[19] H. Farrukh, M. O. Ozmen, F. K. Ors, and Z. B. Celik, "One key to rule them all: Secure group pairing for heterogeneous iot devices," in *IEEE Symposium on Security and Privacy (SP)*, 2022.

[20] "Use apple pencil to enter text in any text field." (), [Online]. Available: https://support.apple.com/en-gb/guide/ipad/ipad355ab2a7/ipados.

[21] P. Zarchan, H. Musoff, A. I. of Aeronautics, and Astronautics, *Fundamentals of Kalman Filtering: A Practical Approach* (Progress in astronautics and aeronautics). American Institute of Aeronautics and Astronautics, Incorporated, 2000, ISBN: 9781563473999. [Online]. Available: https://books.google.com/books?id=AQxRAAAAMAAJ.

[22] E. Bostan, P. D. Tafti, and M. Unser, "A dual algorithm for L1-regularized reconstruction of vector fields," in *IEEE International Symposium on Biomedical Imaging (ISBI)*, IEEE, 2012.

[23] T. Chow, *Introduction to Electromagnetic Theory: A Modern Perspective*. Jones and Bartlett Publishers, 2006, ISBN: 9780763738273. [Online]. Available: https://books.google.com/books?id=dpnpMhw1zo8C.

[24] J. Hightower and G. Borriello, "Particle filters for location estimation in ubiquitous computing: A case study," in *UbiComp 2004: Ubiquitous Computing*, N. Davies, E. D. Mynatt, and I. Siio, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 88–106, ISBN: 978-3-540-30119-6.

[25] J. Diebel, "Representing attitude: Euler angles, unit quaternions, and rotation vectors," 2006.

[26] D. Fox, "Kld-sampling: Adaptive particle filters," in *Advances in neural information processing systems*, 2002, pp. 713–720.

[27] "Celestial coordinates." (), [Online]. Available: http://spiff.rit.edu/classes/phys373/lectures/radec/radec.html#altaz.

[28] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, 2000.

[29] X. X. Lu, "A review of solutions for perspective-n-point problem in camera pose estimation," in *Journal of Physics: Conference Series*, IOP Publishing, vol. 1087, 2018, p. 052 009.

[30] J. C. Maxwell, "Viii. a dynamical theory of the electromagnetic field," *Philosophical transactions of the Royal Society of London*, no. 155, pp. 459–512, 1865.

[31] P. D. Tafti and M. Unser, "On regularized reconstruction of vector fields," *IEEE Transactions on Image Processing*, vol. 20, no. 11, pp. 3163–3178, 2011. DOI: 10.1109/TIP.2011.2159230.

[32] R. Mukundan, "Quaternions: From classical mechanics to computer graphics, and beyond," Jan. 2002.

[33] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[34] S. Kullback and R. A. Leibler, "On information and sufficiency," *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, 1951, ISSN: 00034851. [Online]. Available: http://www.jstor.org/stable/2236703.

[35] "Word frequency data." (), [Online]. Available: https://www.wordfrequency.info/.

[36] J. Howard and S. Ruder, *Universal language model fine-tuning for text classification*, 2018. arXiv: 1801.06146 [cs.CL].

[37] G. R. Scott, "Magnetic shielding," in *Encyclopedia of Geomagnetism and Paleomagnetism*, D. Gubbins and E. Herrero-Bervera, Eds. Dordrecht: Springer Netherlands, 2007, pp. 540–542, ISBN: 978-1-4020-4423-6. DOI: 10.1007/978-1-4020-4423-6_183. [Online]. Available: https://doi.org/10.1007/978-1-4020-4423-6_183.

[38] Y. Michalevsky, D. Boneh, and G. Nakibly, "Gyrophone: Recognizing speech from gyroscope signals," in *23rd USENIX Security Symposium (USENIX Security 14)*, 2014, pp. 1053–1067.

[39] Z. Ba *et al.*, "Learning-based practical smartphone eavesdropping with built-in accelerometer," in *Proceedings of the Network and Distributed Systems Security (NDSS) Symposium*, 2020.

[40] A. Das, N. Borisov, and M. Caesar, "Tracking mobile web users through motion sensors: Attacks and defenses.," in *NDSS*, 2016.

[41] S. Biedermann, S. Katzenbeisser, and J. Szefer, "Hard drive side-channel attacks using smartphone magnetic field sensors," in *International Conference on Financial Cryptography and Data Security*, Springer, 2015, pp. 489–496.

[42] K. Block and G. Noubir, "My magnetometer is telling you where i've been?: A mobile device permissionless location attack," in *Proceedings of the 11th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, ACM, 2018, pp. 260–270.

[43] B. Perez, M. Musolesi, and G. Stringhini, "Fatal attraction: Identifying mobile devices through electromagnetic emissions," in *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks*, ACM, 2019, pp. 163–173.

[44] H. Jiang, "Motion eavesdropper: Smartwatch-based handwriting recognition using deep learning," in *2019 International Conference on Multimodal Interaction*, ser. ICMI '19, Suzhou, China: Association for Computing Machinery, 2019, pp. 145–153, ISBN: 9781450368605. DOI: 10.1145/3340555.3353740. [Online]. Available: https://doi.org/10.1145/3340555.3353740.

[45] Q. Xia, F. Hong, Y. Feng, and Z. Guo, "Motionhacker: Motion sensor based eavesdropping on handwriting via smartwatch," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2018, pp. 468–473. DOI: 10.1109/INFCOMW.2018.8406879.

[46] K.-Y. Chen, S. N. Patel, and S. Keller, "Finexus: Tracking precise motions of multiple fingertips using magnetic sensing," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, ACM, 2016, pp. 1504–1514.

[47] S. H. Yoon, K. Huo, and K. Ramani, "Tmotion: Embedded 3d mobile input using magnetic sensing technique," in *Proceedings of the TEI'16: Tenth International Conference on Tangible, Embedded, and Embodied Interaction*, ACM, 2016, pp. 21–29.

[48] Y. Liu, K. Huang, X. Song, B. Yang, and W. Gao, "Maghacker: Eavesdropping on stylus pen writing via magnetic sensing from commodity mobile devices," in *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, 2020, pp. 148–160.

[49] F. Chollet *et al.* "Keras." (2015), [Online]. Available: https://github.com/fchollet/keras.

[50] "Leave-one-out cross-validation," in *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds. Boston, MA: Springer US, 2010, pp. 600–601, ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_469. [Online]. Available: https://doi.org/10.1007/978-0-387-30164-8_469.

[51] C. E. Hughes, C. B. Stapleton, D. E. Hughes, and E. M. Smith, "Mixed reality in education, entertainment, and training," *IEEE computer graphics and applications*, 2005.

[52] J. Gerup, C. B. Soerensen, and P. Dieckmann, "Augmented reality and mixed reality for healthcare education beyond surgery: An integrative review," *International journal of medical education*, 2020.

[53] B. John and N. Wickramasinghe, "A review of mixed reality in health care," *Delivering Superior Health and Wellness Management with IoT and Analytics*, 2020.

[54] *Ikea place*, https://apps.apple.com/us/app/ikea-place/id1279244498, [Online; accessed 01-May-2023], 2023.

[55] *Apple unveils new ipad pro with breakthrough lidar scanner*, https://www.apple.com/newsroom/2020/03/apple-unveils-new-ipad-pro-with-lidar-scanner-and-trackpad-support-in-ipados/, [Online; accessed 01-May-2023], 2023.

[56] *Mixed reality market*, https://www.fortunebusinessinsights.com/industry-reports/mixed-reality-market-101783, [Online; accessed 01-May-2023], 2023.

[57] J. A. d. Guzman, K. Thilakarathna, and A. Seneviratne, "A first look into privacy leakage in 3d mixed reality data," in *European Symposium on Research in Computer Security*, 2019.

[58] N. Wu, R. Cheng, S. Chen, and B. Han, "Preserving privacy in mobile spatial computing," in *Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2022.

[59] *Mrtk*, https://github.com/Microsoft/MixedRealityToolkit-Unity, [Online; accessed 01-May-2023], 2023.

[60] *Arcore*, https://developers.google.com/ar/, [Online; accessed 01-May-2023], 2023.

[61] *Arkit*, https://developer.apple.com/augmented-reality/, [Online; accessed 01-May-2023], 2023.

[62] D. Du, L. Wang, H. Wang, K. Zhao, and G. Wu, "Translate-to-recognize networks for rgb-d scene recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[63] L. Herranz, S. Jiang, and X. Li, "Scene recognition with cnns: Objects, scales and dataset bias," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[64] S. Liu, G. Tian, and Y. Xu, "A novel scene classification model combining resnet based transfer learning and data augmentation with a filter," *Neurocomputing*, 2019.

[65] A. Wang, J. Cai, J. Lu, and T.-J. Cham, "Modality and component aware feature fusion for rgb-d scene classification," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[66] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva, "Learning deep features for scene recognition using places database," *Advances in neural information processing systems*, 2014.

[67] H. Zhu, J.-B. Weibel, and S. Lu, "Discriminative multi-modal feature fusion for rgbd indoor scene recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[68] J. A. d. Guzman, A. Seneviratne, and K. Thilakarathna, "Unravelling spatial privacy risks of mobile mixed reality data," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2021.

[69] C. Moenning and N. A. Dodgson, "Fast marching farthest point sampling," University of Cambridge, Computer Laboratory, Tech. Rep., 2003.

[70] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *Advances in neural information processing systems*, 2017.

[71] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. NieSSner, "Scannet: Richly-annotated 3d reconstructions of indoor scenes," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[72] G. Baruch *et al.*, "Arkitscenes: A diverse real-world dataset for 3d indoor scene understanding using mobile rgb-d data," in *Conference on Neural Information Processing Systems*, 2021.

[73] *Snap ar*, https://ar.snap.com/en-US/lens-studio, [Online; accessed 01-May-2023], 2023.

[74] *Pokemon go*, https://pokemongolive.com/en/, [Online; accessed 01-May-2023], 2016.

[75] *School vr subjects: Chemistry science resources*, https://www.classvr.com/vr-ar-resources/science-chemistry-vr-teaching-resources/, [Online; accessed 01-May-2023], 2023.

[76] *Holoanatomy software suite*, https://case.edu/holoanatomy/, [Online; accessed 01-May-2023], 2023.

[77] *Hololens 2 hardware*, https://www.microsoft.com/en-us/hololens/hardware, [Online; accessed 01-May-2023], 2023.

[78] S. Rokhsaritalemi, A. Sadeghi-Niaraki, and S.-M. Choi, "A review on mixed reality: Current trends, challenges and prospects," *Applied Sciences*, 2020.

[79] *Spatial mapping*, https://learn.microsoft.com/en-us/windows/mixed-reality/design/spatial-mapping, [Online; accessed 01-May-2023], 2023.

[80] *Verifying device support and user permission*, https://developer.apple.com/documentation/arkit/verifying_device_support_and_user_permission, [Online; accessed 01-May-2023], 2023.

[81] *Enable arcore*, https://developers.google.com/ar/develop/java/enable-arcore, [Online; accessed 01-May-2023], 2023.

[82] J. Hu, A. Iosifescu, and R. LiKamWa, "Lenscap: Split-process framework for fine-grained visual privacy control for augmented reality apps," in *International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2021.

[83] S. Jana, A. Narayanan, and V. Shmatikov, "A scanner darkly: Protecting user privacy from perceptual applications," in *IEEE symposium on security and privacy (S&P)*, 2013.

[84] N. Raval, A. Srivastava, K. Lebeck, L. Cox, and A. Machanavajjhala, "Markit: Privacy markers for protecting visual secrets," in *ACM International joint conference on pervasive and ubiquitous computing*, 2014.

[85] R. Templeman, M. Korayem, D. J. Crandall, and A. Kapadia, "Placeavoider: Steering first-person cameras away from sensitive spaces.," in *Networks and Distributed Systems Security (NDSS)*, 2014.

[86] J. A. De Guzman, K. Thilakarathna, and A. Seneviratne, “Security and privacy approaches in mixed reality: A literature survey,” *ACM Computing Surveys (CSUR)*, 2019.

[87] Y. Kim, S. Boorboor, A. Rahmati, and A. Kaufman, “Design of privacy preservation system in augmented reality,” in *IEEE Symposium on Visualization for Cyber Security*, 2021.

[88] N. Raval, A. Srivastava, A. Razeen, K. Lebeck, A. Machanavajjhala, and L. P. Cox, “What you mark is what apps see,” in *International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2016.

[89] S. Jana *et al.*, “Enabling {fine-grained} permissions for augmented reality applications with recognizers,” in *USENIX Security Symposium*, 2013.

[90] F. Roesner, D. Molnar, A. Moshchuk, T. Kohno, and H. J. Wang, “World-driven access control for continuous sensing,” in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2014.

[91] R. A. Epstein and C. I. Baker, “Scene perception in the human brain,” *Annual review of vision science*, 2019.

[92] R. Caruana, “Multitask learning,” *Machine learning*, 1997.

[93] C. R. Qi, O. Litany, K. He, and L. J. Guibas, “Deep hough voting for 3d object detection in point clouds,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[94] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, “Pointnet: Deep learning on point sets for 3d classification and segmentation,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[95] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, “Dynamic graph cnn for learning on point clouds,” *ACM Transactions On Graphics (ToG)*, 2019.

[96] A. M. Kibriya and E. Frank, “An empirical comparison of exact nearest neighbour algorithms,” in *European conference on principles of data mining and knowledge discovery*, 2007.

[97] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *NeurIPS Deep Learning and Representation Learning Workshop*, 2015.

[98] P. Mukhopadhyay and B. B. Chaudhuri, "A survey of hough transform," *Pattern Recognition*, 2015.

[99] G. Y. Lu and D. W. Wong, "An adaptive inverse-distance weighting spatial interpolation technique," *Computers & geosciences*, 2008.

[100] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, 2019.

[101] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, 2015.

[102] *Cross entropy loss*, https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html, [Online; accessed 01-May-2023], 2023.

[103] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations (ICLR)*, 2014.

[104] Y. Zhang, T. Scargill, A. Vaishnav, G. Premsankar, M. Di Francesco, and M. Gorlatova, "Indepth: Real-time depth inpainting for mobile augmented reality," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2022.

[105] I. Chugunov, S.-H. Baek, Q. Fu, W. Heidrich, and F. Heide, "Mask-tof: Learning microlens masks for flying pixel correction in time-of-flight imaging," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[106] A. E. Johnson and M. Hebert, "Using spin images for efficient object recognition in cluttered 3d scenes," *IEEE Transactions on pattern analysis and machine intelligence*, 1999.

[107] M. A. Uy and G. H. Lee, "Pointnetvlad: Deep point cloud based retrieval for large-scale place recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[108] L. Yang *et al.*, "{Cade}: Detecting and explaining concept drift samples for security applications," in *USENIX Security Symposium*, 2021.

[109] D. Hendrycks and K. Gimpel, “A baseline for detecting misclassified and out-of-distribution examples in neural networks,” *International Conference on Learning Representations (ICLR)*, 2016.

[110] F. Pittaluga, S. J. Koppal, S. B. Kang, and S. N. Sinha, “Revealing scenes by inverting structure from motion reconstructions,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[111] R. Mohamed, H. Farrukh, Y. Lu, H. Wang, and Z. B. Celik, “Istelan: Disclosing sensitive user information by mobile magnetometer from finger touches,” *Proceedings on Privacy Enhancing Technologies*, 2023.

[112] Z. Ling, Z. Li, C. Chen, J. Luo, W. Yu, and X. Fu, “I know what you enter on gear vr,” in *IEEE Conference on Communications and Network Security (CNS)*, 2019.

[113] S. Luo, X. Hu, and Z. Yan, “Hologger: Keystroke inference on mixed reality head mounted displays,” in *IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, 2022.

[114] Ü. Meteriz-Yldran, N. F. Yldran, A. Awad, and D. Mohaisen, “A keylogging inference attack on air-tapping keyboards in virtual environments,” in *IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, 2022.

[115] A. Al Arafat, Z. Guo, and A. Awad, “Vr-spy: A side-channel attack on virtual key-logging in vr headsets,” in *IEEE Virtual Reality and 3D User Interfaces (VR)*, 2021.

[116] C. Shi *et al.*, “Face-mic: Inferring live speech and speaker identity via subtle facial dynamics captured by ar/vr motion sensors,” in *International Conference on Mobile Computing and Networking (MobiCom)*, 2021.

[117] A. Acar *et al.*, “Peek-a-boo: I see your smart home activities, even encrypted!” In *Conference on Security and Privacy in Wireless and Mobile Networks*, 2020.

[118] Y. Zhu *et al.*, “Et tu alexa? when commodity wifi devices turn into adversarial motion sensors,” in *Network and Distributed Systems Security (NDSS)*, 2020.

[119] R. Nandakumar, A. Takakuwa, T. Kohno, and S. Gollakota, “Covertband: Activity information leakage using music,” *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2017.

[120] S. Kumar, S. Gil, D. Katabi, and D. Rus, "Accurate indoor localization with zero start-up cost," in *Annual international conference on Mobile computing and networking*, 2014.

[121] S. Huang, M. Usvyatsov, and K. Schindler, "Indoor scene recognition in 3d," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[122] J. Lahoud, B. Ghanem, M. Pollefeys, and M. R. Oswald, "3d instance segmentation via multi-task metric learning," in *IEEE International Conference on Computer Vision (CVPR)*, 2019.

[123] Q.-H. Pham, T. Nguyen, B.-S. Hua, G. Roig, and S.-K. Yeung, "Jsis3d: Joint semantic-instance segmentation of 3d point clouds with multi-task pointwise networks and multi-value conditional random fields," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[124] X. Wang, S. Liu, X. Shen, C. Shen, and J. Jia, "Associatively segmenting instances and semantics in point clouds," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[125] Y. Ming, X. Yang, G. Zhang, and A. Calway, "Cgis-net: Aggregating colour, geometry and implicit semantic features for indoor place recognition," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022.

[126] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, 1981.

[127] "About face id advanced technology." (), [Online]. Available: https://support.apple.com/en-us/HT208108.

[128] *You should probably turn off the galaxy s10s face unlock if you value basic security.* [Online]. Available: https://www.androidauthority.com/galaxy-s10-face-unlock-insecure-964276/.

[129] H.-K. Jee, S.-U. Jung, and J.-H. Yoo, "Liveness detection for embedded face recognition system," *International Journal of Biological and Medical Sciences*, vol. 1, no. 4, pp. 235–238, 2006.

[130] S. Chen, A. Pande, and P. Mohapatra, “Sensor-assisted facial recognition: An enhanced biometric authentication system for smartphones,” in *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, ACM, 2014, pp. 109–122.

[131] K. Kollreider, H. Fronthaler, and J. Bigun, “Non-intrusive liveness detection by face images,” *Image and Vision Computing*, vol. 27, no. 3, pp. 233–244, 2009.

[132] X. Tan, Y. Li, J. Liu, and L. Jiang, “Face liveness detection from a single image with sparse low rank bilinear discriminative model,” in *European Conference on Computer Vision*, Springer, 2010, pp. 504–517.

[133] Y. Li, Y. Li, Q. Yan, H. Kong, and R. H. Deng, “Seeing your face is not enough: An inertial sensor-based liveness detection for face authentication,” in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2015, pp. 1558–1569.

[134] H. Hayakawa, “Photometric stereo under a light source with arbitrary motion,” *JOSA A*, vol. 11, no. 11, pp. 3079–3089, 1994.

[135] Y. Quéau, J. Durou, and J. Aujol, “Variational methods for normal integration,” *CoRR*, vol. abs/1709.05965, 2017. arXiv: 1709.05965. [Online]. Available: http://arxiv.org/abs/1709.05965.

[136] “Understanding gamma correction.” (), [Online]. Available: https://www.cambridgeincolour.com/tutorials/gamma-correction.htm.

[137] R. J. Woodham, “Photometric method for determining surface orientation from multiple images,” *Optical engineering*, vol. 19, no. 1, p. 191 139, 1980.

[138] L. N. Trefethen and D. Bau III, *Numerical linear algebra*. Siam, 1997, vol. 50.

[139] P. J. Burt and E. H. Adelson, “A multiresolution spline with application to image mosaics,” *ACM Transactions on Graphics (TOG)*, vol. 2, no. 4, pp. 217–236, 1983.

[140] G. Koch, R. Zemel, and R. Salakhutdinov, “Siamese neural networks for one-shot image recognition,” in *ICML deep learning workshop*, Lille, vol. 2, 2015.

[141] K. A. Dukes, "Gramschmidt process," in *Wiley StatsRef: Statistics Reference Online*. American Cancer Society, 2014, ISBN: 9781118445112. DOI: 10.1002/9781118445112.stat05633. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118445112.stat05633. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118445112.stat05633.

[142] A. J. Viterbi, *CDMA: Principles of Spread Spectrum Communication*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1995, ISBN: 0-201-63374-4.

[143] "Face++." (), [Online]. Available: https://www.faceplusplus.com/landmarks/.

[144] M. K. Agoston, *Computer Graphics and Geometric Modeling: Implementation and Algorithms*. Springer London, 2005, pp. 300–306.

[145] F. A. Jenkins and H. E. White, *Fundamentals of optics*. Tata McGraw-Hill Education, 1937.

[146] "Processing raw images in matlab." (), [Online]. Available: https://rcsumner.net/raw_guide/RAWguide.pdf.

[147] D. J. Berndt and J. Clifford, "Using dynamic time warping to find patterns in time series.," in *KDD workshop*, Seattle, WA, vol. 10, 1994, pp. 359–370.

[148] L. Fei-Fei, R. Fergus, and P. Perona, "One-shot learning of object categories," *IEEE transactions on pattern analysis and machine intelligence*, vol. 28, no. 4, pp. 594–611, 2006.

[149] Martn Abadi *et al.*, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: https://www.tensorflow.org/.

[150] "How does face recognition work on galaxy note10, galaxy note10+, and galaxy fold?" (), [Online]. Available: https://www.samsung.com/global/galaxy/what-is/face-recognition/.

[151] H. Chen, W. Wang, J. Zhang, and Q. Zhang, "Echoface: Acoustic sensor-based media attack detection for face authentication," *IEEE Internet of Things Journal*, 2020.

[152] Y. Li *et al.*, “A closer look tells more: A facial distortion based liveness detection for face authentication,” in *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, ser. Asia CCS 19, Auckland, New Zealand: Association for Computing Machinery, 2019, pp. 241–246, ISBN: 9781450367523. DOI: 10.1145/3321705.3329850. [Online]. Available: https://doi.org/10.1145/3321705.3329850.

[153] B. Zhou, J. Lohokare, R. Gao, and F. Ye, “Echoprint: Two-factor authentication using acoustics and vision on smartphones,” in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, ACM, 2018, pp. 321–336.

[154] Y. Chen, J. Sun, X. Jin, T. Li, R. Zhang, and Y. Zhang, “Your face your heart: Secure mobile face authentication with photoplethysmograms,” in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, 2017.

[155] K. Patel, H. Han, and A. K. Jain, “Secure face unlock: Spoof detection on smartphones,” *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 10, pp. 2268–2283, 2016.

[156] P. Lazik and A. Rowe, “Indoor pseudo-ranging of mobile devices using ultrasonic chirps,” in *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, 2012, pp. 99–112.

[157] W. Kim, S. Suh, and J.-J. Han, “Face liveness detection from a single image via diffusion speed model,” *IEEE transactions on Image processing*, vol. 24, no. 8, pp. 2456–2465, 2015.

[158] B. Lin, X. Li, Z. Yu, and G. Zhao, “Face liveness detection by rppg features and contextual patch-based cnn,” in *Proceedings of the 2019 3rd International Conference on Biometric Engineering and Applications*, ser. ICBEA 2019, Stockholm, Sweden: Association for Computing Machinery, 2019, pp. 61–68, ISBN: 9781450363051. DOI: 10.1145/3345336.3345345. [Online]. Available: https://doi.org/10.1145/3345336.3345345.

[159] H. Yu, T.-T. Ng, and Q. Sun, “Recaptured photo detection using specularity distribution,” in *2008 15th IEEE International Conference on Image Processing*, IEEE, 2008, pp. 3140–3143.

[160] K. Kollreider, H. Fronthaler, M. I. Faraj, and J. Bigun, “Real-time face detection and motion analysis with application in liveness assessment,” *IEEE Transactions on Information Forensics and Security*, vol. 2, no. 3, pp. 548–558, 2007.

[161] G. Pan, L. Sun, Z. Wu, and Y. Wang, “Monocular camera-based face liveness detection by combining eyeblink and scene context,” *Telecommunication Systems*, vol. 47, no. 3-4, pp. 215–225, 2011.

[162] T. I. Dhamecha, A. Nigam, R. Singh, and M. Vatsa, “Disguise detection and face recognition in visible and thermal spectrums,” in *Biometrics (ICB), 2013 International Conference on*, IEEE, 2013, pp. 1–8.

[163] A. Lagorio, M. Tistarelli, M. Cadoni, C. Fookes, and S. Sridharan, “Liveness detection based on 3d face shape analysis.,” in *IWBF*, 2013, pp. 1–4.

[164] A. J. Bose and P. Aarabi, *Adversarial attacks on face detectors using neural net based constrained optimization*, 2018. arXiv: 1805.12302 [cs.CV].

[165] Q. Song, Y. Wu, and L. Yang, *Attacks on state-of-the-art face recognition using attentional adversarial attack generative network*, 2018. arXiv: 1811.12026 [cs.CV].

[166] A. Kumar, N. Saxena, G. Tsudik, and E. Uzun, “A comparative study of secure device pairing methods,” *Pervasive and Mobile Computing*, 2009.

[167] I. Stellios, P. Kotzanikolaou, M. Psarakis, C. Alcaraz, and J. Lopez, “A survey of IoT-enabled cyberattacks: Assessing attack paths to critical infrastructures and services,” *IEEE Communications Surveys & Tutorials*, 2018.

[168] W. Zhou *et al.*, “Discovering and understanding the security hazards in the interactions between IoT devices, mobile apps, and clouds on smart home platforms,” in *USENIX Security*, 2019.

[169] R. Roman, J. Zhou, and J. Lopez, “On the features and challenges of security and privacy in distributed internet of things,” *Computer Networks*, 2013.

[170] E. Fernandes, J. Jung, and A. Prakash, “Security analysis of emerging smart home applications,” in *IEEE Symposium on Security and Privacy (S&P)*, 2016.

[171] *Openthread*, ’https://openthread.io/’, [Online; accessed 30-Jul-2022], 2022.

[172] J. Zhang, Z. Wang, Z. Yang, and Q. Zhang, “Proximity based IoT device authentication,” in *IEEE Conference on Computer Communications (INFOCOM)*, 2017.

[173] T. Zhang *et al.*, "Tap-to-pair: Associating wireless devices with synchronous tapping," *ACM Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2018.

[174] M. K. Chong, R. Mayrhofer, and H. Gellersen, "A survey of user interaction for spontaneous device association," *ACM Computing Surveys (CSUR)*, 2014.

[175] M. Rostami, A. Juels, and F. Koushanfar, "Heart-to-heart (H2H) authentication for implanted medical devices," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2013.

[176] D. Schürmann and S. Sigg, "Secure communication based on ambient audio," in *IEEE Transactions on Mobile Computing*, 2011.

[177] S. Birnbach, S. Eberz, and I. Martinovic, "Peeves: Physical event verification in smart homes," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2019.

[178] S. Birnbach, S. Eberz, and I. Martinovic, "Haunted house: Physical smart home event verification in the presence of compromised sensors," *ACM Transactions on Internet of Things*, 2021.

[179] Z. B. Celik, G. Tan, and P. D. McDaniel, "IoTGuard: Dynamic enforcement of security and safety policy in commodity IoT.," in *NDSS*, 2019.

[180] M. O. Ozmen, X. Li, A. Chu, Z. B. Celik, B. Hoxha, and X. Zhang, "Discovering IoT physical channel vulnerabilities," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2022.

[181] M. O. Ozmen, R. Song, H. Farrukh, and Z. B. Celik, "Evasion attacks and defenses on smart home physical event verification," in *NDSS*, 2023.

[182] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley Interdisciplinary Reviews: Computational Statistics*, 2010.

[183] J. C. Bezdek, R. Ehrlich, and W. Full, "FCM: The fuzzy c-means clustering algorithm," *Computers & Geosciences*, 1984.

[184] M. A. Khan and K. Salah, "IoT security: Review, blockchain solutions, and open challenges," *Future Generation Computer Systems*, 2018.

[185] M. Steiner, G. Tsudik, and M. Waidner, "Diffie-hellman key distribution extended to group communication," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 1996.

[186] Y. Dodis, L. Reyzin, and A. Smith, "Fuzzy extractors: How to generate strong keys from biometrics and other noisy data," in *International Conference on the Theory and Applications of Cryptographic Techniques*, 2004.

[187] A. Juels and M. Wattenberg, "A fuzzy commitment scheme," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 1999.

[188] M. Abdalla, E. Bresson, O. Chevassut, and D. Pointcheval, "Password-based group key exchange in a constant number of rounds," in *International Workshop on Public Key Cryptography*, 2006.

[189] M. Abdalla and D. Pointcheval, "A scalable password-based group key exchange protocol in the standard model," in *International Conference on the Theory and Application of Cryptology and Information Security*, 2006.

[190] D. Fiore, M. I. G. Vasco, and C. Soriente, "Partitioned group password-based authenticated key exchange," *The Computer Journal*, 2017.

[191] H. Chan, A. Perrig, and D. Song, "Random key predistribution schemes for sensor networks," in *IEEE Symposium on Security and Privacy (S&P)*, 2003.

[192] L. Eschenauer and V. D. Gligor, "A key-management scheme for distributed sensor networks," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2002.

[193] J. Lee and D. R. Stinson, "Deterministic key predistribution schemes for distributed sensor networks," in *International Workshop on Selected Areas in Cryptography*, 2004.

[194] D. Hankerson, A. J. Menezes, and S. Vanstone, *Guide to elliptic curve cryptography*. Springer Science & Business Media, 2006.

[195] *Tsfresh*, https://tsfresh.readthedocs.io/en/latest/, [Online; accessed 15-Jul-2022], 2022.

[196] *Scikit-fuzzy*, https://pythonhosted.org/scikit-fuzzy/, [Online; accessed 15-Jul-2022], 2022.

[197] S. Aranganayagi and K. Thangavel, “Clustering categorical data using silhouette coefficient as a relocating measure,” in *IEEE International Conference on Computational Intelligence and Multimedia Applications*, 2007.

[198] R. Tibshirani, G. Walther, and T. Hastie, “Estimating the number of clusters in a data set via the gap statistic,” *Journal of the Royal Statistical Society*, 2001.

[199] C. Costello and P. Longa, “Fourq: Four-dimensional decompositions on a q-curve over the mersenne prime,” in *International Conference on the Theory and Application of Cryptology and Information Security*, 2015.

[200] J.-P. Aumasson, L. Henzen, W. Meier, and R. C.-W. Phan, “Sha-3 proposal blake,” *Submission to NIST*, 2008.

[201] Y. Nir and A. Langley, “ChaCha20 and Poly1305 for IETF protocols,” *Internet Engineering Task Force*, 2015.

[202] *Fourqlib*, https://github.com/microsoft/FourQlib, [Online; accessed 24-Jul-2022], 2022.

[203] *Blake2*, https://github.com/BLAKE2/libb2, [Online; accessed 24-Jul-2022], 2022.

[204] *Chachapoly*, https://github.com/grigorig/chachapoly, [Online; accessed 24-Jul-2022], 2022.

[205] *Zeromq*, https://zeromq.org/, [Online; accessed 24-Jul-2022], 2022.

[206] A. Ihler, J. Hutchins, and P. Smyth, “Adaptive event detection with time-varying poisson processes,” in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006.

[207] T. Mahmud, M. Hasan, A. Chakraborty, and A. K. Roy-Chowdhury, “A poisson process model for activity forecasting,” in *IEEE International Conference on Image Processing (ICIP)*, 2016.

[208] M. Rosulek and N. Trieu, “Compact and malicious private set intersection for small sets,” in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2021.

[209] H. Alemdar, H. Ertan, O. D. Incel, and C. Ersoy, "Aras human activity datasets in multiple homes with multiple residents," in *IEEE International Conference on Pervasive Computing Technologies for Healthcare and Workshops*, 2013.

[210] D. J. Cook, A. S. Crandall, B. L. Thomas, and N. C. Krishnan, "Casas: A smart home in a box," *Computer*, 2012.

[211] T. Van Kasteren, A. Noulas, G. Englebienne, and B. Kröse, "Accurate activity recognition in a home setting," in *International Conference on Ubiquitous Computing*, 2008.

[212] J. Mao, S. Zhu, and J. Liu, "An inaudible voice attack to context-based device authentication in smart IoT systems," *Journal of Systems Architecture*, 2020.

[213] T. Trippel, O. Weisse, W. Xu, P. Honeyman, and K. Fu, "Walnut: Waging doubt on the integrity of mems accelerometers with acoustic injection attacks," in *IEEE European Symposium on Security and Privacy (Euro S&P)*, 2017.

[214] Y. Tu, S. Rampazzi, B. Hao, A. Rodriguez, K. Fu, and X. Hei, "Trick or heat? manipulating critical temperature-based control systems using rectification attacks," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2019.

[215] C. Fu, Q. Zeng, and X. Du, "Hawatcher: Semantics-aware anomaly detection for appified smart homes," in *USENIX Security*, 2021.

[216] I. Giechaskiel and K. Rasmussen, "Taxonomy and challenges of out-of-band signal injection attacks and defenses," *IEEE Communications Surveys & Tutorials*, 2019.

[217] A. K. Sikder, L. Babun, H. Aksu, and A. S. Uluagac, "Aegis: A context-aware security framework for smart home systems," in *Annual Computer Security Applications Conference (ACSAC)*, 2019.

[218] Y. Zhang and K. Rasmussen, "Detection of electromagnetic interference attacks on sensor systems," in *IEEE Symposium on Security and Privacy (S&P)*, 2020.

[219] J. M. McCune, A. Perrig, and M. K. Reiter, "Seeing-is-believing: Using camera phones for human-verifiable authentication," in *IEEE Symposium on Security and Privacy (S&P)*, 2005.

[220] C. Li, X. Ji, B. Wang, K. Wang, and W. Xu, “Sencs: Enabling real-time indoor proximity verification via contextual similarity,” *ACM Transactions on Sensor Networks (TOSN)*, 2021.

[221] S. Mathur, R. Miller, A. Varshavsky, W. Trappe, and N. Mandayam, “Proximate: Proximity-based secure pairing using ambient wireless signals,” in *International Conference on Mobile Systems, Applications, and Services*, 2011.

A. List of Publications

A.1 Conference Publications

C1 **Habiba Farrukh**, Reham Mohamed Aburas, Aniket Nare, Antonio Bianchi, and Z. Berkay Celik, "Inferring Semantic Location from Spatial Maps in Mixed Reality", USENIX Security Symposium, 2023.

C2 **Habiba Farrukh***, Muslum Ozgur Ozmen*, Faik Kerem Ors, and Z. Berkay Celik, "One Key to Rule Them All: Secure Group Pairing for Heterogeneous IoT Devices", IEEE Security and Privacy (S&P), 2023.

C3 Reham Mohamed Aburas, **Habiba Farrukh**, He Wang, Yidong Lu, and Z. Berkay Celik, "Disclosing Sensitive User Information by Mobile Magnetometer from Finger Touches", Privacy Enhancing Technologies (PoPETs), 2023.

C4 Muslum Ozgur Ozmen, Ruoyu Song, **Habiba Farrukh**, and Z. Berkay Celik "Evasion Attacks on Smart Home Physical Event Verification and Defenses", Network and Distributed System Security Symposium (NDSS), 2023.

C5 Abdullah Imran, **Habiba Farrukh**, Muhammad Ibrahim, Z. Berkay Celik, and Antonio Bianchi, "SARA: Secure Android Remote Authorization", USENIX Security Symposium, 2022.

C6 Siddharth Divi, Yi-Shan Lin, **Habiba Farrukh**, and Z. Berkay Celik, "New Metrics to Evaluate the Performance and Fairness of Personalized Federated Learning", International Workshop on Federated Learning for User Privacy and Data Confidentiality, co-located with International Conference on Machine Learning (ICML), 2021.

C7 **Habiba Farrukh**, Tinghan Yang, Hanwen Xu, Yuxuan Yin, He Wang, and Z. Berkay Celik, "S^3: Side-channel attack on Stylus Pencils through Sensors", Proceedings of the ACM Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT /UbiComp), 2021.

C8 **Habiba Farrukh**, Reham Aburas, Siyuan Cao, and He Wang, "FaceRevelio: A Face Liveness Detection System for Smartphones with a Single Front Camera", Proceedings of the ACM International Conference on Mobile Computing and Networking (MobiCom), 2020.

C9 Siyuan Cao, **Habiba Farrukh**, and He Wang, "Towards Context Address for Camera-to-Human Communication", IEEE International Conference on Computer Communications (InfoCom), 2020.

C**10** Siyuan Cao, **Habiba Farrukh**, and He Wang, "Demo: Enabling Public Cameras to Talk to the Public", Proceedings of the ACM International Conference on Mobile Systems, Applications, and Services (MobiSys), 2018.

A.2 Workshop/Symposium Publications

W1 Muslum Ozgur Ozmen*, **Habiba Farrukh***, Hyungsub Kim, Antonio Bianchi, and Z. Berkay Celik, "Rethinking Secure Pairing in Drone Swarms", ISOC Symposium on Vehicle Security and Privacy (VehicleSec), 2023.

∗ denotes equal contribution